Fall 2014

Streaming Algorithms Via Reductions

Michael S. Crouch
University of Massachusetts - Amherst

STREAMING ALGORITHMS VIA REDUCTIONS

A Dissertation Presented

by

MICHAEL STEVEN CROUCH

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2014

Computer Science

STREAMING ALGORITHMS VIA REDUCTIONS

A Dissertation Presented

by

MICHAEL STEVEN CROUCH

Approved as to style and content by:

To my parents, for their tireless support and unwavering love.

ACKNOWLEDGEMENTS

We thank an anonymous reviewer for mentioning that ideas from the pattern matching result in Ergün et al. [41] can be applied to cyclic shifts in the time-series model without requiring a second pass (§2.5). We also thank Graham Cormode for suggesting a simplification to the cyclic shift algorithm presented in §2.5.2.

ABSTRACT

STREAMING ALGORITHMS VIA REDUCTIONS

SEPTEMBER 2014

MICHAEL STEVEN CROUCH

B.S., CARNEGIE MELLON UNIVERSITY

B.S., CARNEGIE MELLON UNIVERSITY

M.S., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Andrew McGregor

In the *streaming algorithms* model of computation we must process data "in order" and without enough memory to remember the entire input. We study reductions between problems in the streaming model with an eye to using reductions as an algorithm design technique. Our contributions include

- "Linear Transformation" reductions, which compose with existing linear sketch techniques. We use these for small-space algorithms for numeric measurements of distance-from-periodicity, finding the period of a numeric stream, and detecting cyclic shifts.

- The first streaming graph algorithms in the "sliding window" model, where we must consider only the most recent L elements for some fixed threshold L. We develop basic algorithms for connectivity and unweighted maximum matching, then develop a variety of other algorithms via reductions to these problems.

- A new reduction from maximum weighted matching to maximum unweighted matching. This reduction immediately yields improved approximation guarantees for maximum weighted matching in the semistreaming, sliding window, and MapReduce models, and extends to the more general problem of finding maximum independent sets in p-systems.

- Algorithms in a "stream-of-samples" model which exhibit clear sample vs. space tradeoffs. These algorithms are also inspired by examining reductions. We provide algorithms for calculating F_k frequency moments and graph connectivity.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

The advent of "big data" requires us to develop new algorithmic techniques and to contend with new restrictions in our computational models. The "streaming algorithms" model captures two of the most important of these restrictions. In this model, we have only one-way access to a very large input, and we do not have enough random-access memory space to store the entire input at once.

Within the umbrella of "streaming algorithms" there are a wide variety of specific models, techniques, and problems. Some streaming algorithms are very general; algorithms which can be phrased as linear "sketches" of the data, for instance, are applicable not only in single-processor settings, but in parallel settings, distributed sensor networks, and compressed sensing applications. Other algorithms may be applicable only in a specific model (e.g. when processing is done by a single processor), or for a restricted class of inputs (e.g. non-negative numbers).

In algorithmic development within all of the streaming settings, *reductions between problems* have played a central role. A reduction from problem A to problem B is simply a "computationally easy" way of solving A, given an efficient way of solving instances of problem B. It is a way of using B "as a subroutine" to do "most of the work" involved in A. Of course, using one problem as a subroutine in solving another is one of the most fundamental ideas in algorithm development, but in the field of streaming algorithms these subroutines are often particularly clear, and many different problems are often solved by reductions to a few important "primitives".

Streaming reductions have played a vital role in the development of new streaming algorithms, and in adapting existing algorithms to new computational models. Many results in the streaming literature are already phrased in terms of informal "reductions" between problems. Unfortunately, their study as formal objects has lagged behind their application.

In this thesis, we present new algorithms for problems in data stream models. These algorithms are primarily inspired by examining reductions which preserve streaming resources (particularly memory space and per-item processing time).

1.1 Notation

We write $[n]$ for the set $\{0, 1, 2, \ldots, n-1\}$.

For functions $f(n), g(n)$, we write $f(n) = \tilde{O}(g(n))$ when there is a constant k such that $f(n) = O(g(n) \log^k n)$. When $f(n) = O(\log^k n)$ for some constant $k \geqslant 0$ we will say that $f(n)$ is $O(\operatorname{polylog} n)$, or simply that $f(n)$ is $\operatorname{polylog} n$.

We denote vectors in boldface, e.g. $\mathbf{a} = a_1 a_2 \ldots a_n$. Unless otherwise noted, when a vector is length n, we assume that each element of the vector can be represented by $O(\log n)$ bits. Thus, integers are bounded in size by some polynomial in n, and real numbers are represented to within some precision polynomial in n.

For a true/false statement φ, let $\mathbb{1}[\varphi]$ be the 0–1 indicator function which is 1 exactly when φ is true.

1.1.1 ℓ_p norms

The ℓ_p-norms, where $p \geqslant 0$, are a set of length measures for finite-length vectors. (When $p \in [0, 1)$, the measure is technically not a "norm", but is still well-defined). Given a length-n vector $\mathbf{x} = x_1 \ldots x_n$, we define

$$\|\mathbf{x}\|_p = \ell_p(\mathbf{x}) \triangleq \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p} \tag{1.1}$$

Particularly important norms include the ℓ_1 norm (or Manhattan distance), which gives us

$$\|\mathbf{x}\|_1 = \ell_1(\mathbf{x}) \triangleq \sum_{i=1}^{n} |x_i| \tag{1.2}$$

and the ℓ_2 norm (or Euclidean distance), which gives us

$$\|\mathbf{x}\|_2 = \|\mathbf{x}\| = \ell_2(\mathbf{x}) \triangleq \sqrt{\sum_{i=1}^{n} |x_i|^2}. \tag{1.3}$$

We define ℓ_0 as the number of nonzero elements; if we take $0^0 = 0$ we can write

$$\|\mathbf{x}\|_0 = \ell_0(\mathbf{x}) \triangleq \sum_{i=1}^{n} |x_i|^0 \tag{1.4}$$

Used as a distance measure, the ℓ_0 norm induces the Hamming distance.

We also define the *uniform norm*, *maximum norm*, or *ℓ_∞ norm* as

$$\|\mathbf{x}\|_\infty = \ell_\infty(\mathbf{x}) \triangleq \max_i |x_i| \tag{1.5}$$

As we might hope from the notation, we have

$$\ell_\infty(\mathbf{x}) = \lim_{p\to\infty} \|\mathbf{x}\|_p \tag{1.6}$$

Rather than the ℓ_p norms, work in the literature sometimes refers to the frequency moments F_p, defined by

$$F_p(\mathbf{a}) = \ell_p(\mathbf{a})^p \qquad p \neq 0 \tag{1.7}$$

$$F_0(\mathbf{a}) = \ell_0(\mathbf{a}) \tag{1.8}$$

1.2 The Streaming Model: History

The history of the streaming model has been well documented by several recent theses. We provide a summary of the relevant events here; the reader is recommended to examine McGregor 2007 [82, Chapter 1] and Nelson 2011 [93, Chapter 1] for further details.

Requiring that a Turing machine perform "on-line" processing of its inputs was a natural extension of early language-recognition problems, and was studied by several early papers [99, 79, 63]. Later, the class $\mathsf{1\text{-}L}$ was defined formally, comprising Turing machines with a logarithmic amount of workspace and an input head capable of moving in only one direction [58, 59].

Tantalizingly, $\mathsf{1\text{-}L}$ machines were introduced specifically *as a model of reductions*, in the hope of providing a "more refined tool for studying the feasible complexity classes" than polynomial-time or logarithmic-space reductions [58]. Unfortunately, streaming reductions proved a poor tool for studying non-streaming complexity; the "fine structure" of classes within polynomial time was more fruitfully studied by reductions defined in terms of small circuits (see, e.g., [102]) or simple logical formulas (see, e.g., [64]).

In the decades that followed several algorithms appeared which we would now recognize as being in the streaming model. In 1975, Morris developed a probabilistic algorithm for approximately counting m events using a register with space $O(\log \log m)$ (published in 1978 as [88]). In 1978, Munro and Paterson gave a two-pass algorithm for finding the median of m numbers with space $O(\sqrt{m} \log m)$ [90]. Flajolet and Martin described a single-pass probabilistic approximation algorithm for calculating the number of distinct elements in a list where possible elements range over the set $\{1, \ldots, d\}$ with space $O(\log d)$ bits of space in 1983 [48].

The importance of sublinear memory models became clearer as internet data began to outstrip the memory capacity of single machines. In 1996, Alon, Matias, and

Szegedy developed a small-space algorithm for estimating frequency moments of a data stream [4]; this algorithm has since been called the "tug-of-war sketch". The paper was awarded the Gödel Prize in 2005.

1.3 The Data Stream Model

In all models of streaming algorithms, we are interested in performing calculations without having enough memory to store the entire input, and with only one-way access to the input[1]. The restrictions in this model have led to many successes in analyzing streaming algorithms; in particular, explicit lower bounds are far more common in streaming algorithms than in classical computation.

The restrictions in our model come at a cost, however. Many problems are known to be feasibly solvable only in randomized approximation versions. The complexity of problems may vary depending on seemingly small choices about how the input is represented. In this section, we discuss input models and approximation guarantees.

For problems where our input is a vector of n numbers, many problems of interest can be solved using space $O(\operatorname{polylog} n)$. For problems where our input forms the edges of an n-node graph, there are $\Omega(n)$ lower bounds for many natural problems [45], but there are many interesting algorithms using space $O(n \operatorname{polylog} n)$.

1.3.1 Numeric Streams

For numeric problems, we are interested in calculating properties of some underlying vector $\mathbf{a} = a_1 \ldots a_n$. We will consider three different models of how the stream might define this vector, following the terminology of Muthukrishnan [91, pg. 12–13].

[1]For some applications, it is appropriate to consider algorithms which receive a small number of one-way "passes" over the data. Multi-pass algorithms are particularly appropriate for applications such as large database algorithms, where the choice to use streaming algorithms may reflect the better caching behavior of in-order memory access. In this work, however, we restrict ourselves to single-pass algorithms.

In the *time series* model, our input stream is simply the underlying vector, presented in order. We read elements a_1, a_2, $\ldots$, a_n. Slightly more general is the *permutation* model where coordinates of $\mathbf{a}$ may arrive out of order, i.e., $S = \langle(\pi(0), a_{\pi(0)}) \ldots (\pi(n-1), a_{\pi(n-1)})\rangle$, for some permutation π of $\{0, \ldots, n-1\}$.

The most general model we consider is the *turnstile* model, where instead of reading $\mathbf{a}$ directly, we read a series of "update operations" to $\mathbf{a}$ (which is initially the zero vector). The input stream consists of a series of m pairs $\langle k_1, z_1\rangle, \ldots, \langle k_m, z_m\rangle$, with $k_i \in [n]$ and $z_i \in \mathbb{R}$; informally each pair $\langle k, z\rangle$ has the meaning "increment a_k by z". We will often write these updates as $a_k \mathrel{+}= z$.

Note that a time series input can always be viewed as a permutation input where the inputs happen to occur in order[2]. Similarly, a permutation input can always be viewed as a turnstile input where each element happens to occur exactly once. Turnstile algorithms are thus the most general, since they can always be used to solve problems on permutation or time series inputs.

The converse is not true: there do exist problems which are harder in the turnstile model than the time series model. For example, calculating an ℓ_p norm $\|\mathbf{a}\|_p = (\sum_i |a_i|^p)^{1/p}$ is trivial in the time series or permutation models, because we can maintain a running total of $\sum_i |a_i|^p$. In the turnstile model, this is provably harder [4]; intuitively, the effect of an update $a_k \mathrel{+}= z$ on $\sum_i |a_i|^p$ depends on the previous value of a_k.

1.3.2 Linear Sketches

One of the basic classes of algorithms used in the streaming model is the *linear sketch.* Linear sketches work for the general case of turnstile inputs, and are common

[2]Technically, to turn a time series input into a permutation input we must also annotate each element with its position number. We can keep track of our current position using space $O(\log n)$, which will typically not affect our results.

in compressed sensing. A good review of linear sketch techniques is Gilbert / Indyk 2010 [51].

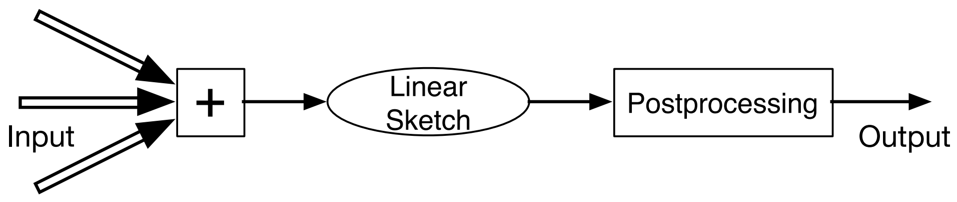
In a *linear sketch* algorithm we choose, perhaps randomly, some linear transformation T from $\mathbb{R}^n$ onto a space $\mathbb{R}^s$ with much smaller dimension ($s \ll n$). Storing the post-transformation vector $T\mathbf{a}$ then requires storing s elements.

$$\begin{bmatrix} \hat{a}_1 \\ \vdots \\ \hat{a}_k \end{bmatrix} = \begin{bmatrix} T_{11} & \cdots\cdots & T_{1n} \\ \vdots & & \vdots \\ T_{k1} & \cdots\cdots & T_{kn} \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ \vdots \\ \vdots \\ a_n \end{bmatrix} \tag{1.9}$$

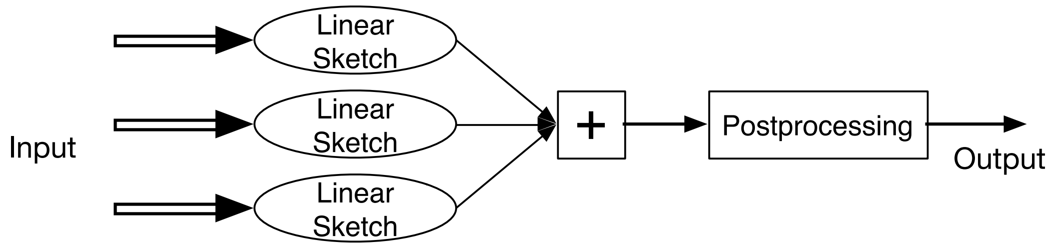
If our quantity of interest can be approximately reconstructed from $T\mathbf{a}$, this yields a natural turnstile algorithm. Letting $\widehat{e_k}$ denote the unit vector which is 1 in the coordinate k, an input update $a_k \mathrel{+}= z$ has the effect $\mathbf{a} \mathrel{+}= z\widehat{e_k}$. If we have been maintaining $T\mathbf{a}$, then by linearity we must simply take $T\mathbf{a} \mathrel{+}= zT\widehat{e_k}$.

Linearity is a powerful property which allows us to combine the sketches of multiple input streams to obtain a sketch of the total, enabling distributed processing (Figure 1.1). In a distributed sensor network, for example, each sensor could perform a sketch of its local observations, and communicate only that small sketch to a central processing node. The central node can sum the sketches together to obtain a sketch of the sum of all measurements.

For many functions, such as quantiles and heavy hitters [31], distinct items [72], and ℓ_1 and ℓ_2 norms [65], linear sketches exist where k is only polylogarithmic in n. Of course, it would still defeat the object of small-space computation if the algorithm needed to explicitly store a random $k \times n$ matrix. Instead the random matrices of interest are constructed either using limited independence or via a pseudo-random generator, e.g., Nisan [94]. Either way, the relevant entries can be reconstructed from some small seed as required.



(a)



(b)

Figure 1.1: Linearity allows distributed processing. Instead of sketching the sum of multiple streams (a), we can perform the sketch of each stream (b), and sum the sketches (which are of much lower dimension that the original signal).

1.3.3 Graph Streams

For graph data, we typically assume that there is an underlying set of n nodes (for known n) and that the input stream defines the edge relation on the graph. Almost all of the previous work on processing graph streams considered what is sometimes referred to as the *partially-dynamic* case, where the stream consists simply of a sequence of edges $\langle e_1, e_2, e_3, \ldots \rangle$, and the graph being monitored consists of the set of edges that have arrived so far. In other words, the graph is formed by a sequence of edge insertions. Over the last decade, it has been shown that many interesting problems can be solved using $O(n \operatorname{polylog} n)$ space, where n is the number of nodes in the graph. This is referred to as the *semi-streaming* space restriction [45].

1.3.4 Approximation and Randomization

Approximation problems are particularly important in the streaming setting, where many problems provably cannot be solved exactly in small space. Consider the case where, given an input X, we are interested in some real-valued function $f(X)$ which is difficult to calculate exactly. We may still be able to create an algorithm which, given some approximation parameter ϵ, outputs some number $\widehat{f}(X)$ such that $\widehat{f}(X) \in (1 \pm \epsilon) f(X)$.

In a typical randomized approximation algorithm, we must choose two parameters before running the algorithm: an acceptable failure probability $\delta > 0$, and an approximation accuracy $\epsilon > 0$. Our algorithm uses more resources as δ and ϵ become smaller. A typical streaming algorithm might use space proportional to $O(\log \frac{1}{\delta})$ and $O(\text{poly } \frac{1}{\epsilon})$.

1.3.5 Sampling Problems

In *sampling problems*, each input $x \in \{0,1\}^n$ corresponds to some probability distribution D_x over poly(n)-bit strings. We are to output a sample drawn from this distribution. For example, given an input representing a graph, we might be asked to select a node at random, selecting each node with probability proportional to its degree.

We are most often interested in approximation versions of sampling problems: given an input x and a parameter $\epsilon > 0$, we are to output samples from some probability distribution D'_x with distance $\|D_x - D'_x\| \leqslant \epsilon$. The distance measure used may vary, and will be specified; typically measurements include the total variation distance and the ℓ_0 distance (the maximum probability difference for any atomic event).

1.3.5.1 Streaming Example: ℓ_p Sampling

A general sampling problem in the streaming setting is to choose an element of the underlying vector with probability proportional to its size (in some measure). In the *exact L_p sampling problem*, we are to output each index i with probability

$$\Pr[\text{sample } i] = \frac{a_i^p}{\|\mathbf{a}\|_p^p} \ . \tag{1.10}$$

We will typically consider the approximate case, which we will simply call *L_p sampling*. For a parameter ϵ, our samples must be output from some distribution which has, for each i,

$$\Pr[\text{sample } i] = \frac{a_i^p}{\|\mathbf{a}\|_p^p}(1 \pm \epsilon) \ . \tag{1.11}$$

1.4 Example Reduction in the Streaming Model

Many results in the streaming literature are already phrased in terms of informal "reductions" between problems. Formalizing these models of reductions has suggested new algorithms, and has helped us extend existing algorithms to new models of computation. In this section, we present an example of streaming reductions used in the literature to describe an algorithm.

1.4.1 Precision Sampling: ℓ_1-Sampling Reduces to Heaviest Hitter

Precision sampling was introduced in [7]. Some of our analysis will also follow [71], which found space improvements over the original algorithm and which extended the algorithm to the $p \in (0, 1)$ case. We will show a reduction from ℓ_1 sampling onto Heaviest Hitter; the analysis of the other reductions is similar.

The *Heaviest Hitter* problem is the problem of determining the location and magnitude of the largest element in a numeric stream. In the exact version of the Heaviest Hitter problem, we would wish to return the index i corresponding to the largest element a_i. An approximation version of Heaviest Hitter was introduced in [26] (where it was referred to as "ApproxTop"). If i is the index of the largest element a_i, then in the Heaviest Hitter problem we are responsible for outputting the index of any element j with $a_j > (1-\epsilon)a_i$. (It is acceptable to output any element which is "approximately" as large as the largest element).

Of relevance here is the fact that space-efficient streaming algorithms are well-known in the streaming model. Berinde et al. 2010 [19] presents provably space-optimal randomized algorithms, with a summary of existing work. The summary paper Cormode / Hadjieleftheriou 2009 [30] is very good for ℓ_1-heavy-hitters.

Approximate ℓ_1 sampling is the problem of returning an index i with some probability in the range

$$\Pr[\text{accept } i] \in \frac{|a_i|}{\|\mathbf{a}\|_1}(1 \pm \epsilon) \tag{1.12}$$

The Precision Sampling technique shows a reduction from approximate ℓ_1 sampling onto $O(\frac{1}{\epsilon})$ parallel copies of Heaviest Hitter. Each of the copies will have only probability $O(\epsilon)$ of returning a sample; however, conditioned on having returned a sample, the sample has been drawn from the correct distribution (1.12). By running $O(\frac{1}{\epsilon})$ of these Heaviest Hitter calculations in parallel, we can guarantee that a sample is returned with high probability.

We will assume that $\frac{1}{2} \leqslant \|\mathbf{a}\|_1 \leqslant 1$. This is not a limitation, since we can use a standard sketch to store a value r with $\|\mathbf{a}\|_1 \leqslant r \leqslant 2\|\mathbf{a}\|_1$, and then consider the vector $\frac{1}{r}\mathbf{a}$. All of our algorithm can be scaled in this way during reconstruction.

Consider an input vector $\mathbf{a} = a_1 \ldots a_n$. For each $i \in [n]$, we choose (pairwise independently) some $u_i \in_R [0,1]$. We will use our heaviest-hitter structure to store the stream $\frac{a_i}{u_i}$ for each i. (Equivalently, when our reduction reads an input stream element $a_i \mathrel{+}= \Delta$, it outputs a stream update $a_i \mathrel{+}= \frac{\Delta}{u_i}$). At the end of the algorithm, we find a heaviest hitter j and the amplitude $\frac{a_j}{u_j}$. We will accept j as our sample iff we have $\frac{a_j}{u_j} \geqslant \frac{1}{\epsilon}$.

The analysis proceeds by analyzing indicator variables

$$s_i \triangleq \begin{cases} 1 & \frac{|a_i|}{u_i} \geqslant \frac{1}{\epsilon}\|\mathbf{a}\|_1 \\ 0 & \text{o.w.} \end{cases} \tag{1.13}$$

We have that

$$\begin{aligned} \Pr[s_i] &= \Pr\left[\frac{|a_i|}{u_i} \geqslant \frac{1}{\epsilon}\|\mathbf{a}\|_1\right] && (1.14) \\ &= \Pr\left[u_i \leqslant \epsilon\frac{|a_i|}{\|\mathbf{a}\|_1}\right] && (1.15) \\ &= \epsilon\frac{|a_i|}{\|\mathbf{a}\|_1} && (1.16) \end{aligned}$$

An item j is accepted as our sample iff we have s_j and we have no $s_{j'}$ for any $j' \neq j$. We thus obtain that

$$\epsilon(1-\epsilon)\frac{|a_i|}{\|\mathbf{a}\|_1} \leqslant \Pr[i \text{ accepted}] \leqslant \epsilon\frac{|a_i|}{\|\mathbf{a}\|_1} \tag{1.17}$$

and that

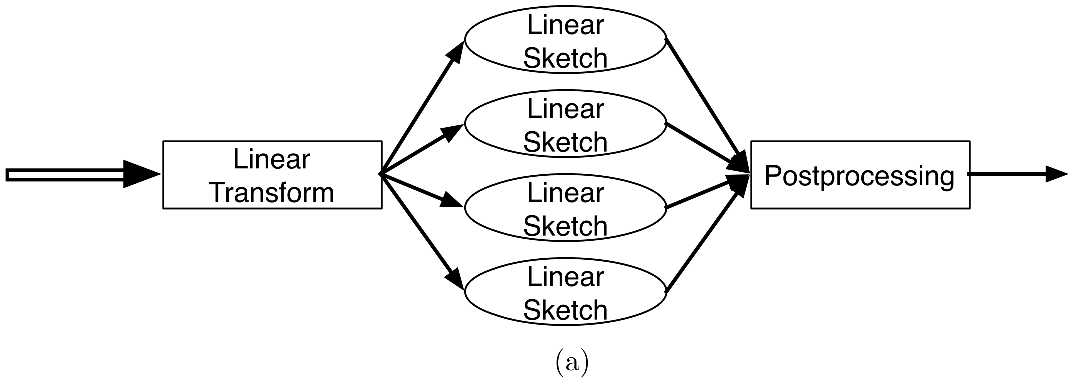
$$\Pr[\text{accept any}] \geqslant \epsilon \tag{1.18}$$

Running $O(1/\epsilon)$ copies of the algorithm in parallel thus gives us a sample from the desired distribution.

1.5 Contributions

1.5.1 Linear Transformation Reductions

Because many streaming sketch algorithms are linear, and because linear sketches have many desirable properties, a natural class of reductions to consider is *linear transformations*. Linear transformations are particularly desirable as reductions because they can be composed with linear sketch matrices (Figure 1.2). By applying the linear transformation to each sketch matrix, we obtain one matrix which performs the linear transformation *and* the sketch.



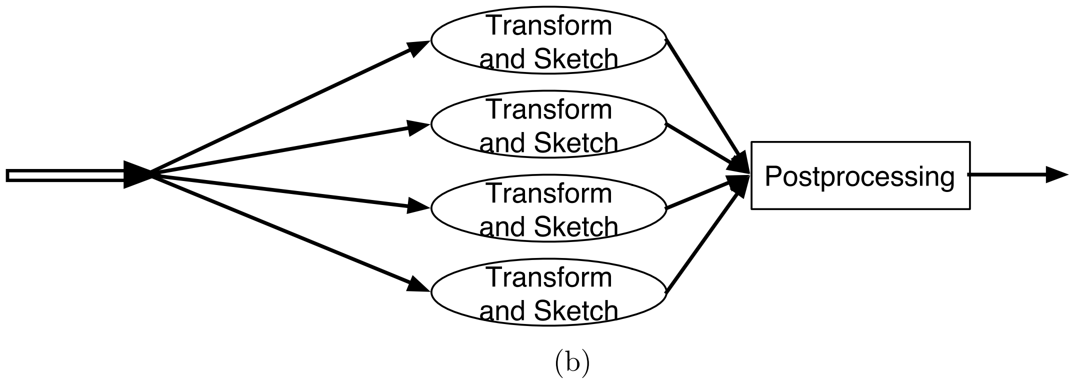


Figure 1.2: Linear sketches can be combined with linear transformation reductions. If a reduction's preprocessing step is a linear transformation (a), we can instead compose the linear transformation matrix with each sketch matrix (b).

Our work defines the class of linear transformation reductions, and uses them to find algorithms for problems relating to detecting repeating patterns in data streams.

1.5.1.1 Periodicity Results

In Chapter 2, we use linear transformation reductions to study the problem of identifying periodic trends in streams of numerical data. Previous work considered signals defined in the time series model; by using linear transformation reductions, we presented the first algorithms for identifying periodic trends in the turnstile model.

We say a signal $\mathbf{a} \in \mathbb{R}^n$ is *p-periodic* if it can be expressed as a concatenation $\mathbf{a} = \mathbf{x} \circ \ldots \circ \mathbf{x} \circ \mathbf{x}'$ for some $\mathbf{x} \in \mathbb{R}^p$ and some $\mathbf{x}' \in \mathbb{R}^{n-p\lfloor n/p \rfloor}$ that is a prefix of $\mathbf{x}$. Given a signal $\mathbf{a} \in \mathbb{R}^n$, we define the distance to p-periodicity as

$$\mathrm{D}_p(\mathbf{a}) \triangleq \min_{\mathbf{y}\ p\text{-periodic}} \|\mathbf{a} - \mathbf{y}\|_2 \tag{1.19}$$

where $\|\mathbf{v}\|_2 = \sqrt{v_1^2 + \cdots + v_n^2}$ denotes the ℓ_2 norm (§1.1.1) of the vector $\mathbf{v} \in \mathbb{R}^n$.

Theorem 2.4.2 presents an $O(\epsilon^{-2} \operatorname{polylog} n)$-space turnstile algorithm to approximate $D_p(\mathbf{a})$ for any p.

The problem of detecting approximate periodicity for unknown periods is more subtle to define. For example a signal which is close to being 3-periodic is at least as close to being 6-periodic. In §2.3.3 we present an algorithm which achieves a natural "gap promise" guarantee: given φ, ϵ with $0 < \varphi < \epsilon < 1$, it returns a period $p \mid n$ with

$$\mathrm{D}_p(\mathbf{a}) \leqslant \epsilon \|\mathbf{a}\|_2 \qquad \text{and} \qquad p \leqslant \min\{q \mid n : \mathrm{D}_q(\mathbf{a}) \leqslant (\epsilon - \varphi)\|\mathbf{a}\|_2\} \tag{1.20}$$

In §2.5, we consider the related problem of determining whether two sequences are approximate cyclic shifts of each other, e.g., given length-n sequences $\mathbf{a}, \mathbf{b}$ we wish to determine whether $\mathbf{b} = a_{s+1} a_{s+2} \cdots a_n a_1 \cdots a_s$. Defining the cyclic shift distance as the minimum ℓ_2 distance (§1.1.1) between $\mathbf{a}$ and any cyclic shift of $\mathbf{b}$, we present a randomized turnstile algorithm for $(1 + \epsilon)$-approximating cyclic shift distance in space $O(\epsilon^{-2}\sqrt{n} \operatorname{polylog} n)$ (Theorem 2.5.1).

1.5.2 Polylog-parallel Reductions

A more general reduction from problem A to problem B is to allow problem A to construct the inputs for some polylogarithmic number of parallel instances of problem B (Figure 1.3). This preserves polylogarthmic space and per-update time

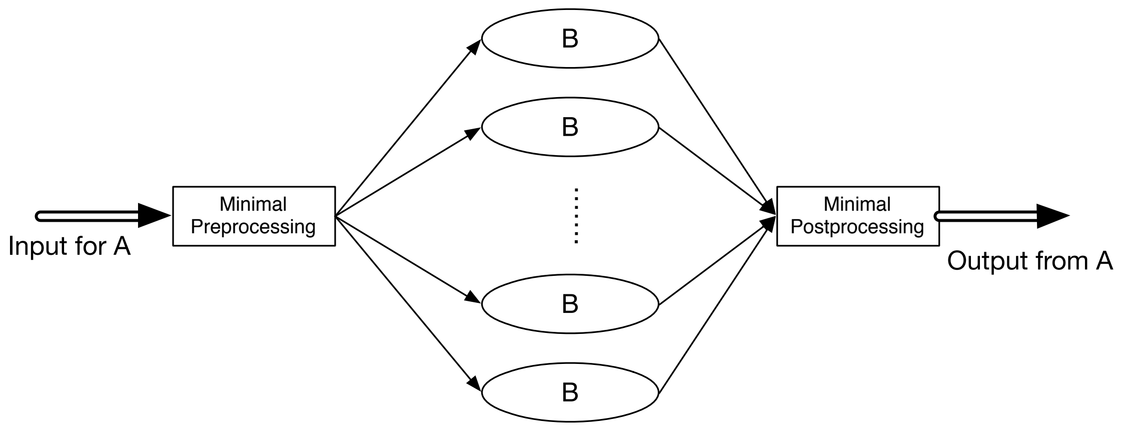


Figure 1.3: A parallel non-adaptive Turing reduction from problem A to problem B. We typically allow a polylogarithmic number of copies of B.

(since "polylogarithmic" is closed under composition). A version of these reductions appeared in Bar-Yossef / Kumar / Sivakumar 2002 [16], and many streaming algorithms are informally defined in terms of such reductions. Our contributions are to use these reductions explicitly in the creation of new algorithms in several streaming models.

1.5.3 Sliding Window Model

The sliding-window model, introduced by Datar et al. [35], has become a popular model for processing infinite data streams in small space when the goal is to compute properties of data that has arrived in the last window of time. Specifically, given an infinite stream of data $\langle a_1, a_2, \dots \rangle$ and a function f, at time t we need to return an estimate of $f(a_{t-L+1}, a_{t-L+2}, \dots, a_t)$. We refer to $\langle a_{t-L+1}, a_{t-L+2}, \dots, a_t \rangle$ as the *active* window where L is length of this window. The length of the window could correspond to hours, days, or years depending on the application. The motivation is that by ignoring data prior to the active window, we focus on the "freshest" data and can therefore detect anomalies and trends more quickly. Existing work has considered estimating various numerical statistics and geometric problems in this model [22, 21, 9, 46, 10, 12, 11], as well as developing useful techniques such as the *exponential histogram* [35] and *smooth histogram* data structures [22, 21].

Our work initiates the study of processing *graphs* in the sliding-window model, where the goal is to monitor the graph described by the last L entries of a stream of inserted edges. We present three basic algorithms in this model, and then use reductions to these algorithms to solve several other problems.

In §3.2 we present a deterministic sliding window algorithm which maintains the most recent k edges across every cut in a graph, using space $O(kn \log n)$. This algorithm allows us to directly solve the problem of network k-edge-connectivity, i.e., determining whether a graph would remain connected when any $k - 1$ edges are removed.

Using reductions to this algorithm, we solve several additional problems in the sliding window model. Theorem 3.2.3 presents a deterministic sliding-window algorithm for testing whether the graph formed by the most recent L edges is bipartite, using space $O(n \log n)$. Theorem 3.4.1 presents a deterministic sliding-window algorithm for maintaining a $(1 + \epsilon)$-approximate minimum spanning tree in weighted graphs, using space $O(\epsilon^{-1} n \log^2 n)$. Theorem 3.2.4 presents a randomized sliding-window algorithm which uses space $O(\epsilon^{-2} n \operatorname{polylog} n)$ to maintain a $(1 + \epsilon)$-sparsifier of the active window, i.e., a sparse weighted subgraph which maintains a $1 + \epsilon$ approximation of the weight across every cut.

The second fundamental problem we address in the sliding window model is maintaining a maximum-cardinality matching[3] in an unweighted graph. In §3.3, we adapt the powerful "smooth histograms" technique of Braverman and Ostrovsky [22, 21] to provide a sliding-window algorithm which maintains, at all times, a $(3 + \epsilon)$-

[3]A *matching* in a graph is a set of edges where no two edges have any vertex in common.

approximation of the maximum cardinality matching in the active window, using space $O(\epsilon^{-1} n \log^2 n)$ (Theorem 3.3.3).

We also present an algorithm for graph spanners: finding a sparse subgraph of the edges which approximately preserves the distance between any pair of points. Our algorithm provides a spanner of stretch $(2t-1)$ using space $O(L^{1/2} n^{(1+1/t)/2})$. We then generalize to provide a spanner of stretch $(2t-1)^k$ for integer $k \geqslant 1$, using space $O(Ln)$

1.5.4 Weighted Matching

The problem of finding maximum-weight matchings in the semi-streaming model has seen a great deal of attention since it was first introduced in [45], which gave a 6-approximation. This was improved to a 5.828-approximation in [81]; a 5.585-approximation in the [104]; and finally the current best, a $4.911+\epsilon$-approximation in [40]. Other work has looked at generalizations of matching to submodular-function matching [25], and at maximum weighted matching in the map reduce model [78].

In Chapter 4 we develop a reduction from maximum weighted matching to maximum unweighted matching. The structure of our reduction is related to [40], a streaming algorithm for maximum weighted matching. That algorithm partitions incoming edges into multiplicatively spaced weight classes and calculates a greedy matching on the edges within each weight class. At the end of the stream, they greedily merge the greedy matchings from largest to smallest.

Our algorithm uses weight classes that have lower bounds but not upper bounds, so classes that admit smaller edges are subsets of all the "more exclusive" classes. This gives rise to a better approximation and to a more broadly applicable proof, allowing us to address a generalization of the problem: finding maximum-weight independent sets in p-systems. p-systems are a type of independence system which generalize both matching on p-bounded hypergraphs and intersections of p matroids.

Problem	Model	Previous	Ref	This Work
MWM	One-pass streaming	4.911	[40]	4
MWM	One-pass sliding window	9.027	[33]	6
MWM	Map-reduce	8	[78]	4
3-MWM	One-pass streaming	9.899	[25]	9
2-MWIS	One-pass streaming	8	[25]	4
3-MWIS	One-pass streaming	9.899	[25]	9

Table 1.1: Approximation factor improvements over previous results. ϵ factors have been omitted. MWM is the "maximum weighted matching" problem; "MWIS" is the "maximum weighted independent set" problem.

The reduction works in the streaming model and in several related models, and improves the best known algorithms for many matching problems. These are summarized in Table 1.1.

1.5.5 Sampling vs. Space

In Chapter 5, we consider an alternative streaming model, where our input is a stream of independent samples (with replacement) from some underlying distribution. We examine the trade-offs between the number of samples we must take from the distribution and the amount of memory space allowed to process these samples. This provides a trade-off between two well-studied quantities:

1. The statistics question is how to bound the *sample complexity*: how many samples are required to estimate $f(D)$ to some prescribed accuracy with high probability?

2. The data stream question is how to bound the *space complexity*: how much memory is required to compute or approximate the estimator for $f(D)$?

The model can also been seen as an extension of the field of *sufficient statistics* [47] to the study of quantities which suffice to maintain *approximate* information about a distribution.

Similarly to the sliding windows model, we study this new model by developing a few algorithmic primitives and then using reductions to easily get additional algorithms.

We begin with a primitive for the F_2 moment (equivalently, the Euclidean norm of the input). We use reductions to this primitive to show that the F_k moment for $k \geqslant 2$ can be estimated using s space and t samples where

$$s \cdot t = \tilde{O}_\epsilon(n^{2-2/k}) \tag{1.21}$$

for any $t = \Omega(n^{1-1/k})$.

Similarly, we study a "random walk" sampling primitive, and use these walks to implement a graph connectivity tester. For a graph with n nodes and m edges, we show that connectivity can be determined with high probability using s space and t samples for

$$s^2 \cdot t = \tilde{O}(nm^2) \tag{1.22}$$

for any $t = \Omega(m \log m)$.

1.6 Other Reduction Models

Other work on defining streaming reductions has focused primarily on decision problems (as opposed to numeric-valued / "function" problems). We briefly recall this work here.

Magniez et al. 2010 [80] defines streaming reductions as a sort of generalization of string homomorphism; each character in the first stream is translated into some $f(n)$ characters on the second stream. Their focus is on language-membership problems over a finite alphabet, which only works well for decision problems. They describe probabilistic reductions, but do not examine approximation.

A modification of these reductions are used to examine streaming algorithms for recognizing various subclasses of the context-free languages in [14].

Ganguly [50] defines space-bounded stream automata, in the context of proving lower bounds for some deterministic streaming algorithms. This analysis uses a deterministic Turing machine with a two-way work tape and a one-way input tape. After reading the input it is responsible for writing some output onto the work tape. The work tape's space is only bounded during the input moves. In particular, after reading the end of the input, it's allowed unlimited space to process the work tape into the output.

1.7 Chernoff-Hoeffding Bounds

When a set of random variables is fully independent, we can typically obtain very strong bounds on how far the mean of those variables is likely to be from its expectation. There are a wide variety of these bounds in the literature; see particularly Dubhashi / Panconesi 2009 [36] for bounds useful to the analysis of algorithms. We follow their nomenclature and refer to the relatively simple form we use as the "Chernoff-Hoeffding bound".

Fully independent variables are "expensive" for our algorithms to use; unlike limited-independence variables, we can't save space by storing some small "seed". Luckily, the bounds we obtain are typically exponential in the number of variables used. Many streaming algorithms work via the approximate outline of:

1. Using limited independence, store an $O(\operatorname{polylog} n)$-bit seed for a polynomial number of random variables.

2. Use these random variables to obtain a single "atomic" estimator for some desired quality.

3. Use the Chernoff-Hoeffding bound to argue that only $O(\operatorname{polylog} n)$ fully independent copies of the atomic estimator are necessary to yield a good estimate of the desired quality.

Fact 1.7.1 (Chernoff-Hoeffding bound)**.** *Let* $X_1, X_2, \ldots, X_n$ *be* independent *random variables bounded by* $0 \leqslant X_i \leqslant 1$*, and let* $X = \sum_i X_i$*. Then for any* $t \geqslant 0$ *we have:*

$$\Pr\Big[X \leqslant \mathbf{E}[X] - t\Big] \leqslant e^{-2t^2/n} \tag{1.23}$$

$$\Pr\Big[X \geqslant \mathbf{E}[X] + t\Big] \leqslant e^{-2t^2/n} \tag{1.24}$$

(See [36, (1.6)] for reference).

From this, we obtain a bound which will be helpful in the analysis of several of our algorithms, on the number of repetitions necessary to distinguish between two Bernoulli distributions:

Lemma 1.7.2. *Fix known* p*,* q *with* $0 \leqslant p < q \geqslant 1$*. Assume we are drawing independent samples from a Bernoulli (0/1) distribution with some unknown success probability* r*; we are guaranteed that either* $r \leqslant p$ *or* $r \geqslant q$*, and we are to determine which is the case.*

In order to make this determination with error probability $\leqslant \delta$*, it will suffice to draw* $2\frac{\log \delta^{-1}}{(q-p)^2}$ *samples from the distribution.*

Proof. We proceed by taking n samples from the distribution. Let $X_1, X_2, \ldots, X_n$ be the results of these samples (thus $X_i \in \{0, 1\}$ for $1 \leqslant i \leqslant n$). Then let $X = \sum_i X_i$.

If $X \geqslant \frac{p+q}{2}n$, we will conclude $r \geqslant q$; otherwise, we will conclude $r \leqslant p$.

Consider first the case where we hold a coin with probability $r < p$ coin. Then $\mathbf{E}[X] \leqslant pn$. We make an error iff $X \geqslant \frac{p+q}{2}n$; using Fact 1.7.1 we have:

$$\begin{aligned}
\Pr[\text{fail}] &= \Pr\left[X \geqslant \frac{p+q}{2}n\right] && (1.25)\\
&= \Pr\left[X \geqslant pn + \frac{q-p}{2}n\right] && (1.26)\\
&\leqslant \exp\left(-2\frac{(q-p)^2 n^2}{2^2 n}\right) && (1.27)\\
&= \exp\left(-\frac{(q-p)^2 n}{2}\right) && (1.28)
\end{aligned}$$

Setting $\Pr[\text{fail}] \leqslant \delta$ then yields

$$\begin{aligned}
\exp\left(-\frac{(q-p)^2 n}{2}\right) &\leqslant \delta && (1.29)\\
\frac{(q-p)^2 n}{2} &\geqslant \log\frac{1}{\delta} && (1.30)\\
n &\geqslant \frac{2\log\frac{1}{\delta}}{(q-p)^2} && (1.31)
\end{aligned}$$

The $r \geqslant q$ case is symmetric. □

1.8 Organization

Chapter 2 describes algorithms for detecting periodicity and cyclic shifts in numeric streams, based primarily on linear transformation reductions. Chapter 3 introduces the first sliding window graph algorithms, primarily as reductions to the "primitive" problems of detecting connectivity and unweighted graph matching. Chapter 4 presents a new reduction from weighted graph matching to unweighted graph matching in detail, giving improved approximation guarantees in a variety of models. Chapter 5 examines basic algorithms in the sampling model for sample-vs.-space tradeoffs. Chapter 6 presents conclusions.

CHAPTER 2
PERIODICITY

In this chapter, we present a detailed example of novel streaming algorithms found via a linear transformation reduction. We develop algorithms for examining the periodicity of a streamed signal vector in both the time series and turnstile models (§1.3.1). Many questions about periodicity reduce to ℓ_2 distance and ℓ_2 sampling questions; the reduction in this case is typically the Discrete Fourier Transform. This work was originally done with Andrew McGregor and was published as [32].

2.1 Introduction

We consider the problem of identifying periodic trends in data streams. Motivated by applications in computational biology and data mining, there has recently been a series of papers related to finding such trends in large data sets [41, 42, 66, 34, 95]. We say a signal $\mathbf{a} \in \mathbb{R}^n$ is *p-periodic* if it can be expressed as a concatenation $\mathbf{a} = \mathbf{x} \circ \ldots \circ \mathbf{x} \circ \mathbf{x}'$ for some $\mathbf{x} \in \mathbb{R}^p$ and some $\mathbf{x}' \in \mathbb{R}^{n-p\lfloor n/p \rfloor}$ that is a prefix of $\mathbf{x}$. We say $\mathbf{a}$ is *perfectly p-periodic* if $\mathbf{a}$ is p-periodic and $p \mid n$. Given a signal $\mathbf{a} \in \mathbb{R}^n$, we define the distance to p-periodicity as

$$\mathrm{D}_p(\mathbf{a}) \triangleq \min_{\mathbf{y} \in P_{p,n}} \|\mathbf{a} - \mathbf{y}\|_2 \quad \text{where } P_{p,n} = \{\mathbf{y} \in \mathbb{R}^n : \mathbf{y} \text{ is } p\text{-periodic}\}$$

where $\|\mathbf{v}\|_2 = \sqrt{v_1^2 + \cdots + v_n^2}$ denotes the ℓ_2 norm (§1.1.1) of the vector $\mathbf{v} \in \mathbb{R}^n$. (In §2.2.2 we discuss our choice of distance measure and observe that many of our results still hold if an alternative measure is chosen.) We denote the minimum period of a signal $\mathbf{a} \in \mathbb{R}^n$ by

$$\text{period}(\mathbf{a}) = \min\{p : \mathbf{a} \text{ is } p\text{-periodic}\} .$$

Previous periodicity work has considered signals defined in the time series model (§1.3.1), e.g., the stream $\langle 1, 2, 3, 4 \rangle$ defines the signal $\mathbf{a} = [1, 2, 3, 4]$. However, we wish to consider a more general setting. For example, consider a sensor network in which each node is tasked with recording the times when certain local events occur. These records are forwarded through the network to some central node for processing. In this situation, there is no guarantee that the records are received in the order they were generated. Hence, we would need an algorithm that could identify patterns even if the records arive out of order. A yet more challenging example would be if each sensor monitors the local temperature at each time step and we are interested in identifying periodic trends in the average temperature. In this case, not only can records arrive out of order but the signal will be determined by the value of multiple records.

Next we examine estimating the period of a nearly-periodic sequence in the presence of noise. While a seemingly natural problem, defining the precise problem is subtle. For example, should we deem the noisy signal

$$\mathbf{a} = [1, 2, 3, 1, 2, 3.5, 1, 2, 3.1, 1, 2, 3.4] \tag{2.1}$$

to be 3-periodic, 6-periodic, or aperiodic? Our algorithm achieves a natural "gap promise" guarantee: given φ, ϵ with $0 < \varphi < \epsilon < 1$, it returns a period $p \mid n$ with

$$\mathrm{D}_p(\mathbf{a}) \leqslant \epsilon \|\mathbf{a}\|_2 \qquad \text{and} \qquad p \leqslant \min\{q \mid n : \mathrm{D}_q(\mathbf{a}) \leqslant (\epsilon - \varphi)\|\mathbf{a}\|_2\} \tag{2.2}$$

(Note that there is always such a p, since any length-n signal trivially has $\mathrm{D}_n(\mathbf{a}) = 0$.) In other words, we ensure that $\mathbf{a}$ is close to being perfectly p-periodic and that there is no $q \leqslant p$ such that $\mathbf{a}$ is "significantly closer" to being perfectly q-periodic.

All of our algorithms work in the turnstile model and are *sketch-based*. We discuss sketches in more detail in §1.3.2 but note here that one of their main advantages is that they work in a distributed setting where parts of the streams are monitored at different locations: after the stream concludes, it is sufficient to communicate only the sketches, as these can then be merged in order to estimate the global property of interest. This would enable data aggregation in the sensor network example outlined above.

2.1.1 Results and Related Work

We first examine reductions using the Discrete Fourier Transform. We obtain an $O(\epsilon^{-2} \operatorname{polylog} n)$-space algorithm (Theorem 2.3.1) for $(1+\epsilon)$-appoximating $D_p(\mathbf{a})$ for fixed p in the case of perfect periodicity (p divides n). This algorithm operates by reducing $D_p(\mathbf{a})$ to the problem of calculating an ℓ_2 norm; the reduction is the Discrete Fourier Transform, composed with a simple "filtering" matrix removing nonperiodic components.

By using the above algorithm as a subroutine, we are able to *determine* the period of a perfectly periodic noiseless signal (Theorem 2.3.3). This corresponds to reducing the problem of determining the period to $O(\log n)$ parallel ℓ_2 norm estimation problems, one for each prime or power-of-a-prime factor of n. This algorithm uses space $O(\operatorname{polylog} n)$. In contrast, an earlier paper by Ergün et al. [41] presents a single-pass, $O(\operatorname{polylog} n)$-space algorithm for computing period($\mathbf{a}$) in the time-series model. Our results generalize this result to the turnstile model although our algorithm in this case requires that $\mathbf{a}$ is perfectly periodic.

In §2.3.3 we discuss algorithms which use the Discrete Fourier Transform to reduce onto *sampling* questions instead of norm estimation questions. We first discuss an alternative algorithm for determining the period of a noiseless prfectly periodic signal (§2.3.3.2), then discuss how this extends to an algorithm which works for noisy signals and provides a natural "gap promise" guarantee (Theorem 2.3.4), using $\mathrm{poly}(\log n, \varphi^{-1})$ space. There is no analog in the recent Ergün et al. [41] paper but an earlier result [42] in the combinatorial property-testing model can be applied in the streaming setting if we may use $O(\sqrt{n}\,\mathrm{polylog}\, n)$ space.

In Theorem 2.4.2 we present an alternative $O(\epsilon^{-2}\,\mathrm{polylog}\, n)$ space algorithm that $(1+\epsilon)$-approximates $\mathrm{D}_p(\mathbf{a})$ for any given p (for this algorithm p need not divide the length of the sequence). This algorithm again works by reducing D_p to an ℓ_2 distance question, but this time via an averaging method rather than via the Discrete Fourier Transform. We show that the bilinear sketches of Indyk and McGregor [67] can be adapted to provide small per-update time as well as small memory usage. In contrast, an earlier paper by Ergün et al. [41] presented an algorithm using $O(\epsilon^{-5.5}\sqrt{p}\,\mathrm{polylog}\, n)$ space for estimating the Hamming distance to the nearest p-periodic signal.

We conclude with a simple sketch algorithm (Theorem 2.5.1) for the related problem of identifying when two sequences are cyclic shifts of one another. This algorithm uses $O(\epsilon^{-2}\sqrt{n}\,\mathrm{polylog}\, n)$ space and has the additional feature that it actually approximates how close the strings are to being cyclic shifts. Subsequent work [6] developed sketches which determined whether two sequences were within Hamming distance t of being cyclic shifts of each other using $O((t + D(n))\,\mathrm{polylog}\, n)$, where $D(n)$ is the number of divisors of n. This is less space, but their algorithm does not provide an approximation of the distance from being a cyclic shift.

2.1.2 Notation

Recall $[n] = \{0, 1, 2, \ldots, n-1\}$. In this chapter, we denote signals in lower-case bold and their corresponding Fourier transforms in upper-case bold. For a complex number $z \in \mathbb{C}$ we denote the real and imaginary parts by $\mathrm{Re}(z)$ and $\mathrm{Im}(z)$ respectively. Recall $\mathbb{1}[\varphi]$ is the 0–1 indicator function which is 1 whenever φ is true.

2.1.3 Precision

Throughout this chapter, we will assume that the values of the signals can be exactly stored with $1/\operatorname{poly}(n)$ precision. For example, this would be guaranteed in the turnstile model with a number of updates $m = \operatorname{poly}(n)$ and with each $\Delta_j \in \{-M, -M+1, \ldots, M-1, M\}$ for some $M = \operatorname{poly}(n)$. We also assume that the approximation parameters $\epsilon, \varphi, \delta$ satisfy $1/\epsilon, 1/\delta, 1/\varphi \in O(\operatorname{poly} n)$.

2.2 Fourier Preliminaries and Choice of Distance Function

In this section, we review the basic definition and properties of the discrete Fourier transform. We then discuss the utility of the transform in the context of linear sketch-based data stream algorithms. Finally, we discuss our choice of the ℓ_2 norm as a distance measure.

2.2.1 Discrete Fourier Transform and Sketches

Given a signal $\mathbf{a} \in \mathbb{R}^n$, the discrete Fourier transform of $\mathbf{a}$, denoted $\mathbf{A} = \mathcal{F}(\mathbf{a})$, is defined as

$$\mathbf{A} = (A_0, A_1, \ldots, A_{n-1}) \quad \text{where} \quad A_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} a_j e^{\frac{2\pi i}{n} jk} .$$

The following proposition states some standard properties that will be of use.

Proposition 2.2.1. *For any signal* $\mathbf{a} \in \mathbb{R}^n$,

1. $\mathbf{a}$ *is perfectly p-periodic iff* $(A_k \neq 0 \Rightarrow n/p \mid k)$.

2. $\|\mathbf{a}\|_2 = \|\mathbf{A}\|_2$ *(Parseval's identity).*

Of particular importance in the context of data streams is the fact that the transformation from $\mathbf{a}$ to $\mathbf{A}$ is a *linear* transformation, i.e.,

$$\mathbf{A}^T = V\mathbf{a}^T \text{ where } V \in \mathbb{C}^{n\times n} \text{ and } V_{kj} = \tfrac{1}{\sqrt{n}} e^{\frac{2\pi i}{n}kj} \text{ for } k, j \in [n]\ . \tag{2.3}$$

This is significant because many data stream algorithms are based on randomized linear projections called *linear sketches* (§1.3.2). In particular, we will use linear sketches for calculating the ℓ_2 norm of the input, from [4], and for ℓ_2 sampling, from [87].

We will make use of the simple, but very useful, observation that rather than estimating functions in the time domain, we may estimate these functions in the frequency domain by combining the change of basis matrix V with the sketch matrix W. For example, if the random sketch matrix $W \in \mathbb{R}^{k\times n}$ can be used to estimate the number of non-zero entries in $\mathbf{a}$ then the sketch matrix $WV \in \mathbb{C}^{k\times n}$ can be used to estimate the number of non-zero entries[1] in $\mathbf{A}$.

2.2.2 Choice of Distance Function

In the context of the Fourier transform and many signal processing applications, the natural measure of dissimilarity between two signals is the ℓ_2 norm of their difference. In contrast, Ergün and coauthors [41, 42] considered a measure based on the Hamming distance, $\mathrm{D}^0_p(\mathbf{a}) \triangleq \min_{\mathbf{y}\in P_{p,n}} \Delta(\mathbf{a},\mathbf{y})$ where $\Delta(\mathbf{a},\mathbf{y}) = |\{i \in [n] : a_i \neq y_i\}|$.

[1]To be precise, it is often necessary to separate real and imaginary parts of V. That is, we consider $W \in \mathbb{R}^{k\times 2n}$ and let $V \in \mathbb{R}^{2n\times n}$ have entries $V_{kj} = \cos(2\pi jk/n)$ for $k \in \{0,\ldots,n-1\}$ and $V_{kj} = \sin(2\pi jk/n)$ for $k \in \{n,\ldots,2n-1\}$. In calculating the ℓ_2 norm this causes no difficulties, but in other cases we may need to be careful. If we counted the number of nonzero entries of V, for example, we would find the total number of non-zero real parts and non-zero imaginary parts.

While different measures are suited to different applications, many of our algorithms can also be applied to approximate the Hamming distance, at least in the permutation model.

Suppose $\Sigma = \{\sigma_1, \ldots, \sigma_r\}$ and consider the mapping h from $\Sigma \to \{\mathtt{0}, \mathtt{1}\}^r$:

$$h(\sigma) = x_1 \ldots x_r \quad \text{where for each position } j, \quad x_j = \begin{cases} \mathtt{1} & \text{if } \sigma = \sigma_j \\ \mathtt{0} & \text{otherwise} \end{cases} .$$

The following lemma demonstrates that $\mathrm{D}_p^0(\mathbf{a})$ and $(\mathrm{D}_p(h(\mathbf{a})))^2/2$ are closely related. Hence, if each element of the sequence is first transformed using h (as is possible in the permutation model) then the Hamming distance to periodicity can be approximated via the ℓ_2 distance to periodicity. The approximation is by a factor close to 1 if the sequence is close to being p-periodic. Note that we would expect this to be the more relevant case in the sense that we would be measuring the distance from periodicity of a nearly-periodic sequence.

Lemma 2.2.2. *For any* $\mathbf{a} \in \Sigma^n$*, with* $\Sigma = \{\sigma_1, \ldots, \sigma_r\}$*, let* $T(\mathbf{a}) = (\mathrm{D}_p(h(\mathbf{a})))^2/2$*. Then we have,*

$$\tfrac{1}{2}\,\mathrm{D}_p^0(\mathbf{a}) \leqslant T(\mathbf{a}) \leqslant \mathrm{D}_p^0(\mathbf{a}) \ . \tag{2.4}$$

Furthermore, if $\mathbf{a}$ *is almost periodic in the sense that at least a* $1-\epsilon$ *fraction of the elements* $\{a_j, a_{j+p}, \ldots, a_{j+n-p}\}$ *are identical for each* $j \in [p]$*, then*

$$(1-\epsilon)\,\mathrm{D}_p^0(\mathbf{a}) \leqslant T(\mathbf{a}) \leqslant \mathrm{D}_p^0(\mathbf{a}) \ . \tag{2.5}$$

Proof. Let $d = n/p$. For $\sigma \in \Sigma$ and $j \in [p]$ define

$$\gamma_{\sigma,j} = |\{a_k = \sigma : p \text{ divides } k - j\}|/d \tag{2.6}$$

and let $\gamma_{\star,j} = \max_{\sigma\in\Sigma} \gamma_{\sigma,j}$. It follows from basic properties of ℓ_2 that

$$2T(\mathbf{a}) = \sum_{j\in[p]} \sum_{\sigma\in\Sigma} d\gamma_{\sigma,j}(1-\gamma_{\sigma,j})^2 + d(1-\gamma_{\sigma,j})\gamma_{\sigma,j}^2 = d\sum_{j\in[p]}(1-\sum_{\sigma\in\Sigma}\gamma_{\sigma,j}^2) \tag{2.7}$$

whereas $\mathrm{D}_0(\mathbf{a}) = d\sum_{j\in[p]}(1-\gamma_{\star,j})$. The RHS of the first bound follows because

$$\frac{1}{2} - \frac{1}{2}\sum_{\sigma\in\Sigma}\gamma_{\sigma,j}^2 \leqslant \frac{1}{2} - \frac{1}{2}\gamma_{\star,j}^2 \leqslant 1-\gamma_{\star,j} \ ,$$

where the last inequality follows from $(\gamma_{\star,j} - 1/2)^2 \geqslant 0$. The LHS of the first bound follows because

$$1-\gamma_{\star,j} = 1-\gamma_{\star,j}\sum_{\sigma\in\Sigma}\gamma_{\sigma,j} \leqslant 1-\sum_{\sigma\in\Sigma}\gamma_{\sigma,j}^2 \ .$$

However, if $\gamma_{\star,j} \geqslant 1-\epsilon$, this can be tightened to

$$1-\sum_{\sigma\in\Sigma}\gamma_{\sigma,j}^2 \geqslant 1-\gamma_{\star,j}^2 - (1-\gamma_{\star,j})^2 = 2(1-\gamma_{\star,j})\gamma_{\star,j} \geqslant 2(1-\gamma_{\star,j})(1-\epsilon) \ .$$

□

We can also relate $\mathrm{D}_p(\mathbf{a})$ to the ℓ_1 distance to the nearest p-periodic signal. For this, consider the alphabet $\Sigma = \{1,\ldots,t\}$, and use the mapping $h(\sigma) = x_1 \ldots x_t$ where $x_j = \mathbb{1}[\sigma \geqslant j]$. This mapping satisfies $\mathrm{D}_p^1(\mathbf{a}) = \mathrm{D}_{pt}^2(h(\mathbf{a}))$, but at the cost of increasing the length of the signal by a factor of t.

2.3 Reductions Using the Discrete Fourier Transform

2.3.1 Distance from Fixed Periodicity

We first present a small-space algorithm for measuring the distance between the signal and the closest (under the ℓ_2 norm) p-periodic sequence, for fixed p. If $\mathbf{a}$ is perfectly periodic with period p, then the Fourier transform $\mathbf{A} = \mathcal{F}(\mathbf{a})$ has at most p nonzero components. Letting $d = n/p$, we know by Prop. 2.2.1 that the only non-zero coordinates of $\mathbf{A}$ are A_{kd} for $k \in \{0, \ldots, p-1\}$. For the case of general $\mathbf{a}$, let $\mathbf{X}_p$ denote the restriction of $\mathbf{A}$ to the coordinates corresponding to a perfectly p-periodic signal, i.e.,

$$\mathbf{X}_p = (A_0, 0, \ldots, 0, A_d, 0, \ldots, 0, \ldots, A_{(p-1)d}, 0, \ldots, 0) \ .$$

In the frequency domain, $\mathbf{X}_p$ is the closest Fourier transform of a period-p vector to $\mathbf{A}$. By Parseval's theorem, $\mathcal{F}$ and $\mathcal{F}^{-1}$ preserve inner products and ℓ_2 distances. Therefore, $\mathcal{F}^{-1}(\mathbf{X}_p)$ is the p-periodic vector that is closest to $\mathbf{a}$ in the ℓ_2 distance. This implies that

$$\mathrm{D}_p(\mathbf{a}) = \|\mathbf{a} - \mathcal{F}^{-1}(\mathbf{X}_p)\|_2 = \|\mathbf{A} - \mathbf{X}_p\|_2 = \|\mathbf{Y}_p\|_2 = \sqrt{\sum_{d \nmid k} |A_k|^2} \ . \tag{2.8}$$

There is a linear transformation $V : \mathbb{R}^{2n \times 2n}$ which performs the Discrete Fourier Transform, and we can easily create a linear transformation

$$U \in \mathbb{R}^{2n \times 2n} \quad \text{where } U_{kj} = \begin{cases} 1 & \text{for } j = k \text{ and } d \nmid j \\ 0 & \text{otherwise} \end{cases} \tag{2.9}$$

which "filters out" all except the p-periodic components. We then have

$$\mathrm{D}_p(\mathbf{a}) = \|UV\mathbf{a}\|_2 \tag{2.10}$$

Tug-of-War sketches [4] provide a natural linear sketch for the ℓ_2 norm of a vector. Alon et al. showed that if the entries of a random vector $\mathbf{z} = z_0 \ldots z_{n-1} \in \{-1, 1\}^n$ are chosen with 4-wise independence then the random variable $T = \sum_{i=0}^{n-1} z_i a_i$ satisfies $\mathbf{E}[T^2] = \|\mathbf{a}\|_2^2$. They show that the estimator has sufficiently low variance that, by averaging $O(\epsilon^{-2} \log \delta^{-1})$ independent estimators, we can find a $(1+\epsilon)$ approximation for $\|\mathbf{a}\|_2^2$ with error probability $\leqslant \delta$.

Tug-of-War sketches are linear, and take the $2n$-dimensional input space onto a $k = O(\epsilon^{-2} \log \delta^{-1})$-dimensional sketch space (containing the independent estimators necessary for our desired error bound). We can view the sketching process as a matrix $W \in \mathbb{R}^{k \times 2n}$. Sketching $\mathrm{D}_p(\mathbf{a})$ thus requires that we maintain $WUV\mathbf{a}$. By viewing WUV as a single $k \times 2n$ sketch matrix, we obtain

Theorem 2.3.1. *There is a one-pass $O(\epsilon^{-2} \log \delta^{-1} \operatorname{polylog} n)$-space algorithm for $(1+\epsilon)$-estimating $\mathrm{D}_p(\mathbf{a})$ with error probability $\leqslant \delta$.*

2.3.2 Determining Perfect Periodicity: Noiseless Case

In this and the next section we consider *finding* the period of a sequence that is perfectly periodic. In this case, a possible approach to detecting periodicity with unknown period would be to use the above algorithm to test all factors $p \mid n$ and return the minimum p such that $\mathrm{D}_p(\mathbf{a}) = 0$ (it suffices to set $\epsilon = 1$ for this purpose). Unfortunately, in the worst case n may have $d(n) = O(\exp(\log n / \log \log n))$ factors [57, pp. 260–264] and therefore this approach would take too much time and space. However, a simple modification suffices: we check for periodicity at each prime or power-of-a-prime factor k of n. Define the set

$$K(n) = \{k : k \text{ divides } n \text{ and is the power of a prime}\} .$$

We first observe that $|K(n)| \leqslant O(\log n)$ (since each prime factor of n is at least 2, we have from the prime factorization $n = p_1^{r_1} p_2^{r_2} \dots p_t^{r_t}$ that $|K(n)| = \sum r_i \leqslant \log_2 n$). The following lemma demonstrates that testing periodicity for $p \in K(n)$ is sufficient to determine $\operatorname{period}(\mathbf{a})$:

Lemma 2.3.2. *For any $\mathbf{a} \in \mathbb{R}^n$ which is perfectly periodic,*

$$\operatorname{period}(\mathbf{a}) = \operatorname{GCD}(n/k : k \in K(n) \textit{ and } \mathbf{a} \textit{ is } n/k\textit{-periodic}) .$$

Proof. In this proof we will make use of two simple facts about periodicity: a) if $\mathbf{a}$ is p-periodic then n is kp-periodic for any $kp \mid n$; and b) that if $\mathbf{a}$ is both p_1-periodic and p_2-periodic then $\mathbf{a}$ is $\mathrm{GCD}(p_1, p_2)$-periodic. Let $p = \mathrm{period}(\mathbf{a})$ and consider the prime factorization of n,p, and $g = \mathrm{GCD}(n/k : k \in K(n)$ and $\mathbf{a}$ is n/k-periodic$)$

$$n = q_1^{r_1} \dots q_m^{r_m} \ , \quad p = q_1^{s_1} \dots q_m^{s_m} \ , \quad \text{and} \quad g = q_1^{t_1} \dots q_m^{t_m} \ . \tag{2.11}$$

It easy to see that for each j, $0 \leqslant s_j, t_j \leqslant r_j$. To prove the lemma we need to show that $s_j = t_j$ for each j.

Consider a fixed $j \in [m]$. Let $A = \{n/k : k \in K(n)$ and $\mathbf{a}$ is n/k-periodic$\}$ and $B = \{b_0, \dots, b_{r_j}\}$ where

$$b_k = n \frac{q_j^k}{q_j^{r_j}} = q_1^{r_1} \dots q_{j-1}^{r_{j-1}} q_j^k q_{j+1}^{r_{j+1}} \dots q_m^{r_m} \tag{2.12}$$

Then $b_k \in A$ for $s_j \leqslant k \leqslant r_j$ since these terms are multiples of p. On the other hand $b_k \notin A$ for $0 \leqslant k \leqslant s_j - 1$: otherwise $\mathrm{GCD}(b_k, p) < p$ and this contradicts the minimality of p. Note that all terms in $A \backslash B$ have $q_j^{r_j}$ as factor. Hence, $q_j^{s_j} \mid g$ but $q_j^{s_j+1} \nmid g$ and so $s_j = t_j$ as required. □

We can thus detect the minimum p for which $\mathbf{a}$ is perfectly p-periodic by running $|K| = O(\log n)$ parallel copies of the algorithm of §2.3.1. With $O(\log n)$ points of failure, we must ensure that each algorithm fails with probability at most $\delta/\log n$; this increases the space by a $\log\log n$ factor which is dominated by other factors in the analysis.

Theorem 2.3.3. *There is a single-pass, turnstile algorithm for computing* $\mathrm{period}(\mathbf{a})$ *of perfectly periodic strings that uses* $O(\mathrm{polylog}\, n)$ *space and update time.*

2.3.3 Determining Perfect Periodicity: Noisy Case

In this section, we present an algorithm for estimating the periodicity of a noisy signal. As a stepping stone to this result, we discuss an alternative approach for the noiseless case based on sampling. An advantage of the alternative approach is that it does not require the factorization of n to be computed thereby avoiding any (admittedly sublinear time) preprocessing. However, the guarantee achieved is weaker.

2.3.3.1 Fourier Sampling

If $\mathbf{a}$ is perfectly periodic with period p, then the Fourier transform $\mathbf{A} = \mathcal{F}(\mathbf{a})$ has at most p nonzero components. Letting $d = n/p$, we know by Prop. 2.2.1 that the only non-zero coordinates of $\mathbf{A}$ are A_{kd} for $k \in \{0, \ldots, p-1\}$. For the case of general $\mathbf{a}$, let $\mathbf{X}_p$ denote the restriction of $\mathbf{A}$ to the coordinates corresponding to a perfectly p-periodic signal, i.e.,

$$\mathbf{X}_p = (A_0, 0, \ldots, 0, A_d, 0, \ldots, 0, \ldots, A_{(p-1)d}, 0, \ldots, 0) \ .$$

In the frequency domain, $\mathbf{X}_p$ is the closest Fourier transform of a period-p vector to $\mathbf{A}$. By Plancherel's theorem, $\mathcal{F}$ and $\mathcal{F}^{-1}$ preserve inner products and ℓ_2 distances. Therefore, $\mathcal{F}^{-1}(\mathbf{X}_p)$ is the p-periodic vector that is closest to the original signal $\mathbf{a}$ in the ℓ_2 distance. This implies that

$$\mathrm{D}_p(\mathbf{a}) = \|\mathbf{a} - \mathcal{F}^{-1}(\mathbf{X}_p)\|_2 = \|\mathbf{A} - \mathbf{X}_p\|_2 = \|\mathbf{Y}_p\|_2 = \sqrt{\sum_{d \nmid k} |A_k|^2} \ . \tag{2.13}$$

Our algorithms in this section are based on combining the above relationship with a technique for sampling in the Fourier domain.

Monemizadeh and Woodruff [87] present a general approach for ℓ_p*-sampling in the time-domain*: for a signal $\mathbf{a} \in \mathbb{R}^n$ defined in the turnstile model, the goal here is to output k with probability in the interval

$$\left[(1-\alpha)\frac{|a_k|^p}{\ell_p^p(\mathbf{a})}, (1+\alpha)\frac{|a_k|^p}{\ell_p^p(\mathbf{a})}\right] \tag{2.14}$$

for some small user-defined parameter $\alpha > 0$. They show that this can be performed efficiently in space $\text{poly}(\alpha^{-1}\log n)$.[2] A series of later papers improves the space usage, culminating in algorithms using space $O(\epsilon^{-1}\log^2 n)$ for $p \in (0,1)$, $O(\epsilon^{-p}\log^2 n)$ for $p \in (1,2]$, and $O(\log(\epsilon^{-1})\epsilon^{-1}\log^2 n)$ for $p = 1$ [71].

For our purposes, rather than considering the time-series vector $\mathbf{a}$, we consider the vector

$$\mathbf{A}' = (\text{Re}(A_1), \ldots, \text{Re}(A_n), \text{Im}(A_1), \ldots, \text{Im}(A_n)) \in \mathbb{R}^{2n} . \tag{2.15}$$

defined by applying the appropriate Fourier transform matrix to the signal. If ℓ_2-sampling is performed on $\mathbf{A}'$ and we return the value modulo n, then the probability that k is returned is in the interval:

$$\left[(1-\alpha)\frac{|A_k|^2}{\|\mathbf{A}\|_2^2}, (1+\alpha)\frac{|A_k|^2}{\|\mathbf{A}\|_2^2}\right] , \tag{2.16}$$

because $\frac{\text{Re}(A_k)^2+\text{Im}(A_k)^2}{\sum_{i\in[n]}\text{Re}(A_j)^2+\text{Im}(A_j)^2} = \frac{|A_k|^2}{\|\mathbf{A}\|_2^2}$.

To perform this sampling we use the fact that the ℓ_p-sampling algorithms described above can be performed using a sketch matrix W and that there exists a matrix transformation $V \in \mathbb{R}^{2n\times n}$ that transforms any signal $\mathbf{a} \in \mathbb{R}^n$ into the corresponding $\mathbf{A}'$ vector. Hence, applying the sketch matrix WV allows us to sample from $\mathbf{A}'$ as required. We will show how to use this sampling in the next two sections.

[2]There is an additive error probability of n^{-C} for arbitrarily large constant C but this can be ignored in our subsequent analysis.

2.3.3.2 Application to the Noiseless Case

Suppose there is no noise and that $p = \mathrm{period}(\mathbf{a})$. Let the samples collected be $k_1, \ldots, k_w \in [n]$. We know from Prop. 2.2.1 that each sample $k_i = cd$ for some $c \in [p]$. Let $q = n/\operatorname{GCD}(k_1, \ldots, k_w, n)$. We have $q = p/c'$ for some $c' \mid p$. Next we will show that for sufficiently large w, with high probability, either $q = p$ or the sequence was nearly perfectly q-periodic. (For example, in the case of the sequence in Eq. (2.1), perhaps we return $q = 6$.)

Choose an approximation parameter $\varphi > 0$. Assume for contradiction that $q = p/c'$ for some $c' > 1$, but that $\mathrm{D}_q(\mathbf{a}) \geqslant \varphi\sqrt{1+\alpha}\|\mathbf{a}\|_2$. Summing over bins j, by appealing to Eq. (2.13), we have that

$$\sum_{n/q \nmid j} \frac{|A_j|^2}{\|\mathbf{A}\|_2^2} \;=\; \frac{1}{\|\mathbf{a}\|_2^2}\sum_{n/q \nmid j} |A_j|^2 \;=\; \frac{(\mathrm{D}_q(\mathbf{a}))^2}{\|\mathbf{a}\|_2^2} \;\geqslant\; \varphi^2(1+\alpha)\;. \tag{2.17}$$

Therefore, using the $(1+\alpha)$ approximation to ℓ_2-sampling, the probability that we return a sample that is not a multiple of n/q is at least φ^2. Taking $w = O(\varphi^{-2}\log(\delta^{-1}\log p))$ samples ensures that we find some sample that is not a multiple of n/q for all $O(\log p)$ prime factors q of p. Consequently, if the algorithm does not return the exact value of $\mathrm{period}(\mathbf{a})$, it returns a value $h \mid \mathrm{period}(\mathbf{a})$ such that the sequence was very close to being h-periodic with high probability.

2.3.3.3 Application to the Noisy Case

For noisy signals, a natural question is to find the smallest period p such that $\mathrm{D}_p(\mathbf{a}) \leqslant \epsilon\|\mathbf{a}\|_2$. Unfortunately, since $\mathrm{D}_p(\mathbf{a})$ could be just under $\epsilon\|\mathbf{a}\|_2$ while another value $q < p$ may have $\mathrm{D}_q(\mathbf{a})$ just larger than $\epsilon\|\mathbf{a}\|_2$, this is too much to hope for. Instead we consider two parameters ϵ, φ with $\epsilon > \varphi > 0$, and use a slight modification of the above approaches to accept some $p \mid n$ such that $\mathrm{D}_p(\mathbf{a}) \leqslant \epsilon\|\mathbf{a}\|_2$, and for no smaller q do we have $\mathrm{D}_q(\mathbf{a}) \leqslant (\epsilon - \varphi)\|\mathbf{a}\|_2$.

Our algorithm proceeds by taking samples of the Fourier coefficients as before. It then returns the smallest value $p \mid n$ such that at least $1-(\epsilon-\varphi/2)$ fraction of the samples are of Fourier coefficients $k = cn/p$. With probability at least $1-\delta$, we can guarantee that this condition is satisfied for all p with $\mathrm{D}_p(\mathbf{a}) \leqslant (\epsilon-\varphi)\|\mathbf{a}\|_2$, and by no p with $\mathrm{D}_p(\mathbf{a}) > \epsilon\|\mathbf{a}\|_2$; this requires $O(\varphi^{-2}\log\delta^{-1})$ samples by an application of Lemma 1.7.2.

Theorem 2.3.4. *For any ϵ, φ, δ, there exists a single-pass, $O(\mathrm{poly}(\log n, \varphi^{-1}))$-space turnstile algorithm which returns $p \mid n$ such that both of the following conditions are satisfied with high probability:*

1. $\mathrm{D}_p(\mathbf{a}) < \epsilon\|\mathbf{a}\|_2$
2. *There does not exist $q < p$ such that $q \mid n$ and $\mathrm{D}_q(\mathbf{a}) < (\epsilon-\varphi)\|\mathbf{a}\|_2$.*

2.4 Distance from Fixed Periodicity

In this section, we present an alternative algorithm for measuring the distance between the signal and the closest (under the ℓ_2 norm) p-periodic sequence, for fixed p. This algorithm has significantly faster update processing time than the previous algorithm, and has the advantage that it does not require that the length of the sequence is a perfect multiple of the periods considered.

For $p < n$, we write $n = dp + r$ where

$$d = \lfloor n/p \rfloor \quad \text{and} \quad r = n \bmod p \ .$$

Basic properties of the ℓ_2 norm imply that the p-periodic pattern that is ℓ_2-closest to a vector $\mathbf{a}$ is the arithmetic mean of length-p segments of the vector:

Fact 2.4.1. *For any sequence* $\mathbf{a} \in \mathbb{R}^n$*, let* $\mathbf{c} = \operatorname{argmin}_{\mathbf{y} \in P_{n,p}} \|\mathbf{a} - \mathbf{y}\|_2$ *be the p-periodic vector which is* ℓ_2*-closest to* $\mathbf{a}$*. Then* $\mathbf{c} = \mathbf{b} \circ \ldots \mathbf{b} \circ (b_0 b_1 \ldots b_{r-1})$ *where*

$$b_i = \begin{cases} \sum_{j=0}^{d} a_{i+jp}/(d+1) & \text{for } 0 \leqslant i < r \\ \sum_{j=0}^{d-1} a_{i+jp}/d & \text{for } r \leqslant i \leqslant p-1 \end{cases} .$$

With this explicit form for $\mathbf{c}$, we can again use the Tug-of-War sketches of [4] (described above in §2.3.1) to approximate $\mathrm{D}_p(\mathbf{a}) = \|\mathbf{a} - \mathbf{c}\|_2$. We choose 4-wise independently the entries of a random vector $\mathbf{z} = z_0 \ldots z_{n-1} \in \{-1, 1\}^n$ and construct the random variable $T = \sum_{i=0}^{n-1} z_i(a_i - c_i)$, which satisfies $\mathbf{E}[T^2] = \|\mathbf{a} - \mathbf{c}\|_2^2$. By averaging $O(\epsilon^{-2} \log \delta^{-1})$ independent copies of this estimator, we can find a $(1 + \epsilon)$ approximation for $\|\mathbf{a} - \mathbf{c}\|_2^2$.

Note that the value of each estimator T can easily be constructed in a streaming fashion: when the ith element of $\mathbf{a}$ is incremented by Δ we increment

$$T \mathrel{+}= \left(z_i - \sum_{j : i = j \bmod p} \frac{z_j}{|\{j : 0 \leqslant j \leqslant n-1, i = j \bmod p\}|} \right) \Delta \tag{2.18}$$

A naive implementation of this update method takes $\Omega(n/p)$ time per update. To avoid this we adapt the bilinear sketch method of Indyk and McGregor [67]. This technique was originally designed to detect correlations in data streams but we can exploit the structure of this sketch to reduce the update time. Intuitively, we will be taking advantage of the fact that $\mathbf{c}$ has only p independent entries. Rather than view $\mathbf{a}$ as a length n vector, we encode it as a $(d+1) \times p$ matrix A where $A_{ij} = a_{ip+j}$ if $ip + j \leqslant n - 1$ and $A_{ij} = b_j$ otherwise. Similarly let C be the $(d+1) \times p$ matrix where $C_{ij} = b_j$. E.g., for $n = 10$ and $p = 4$ we have the matrices

$$A = \begin{pmatrix} a_0 & a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 & a_7 \\ a_8 & a_9 & b_2 & b_3 \end{pmatrix} \qquad \text{and} \qquad C = \begin{pmatrix} b_0 & b_1 & b_2 & b_3 \\ b_0 & b_1 & b_2 & b_3 \\ b_0 & b_1 & b_2 & b_3 \end{pmatrix} .$$

Let $\mathbf{x} \in \{-1,1\}^p$ and $\mathbf{y} \in \{-1,1\}^{d+1}$ be random vectors whose entries are 4-wise independent. Indyk and McGregor extended the Alon et al. result to show that the outer product of $\mathbf{x}$ and $\mathbf{y}$ had sufficient randomness for a result similar to the Tug-of-War sketch. In our context, the result implies that if $T = \sum_{0 \leqslant i \leqslant d, 0 \leqslant j \leqslant p-1} x_j y_i (A_{ij} - C_{ij})$, then by appealing to Fact 2.4.1, we have that

$$\mathbf{E}\big[T^2\big] = \sum_{0 \leqslant i \leqslant d, 0 \leqslant j < p} (A_{ij} - C_{ij})^2 = \mathrm{D}_p^2(\mathbf{a})$$

and there is still sufficiently low variance for $O(\epsilon^{-2} \log \delta^{-1})$ parallel repetitions to be sufficient for constructing a $(1+\epsilon)$ approximation with probability $1-\delta$. We next show that each T can be constructed in only $O(1)$ update time. To do this, decompose T as

$$T = \sum_{\substack{0 \leqslant i \leqslant d \\ 0 \leqslant j < p}} x_j y_i A_{ij} - \sum_{\substack{0 \leqslant i \leqslant d \\ 0 \leqslant j < p}} x_j y_i C_{ij} = \sum_{\substack{0 \leqslant i \leqslant d \\ 0 \leqslant j < p}} x_j y_i A_{ij} - \left(\sum_{0 \leqslant i \leqslant d} y_i\right)\left(\sum_{0 \leqslant j < p} x_j b_j\right)$$

and define $T_1 = \sum_{0 \leqslant i \leqslant d, 0 \leqslant j < p} x_j y_i A_{ij}$ and $T_2 = \sum_{0 \leqslant j < p} x_j b_j$. Since $\sum_{0 \leqslant i \leqslant d} y_i$ can be computed in pre-processing, it suffices to compute T_1 and T_2. We initialize $T_1 = T_2 = 0$. As the stream is read T_1 and T_2 are updated in $O(1)$ time using the following rule: when the $(ip+j)$th entry of $\mathbf{a}$ is incremented by Δ,

$$T_1 \mathrel{+}= \left(x_j y_i + \mathbb{1}[j \geqslant r] \frac{x_j y_d}{d}\right)\Delta \quad \text{and} \quad T_2 \mathrel{+}= \left(\mathbb{1}[j < r] \frac{x_j}{d+1} + \mathbb{1}[j \geqslant r] \frac{x_j}{d}\right)\Delta$$

where $r = n \bmod p$ and $\mathbb{1}$ is the indicator function.

Theorem 2.4.2. $\mathrm{D}_p(\mathbf{a})$ *can be approximated up to a factor* $(1+\epsilon)$ *with probability* $1-\delta$ *using* $\tilde{O}(\epsilon^{-2})$ *space and* $\tilde{O}(\epsilon^{-2})$ *update time. The algorithm operates in the turnstile model using one pass.*

2.5 Cyclic Shifts

In this section, we consider the problem of identifying whether two sequences $\mathbf{a}, \mathbf{b} \in \Sigma^n$ are close to being cyclic shifts of each other. We will assume for convenience that $\Sigma \subset \mathbb{R}$. Let $\mathrm{CS}_s : \mathbb{R}^n \to \mathbb{R}^n$ be the function that "rotates" the input sequence by s positions, i.e.,

$$\mathrm{CS}_s(a_1 a_2 \ldots a_n) = a_{s+1} a_{s+2} \ldots a_n a_1 \ldots a_s \ .$$

Then $\mathbf{a}$ and $\mathbf{b}$ are cyclic shifts iff there exists s such that $\mathbf{b} = \mathrm{CS}_s(\mathbf{a})$.

Our goal is to recognize cyclic shifts using linear sketches. We first note that the analogous problem in the simultaneous communication model is rather straightforward. Supose Alice knows $\mathbf{a} \in \Sigma^n$ and Bob knows $\mathbf{b} \in \Sigma^n$. They can easily determine whether $\mathrm{CS}_s(\mathbf{a}) = \mathbf{b}$ for some s by each transforming $\mathbf{a}$ and $\mathbf{b}$ into some canonical form and then using an equality test. Specifically, consider an arbitrary ordering of the sequences in Σ^n. Alice generates the cyclic shift $\hat{\mathbf{a}}$ of $\mathbf{a}$ that is minimal under this ordering. Similarly, Bob generates the minimal cyclic shift $\hat{\mathbf{b}}$ of $\mathbf{b}$. Clearly $\hat{\mathbf{a}} = \hat{\mathbf{b}}$ iff $\mathbf{a}$ is a cyclic shift of $\mathbf{b}$. This can be verified with $O(\log n)$ communication using standard fingerprinting techniques such as Rabin fingerprints (see e.g. [89]).

Obviously such an approach is not possible in the data stream model. In the time-series model, existing work combined with simple observations leads to an efficient algorithm for determining if two sequences are cyclic shifts. We first review this before presenting a new streaming algorithm that is sketch-based and thus applies in the turnstile steaming model. Furthermore, it can estimate the distance of two sequences from being cyclic shifts.

2.5.1 Time-Series Model

In the time-series model, a one-pass $O(\operatorname{polylog} n)$-space algorithm follows from Ergün et al.'s extensions [41] of the pattern matching algorithm of Porat and Porat [95]. The algorithm works when one of the strings precedes the other, i.e.,

$S = \langle a_0, a_1, \ldots, a_{n-1}, b_0, b_1, \ldots, b_{n-1} \rangle$, or when the strings are interleaved, i.e., $S = \langle a_0, b_0, a_1, b_1, \ldots, a_{n-1}, b_{n-1} \rangle$. (It is actually sufficient for the elements of one sequence to always precede the corresponding elements of the other; e.g., the stream $S = \langle a_0, b_0, a_1, a_2, b_1, a_3, b_2, b_3 \rangle$ is acceptable.)

The pattern-matching algorithm of [41] uses a fingerprinting function Φ to maintain a series of exponentially-lengthening fingerprints $\varphi_j = \Phi(a_0 \ldots a_{2^j-1})$; by cleverly updating appropriate fingerprints of $\mathbf{b}$, they keep track of each match for φ_j which occurred within the last 2^j characters. When we reach the final character of $\mathbf{b}$, for each m such that $\Phi(b_m \ldots b_{m+2^j-1}) = \Phi(a_0 \ldots a_{2^j-1})$, we have access to the fingerprints $\Phi(b_0 \ldots b_{m-1})$, $\Phi(b_m \ldots b_{m+2^j-1})$, and $\Phi(b_{m+2^j} \ldots b_{n-1})$. By adjusting the fingerprints appropriately, we can determine whether there exists $m \in [n]$ such that

$$\Phi(a_0 \ldots a_{n-1}) = \Phi(b_m \ldots b_{m+2^j-1} b_{m+2^j} \ldots b_{n-1} b_0 \ldots b_{m-1}) \ .$$

2.5.2 Cyclic Shift Distance

In this section, we present a simple turnstile algorithm for estimating how close two sequences are to being cyclic shifts. We define the cyclic shift distance, CSD, between two strings as

$$\mathrm{CSD}(\mathbf{a}, \mathbf{b}) = \min_s \|\mathbf{a} - \mathrm{CS}_s(\mathbf{b})\|_2 \ .$$

Clearly, if $\mathbf{b}$ is a cyclic shift of $\mathbf{a}$ then $\mathrm{CSD}(\mathbf{a}, \mathbf{b}) = 0$.

The algorithm proceeds as follows: We will use two sets of candidate shifts, $S = \left\{0, 1, 2, \ldots, \lceil \sqrt{n} \rceil - 1\right\}$ and $T = \left\{\lceil \sqrt{n} \rceil, 2\lceil \sqrt{n} \rceil, 3\lceil \sqrt{n} \rceil, \ldots, \left(\left\lceil \frac{n}{\lceil \sqrt{n} \rceil} \right\rceil - 1\right)\lceil \sqrt{n} \rceil\right\}$. As we process the turnstile stream, we construct Tug-of-War sketches [4] of $\mathrm{CS}_s(\mathbf{a})$ and $\mathrm{CS}_t(\mathbf{b})$ for each $s \in S$, $t \in T$. Using $O(\epsilon^{-2} \log \frac{1}{\delta} \log n)$-sized sketches, this allows us to $(1+\epsilon)$-approximate $\| \mathrm{CS}_s(\mathbf{a}) - \mathrm{CS}_t(\mathbf{b})\|_2$ for each $s \in S$ and $t \in T$ with probability at least $1-\delta'$. Since for all r, s we have that $\|\mathbf{a} - \mathrm{CS}_s(\mathbf{b})\|_2 = \| \mathrm{CS}_r(\mathbf{a}) - \mathrm{CS}_{r+s}(\mathbf{b})\|_2$, these shifts suffice to $(1+\epsilon)$-approximate $\|\mathbf{a} - \mathrm{CS}_u(\mathbf{b})\|_2$ for each $u \in \{1, \ldots, n\}$.

Choosing $\delta' = \frac{\delta}{n}$, we have that each pair r, s is simultaneously a $(1+\epsilon)$-approximation with probability $\geqslant 1-\delta$. We then find:

$$\Pr\left[\left| \min_{s\in S, t\in T} \|\mathrm{CS}_s(\mathbf{a}) - \mathrm{CS}_t(\mathbf{b})\|_2 - \mathrm{CSD}(\mathbf{a},\mathbf{b})\right| \geqslant \epsilon\, \mathrm{CSD}(\mathbf{a},\mathbf{b})\right] \leqslant \delta \ . \tag{2.19}$$

Theorem 2.5.1. *There exists a single pass algorithm using space $\tilde{O}(\epsilon^{-2}\sqrt{n})$ that returns a $(1+\epsilon)$ approximation for* $\mathrm{CSD}(\mathbf{a},\mathbf{b})$ *with probability at least* $1-\delta$.

2.6 Conclusion

We presented one-pass data stream algorithms for detecting periodic sequences and cyclic shifts, and for measuring the distance to the closest periodic sequence or cyclic shift. Our principle goal was to minimize the space used, and all of our periodicity algorithms used $O(\operatorname{polylog} n)$ space. Our algorithms used a range of techniques including bilinear sketches and combining a Fourier change of basis transform with a range of sketching techniques. This second technique is particularly powerful and we would be surprised if it didn't have applications that were still to be discovered (either via the Fourier basis or other bases). An important future direction is analyzing the structure of the sketches formed by combining the transform and sketch matrices: among other things, this could lead to more time-efficient algorithms. Another question is to generalize our results in Sects. 2.3.2 and 2.3.3 to estimate the period of signals that conclude with a partial repetition. This was not an issue with time-series data since there would always be a point near the end of the stream where there had been an exact number of repetitions. In the turnstile model the issue is more complicated, but we are hopeful that a more involved analysis of the Fourier approach may yield results.

CHAPTER 3
SLIDING WINDOW GRAPH STREAMS

Massive graphs arise in any application where there is data about both basic entities and the relationships between these entities, e.g., web-pages and hyperlinks; papers and citations; IP addresses and network flows; phone numbers and phone calls; Tweeters and their followers. Graphs have become the de facto standard for representing many types of highly-structured data. Furthermore, many interesting graphs are dynamic, e.g., hyperlinks are added and removed, citations hopefully accrue over time, and the volume of network traffic between two IP addresses may vary depending on the time of day.

Consequently there is a growing body of work on designing algorithms for analyzing dynamic graphs. This includes both traditional data structures where the goal is to enable fast updates and queries [69, 39, 62, 61, 101] and data stream algorithms where the primary goal is to design randomized data structures of *sublinear size* that can answer queries with high probability [45, 3, 2, 81, 104, 40, 74]. This chapter focuses on the latter: specifically, processing graphs using sublinear space in the sliding-window model. Although the focus is not on update time, many of these algorithms can be made fast by using standard data structures.

Work in this chapter was originally done with Andrew McGregor and Daniel Stubbs, and was published as [33].

3.1 Introduction

3.1.1 Sliding-Window Model

The sliding-window model, introduced by Datar et al. [35], has become a popular model for processing infinite data streams in small space when the goal is to compute properties of data that has arrived in the last window of time. Specifically, given an infinite stream of data $\langle a_1, a_2, \ldots \rangle$ and a function f, at time t we need to return an estimate of $f(a_{t-L+1}, a_{t-L+2}, \ldots, a_t)$. We refer to $\langle a_{t-L+1}, a_{t-L+2}, \ldots, a_t \rangle$ as the *active* window where L is length of this window. The length of the window could correspond to hours, days, or years depending on the application. The motivation is that by ignoring data prior to the active window, we focus on the "freshest" data and can therefore detect anomalies and trends more quickly. Existing work has considered estimating various numerical statistics and geometric problems in this model [22, 21, 9, 46, 10, 12, 11], as well as developing useful techniques such as the *exponential histogram* [35] and *smooth histogram* data structures [22, 21].

3.1.2 Results

This work initiates the study of processing graphs in the sliding-window model, where the goal is to monitor the graph described by the last L entries of a stream of inserted edges. Note the following differences between this model and fully-dynamic model. In the sliding-window model the edge deletions are *implicit*, in the sense that when an edge leaves the active window it is effectively deleted but we may not know the identity of the deleted edge unless we store the entire window. In the case of fully-dynamic graph streams, the identity of the deleted edge is *explicit* but the edge could correspond to any of the edges already inserted but not deleted.

We present semi-streaming algorithms in the sliding-window model for various classic graph problems including testing connectivity, constructing minimum spanning trees, and approximating the size of graph matchings. We also present algorithms for

	Insert-Only	Insert-Delete	Sliding Window (here)
Connectivity	Deterministic[45]	Randomized[2]	Deterministic
Bipartiteness	Deterministic[45]	Randomized[2]	Deterministic
$(1+\epsilon)$-Sparsifier	Deterministic[1]	Randomized[3, 52]	Randomized
$(2t-1)$-Spanner	$n^{1+1/t}$ spc[18, 38]	None	$L^{1/2}n^{(1+1/t)/2}$ spc
Min. Span. Tree	Exact[45]	$(1+\epsilon)$-approx.[2]	$(1+\epsilon)$-approx.
Unweight. Match	2-approx.[45]	None	$(3+\epsilon)$-approx.
Weight. Match	4.911-approx.[40]	None	$(6+\epsilon)$-approx.

Table 3.1: Single-pass, semi-streaming results. All the above algorithms use $O(n \operatorname{polylog} n)$ space with the exception of the spanner constructions, where the space usage on the table ignores constant factors.

constructing graph synopses including *sparsifiers* and *spanners*. We say a subgraph H of G is a $(2t-1)$-spanner if:

$$\forall u, v \in V: \quad d_G(u,v) \leqslant d_H(u,v) \leqslant (2t-1)d_G(u,v)$$

where $d_G(u,v)$ and $d_H(u,v)$ denote the distance between nodes u and v in G and H respectively. We say a weighted subgraph H of G is a $(1+\epsilon)$ sparsifier if

$$\forall U \subset V: \quad (1-\epsilon)\lambda_G(U) \leqslant \lambda_H(u,v) \leqslant (1+\epsilon)\lambda_G(U)$$

where $\lambda_G(U)$ and $\lambda_H(U)$ denote the weight of the cut $(U, V \backslash U)$ in G and H respectively. A summary of our results can be seen in Table 3.1 along with the state-of-the-art results for these problems in the insert-only and insert/delete models.

3.2 Connectivity and Graph Sparsification

We first consider the problem of testing whether the graph is k-edge connected for a given $k \in \{1, 2, 3 \ldots\}$. Note that $k = 1$ corresponds to testing connectivity. To do this, it is sufficient to maintain a set of edges $F \subseteq \{e_1, e_2, \ldots, e_t\}$ along with the time-of-arrival $\text{toa}(e)$ for each $e \in F$ where F satisfies the following property:

- *Recent Edges Property.* For every cut $(U, V\backslash U)$, the stored edges F contain the most recent $\min(k, \lambda(U))$ edges across the cut where $\lambda(U)$ denotes the total number of edges from $\{e_1, e_2, \ldots, e_t\}$ that cross the cut.

Then, we can easily tell whether the graph on the active edges, $\langle e_{t-L+1}, e_{t-L+2}, \ldots, e_t\rangle$, is k-connected by checking whether F would be k-connected once we remove all edges $e \in F$ where $\text{toa}(e) \leqslant t - L$. This follows because if there are k or more edges among the last L edges across a cut, F will include the k most recent of them.

3.2.1 Algorithm

The following simple algorithm maintains a set F with the above property. The algorithm maintains k disjoint sets of edges $F_1, F_2, \ldots, F_k$ where each F_i is acyclic. Initially, $F_1 = F_2 = \ldots = F_k = \varnothing$ and on seeing edge e in the stream, we update the sets as follows:

1. Define the sequence $f_0, f_1, f_2, f_3, \ldots$ where $f_0 = \{e\}$ and for each $i \geqslant 1$, f_i consists of the oldest edge in a cycle in $F_i \cup f_{i-1}$ if such a cycle exists and $f_i = \varnothing$ otherwise. Since each F_i is acyclic, there will be at most one cycle in each $F_i \cup f_{i-1}$.

2. For $i \in \{1, 2, \ldots, k\}$,
$$F_i \leftarrow (F_i \cup f_{i-1})\backslash f_i$$

In other words, we add the new edge e to F_1. If it completes a cycle, we remove the oldest edge on this cycle and add that edge to F_2. If we now have a cycle in F_2, we remove the oldest edge on this cycle and add that edge to F_3. And so on. By using an existing data structure for constructing online minimum spanning trees [100], the above algorithm can be implemented with $O(k \log n)$ update time.

3.2.2 Analysis

Lemma 3.2.1. *$F = F_1 \cup F_2 \cup \ldots \cup F_k$ satisfies the Recent Edges Property.*

Proof. Fix some $i \in [k]$ and a cut $(U, V \backslash U)$. Observe that the youngest edge $y \in F_i$ crossing a cut $(U, V \backslash U)$ is never removed from F_i since its removal would require it to be the oldest edge in some cycle C. This cannot be the case since there must be an even number of edges in C that cross the cut and so there is another edge $x \in C$ crossing the cut. This edge must have been older than y since y was the youngest.

It follows that F_1 always contains the youngest edge crossing any cut, and by induction on i, the ith youngest edge crossing any cut is contained in $\bigcup_{j=1}^{i} F_j$. This is true because this edge was initially added to $F_1 \subseteq \bigcup_{j=1}^{i} F_j$, and cannot leave $\bigcup_{j=1}^{i} F_j$. For the ith youngest edge to be evicted from F_i, there would have to be an even younger crossing edge in F_i; however, inductively, any such edge is instead contained in the set of *earlier* matchings $\bigcup_{j=1}^{i-1} F_j$. $\square$

Theorem 3.2.2. *There exists a sliding-window algorithm for monitoring k-connectivity using $O(kn \log n)$ space.*

3.2.3 Applications: Bipartiteness and Graph Sparsification

3.2.3.1 Bipartiteness

To monitor whether a graph is bipartite, we run the connectivity tester on the input graph and also simulate the connectivity tester on the *cycle double cover* of the input graph. The cycle double cover $D(G)$ of a graph $G = (V, E)$ is formed by replacing each node $v \in V$ by two copies v_1 and v_2 and each edge $(u, v) \in E$ by the edges (u_1, v_2) and (u_2, v_1). Note that this transformation can be performed in a streaming fashion. Furthermore, $D(G)$ has exactly twice the number of connected components as G iff G is bipartite [2].

Theorem 3.2.3. *There exists a sliding-window algorithm for monitoring bipartiteness using $O(n \log n)$ space.*

3.2.3.2 Graph Sparsification

Using the k-connectivity tester as a black-box we can also construct a $(1+\epsilon)$-sparsifier following the approach of Ahn et al. [3]. The approach is based upon a result by Fung et al. [49] that states that sampling each edge e with probability $p_e \geqslant \min\{253\lambda_e^{-1}\epsilon^{-2}\log^2 n, 1\}$, where λ_e is the size of the minimum cut that includes e, and weighting the sampled edges by $1/p_e$ results in a $(1+\epsilon)$ sparsifier with high probability. To emulate this sampling without knowing λ_e values, we subsample the graph stream to generate sub-streams that define $O(\log n)$ graphs $G_0, G_1, G_2, \ldots$ where each edge is in G_i with probability 2^{-i}. For each i, we store the set of edges $F(G_i)$ generated by the k-connectivity algorithm. If $k = \Theta(\epsilon^{-2}\log^2 n)$, then note that e is in some $F(G_i)$ with probability at least $\min\{\Omega(\lambda_e^{-1}\epsilon^{-2}\log^2 n), 1\}$ as required. See Ahn et al. [3] for further details.

Theorem 3.2.4. *There exists a sliding-window algorithm for maintaining a $(1+\epsilon)$ sparsifier using $O(\epsilon^{-2} n \operatorname{polylog} n)$ space.*

3.3 Matchings

We next consider the problem of finding large matchings in the sliding-window model. We first consider the unweighted case, maximum cardinality matching, and then generalize to the weighted case.

3.3.1 Maximum Cardinality Matching

Our approach for estimating the size of the maximum cardinality matching combines ideas from the powerful "smooth histograms" technique of Braverman and Ostrovsky [22, 21] with the fact that graph matchings are submodular and satisfy a "smooth-like" condition.

3.3.1.1 Smooth Histograms

The smooth histogram technique gives a general framework for maintaining an estimate of a function f on a sliding window provided that f satisfies a certain set of conditions. Among these conditions are:

1. *Smoothness:* For any $\alpha \in (0, 1)$ there exists $\beta \in (0, \alpha]$ such that

$$f(B) \geqslant (1 - \beta)f(AB) \quad \text{implies} \quad f(BC) \geqslant (1 - \alpha)f(ABC) \tag{3.1}$$

   where A, B, and C are disjoint segments of the stream and AB, BC, ABC denote concatenations of these segments.

2. *Approximability:* There exists a sublinear space stream algorithm that returns an estimate $\tilde{f}(A)$ for f evaluated on a (non-sliding-window) stream A, such that

$$(1 - \beta/4)f(A) \leqslant \tilde{f}(A) \leqslant (1 + \beta/4)f(A)$$

The basic idea behind smooth histograms is to approximate f on various suffixes $B_1, B_2, \ldots, B_k$ of the stream where $B_1 \supseteq W \supsetneq B_2 \supsetneq \cdots \supsetneq B_k$ and W is the active window. We refer to the B_i as "buckets." Roughly speaking, if we can ensure that $f(B_{i+1}) \approx (1-\epsilon)f(B_i)$ for each i then $f(B_2)$ is a good approximation for $f(W)$ and we will only need to consider a logarithmic number of suffixes. We will later present the relevant parts of the technique in more detail in the context of approximate matching.

3.3.1.2 Matchings are Almost Smooth

Let $m(A)$ denote the size of the maximum matching on a set of edges A. Unfortunately, the function m does not satisfy the above smoothness condition and cannot be approximated to sufficient accuracy. It does however satisfy a "smooth-like" condition:

Lemma 3.3.1. *For disjoint segments of the stream A, B, and C and for any $\beta > 0$:*

$$m(B) \geqslant (1-\beta)m(AB) \quad \text{implies} \quad m(BC) \geqslant \frac{1}{2}(1-\beta)m(ABC) \tag{3.2}$$

Proof. $2m(BC) \geqslant m(B) + m(BC) \geqslant (1-\beta)m(AB) + m(BC) \geqslant (1-\beta)m(ABC)$. The last step follows since $m(AB) + m(BC) \geqslant m(A) + m(BC) \geqslant m(ABC)$. □

The best known semi-streaming algorithm for approximating m on a stream A is a 2-approximation and a lower bound 1.582 has recently been proved [74]. Specifically, let $\hat{m}(A)$ be the size of the greedy matching on A. Then it is easy to show that

$$m(A) \geqslant \hat{m}(A) \geqslant m(A)/2 \tag{3.3}$$

Unfortunately, it is not possible to maintain a greedy matching over a sliding window.[1] However, by adjusting the analysis of [22], properties (3.2) and (3.3) suffice to show that smooth histograms can obtain an $(8+\epsilon)$-approximation of the maximum matching in the sliding-window model. However, by proving a modified smoothness condition that takes advantage of relationships between m and $\hat{m}$, and specifically the fact that $\hat{m}$ is maximal rather than just a 2-approximation, we will show that a smooth histograms-based approach can obtain a $(3+\epsilon)$-approximation.

[1]Maintaining the matching that would be generated by a greedy algorithm on the active window requires $\Omega(\min(n^2, L))$ space since it would always contain the oldest edge in the window and advancing the window allows us to recover all the edges. Similarly, it is not possible to construct the matching that would be returned by a greedy algorithm on reading the active window in reverse. This can be seen to require $\Omega(n^2)$ space even in the unbounded-stream model via reduction from INDEX. Alice considers the possible edges on an n-clique, and includes an edge iff the corresponding bit of her input is a 1. Bob then adds edges forming a perfect matching on all nodes except the endpoints of an edge of interest. The backwards greedy matching on the resulting graph consists of all of Bob's edges, plus one additional edge iff Alice's corresponding bit was a 1.

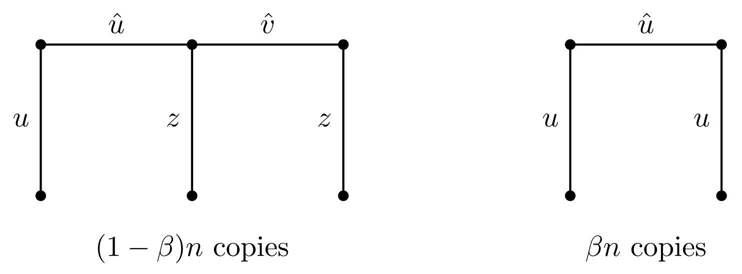


Figure 3.1: A graph with $O(n)$ nodes where Lemma 3.3.2 is tight. Edges arrive in the order: $\hat{u}$ edges (placed in greedy matching), other u edges, $\hat{v}$ edges, z edges. We then have $\hat{m}(AB) = n$, $\hat{m}(B) = (1-\beta)n = \hat{m}(BC)$, $m(ABC) = (3-\beta)n$.

Lemma 3.3.2. *Consider any disjoint segments A, B, C of a stream of edges and $\beta \in (0,1)$.*

$$\hat{m}(B) \geqslant (1-\beta)\hat{m}(AB) \quad \text{implies} \quad m(ABC) \leqslant \left(3 + \frac{2\beta}{1-\beta}\right)\hat{m}(BC) \ .$$

Note that it is the size of the *maximum* matching on ABC that is being compared with the size of the greedy matching on BC. To see that the above lemma is tight for any $\beta \in (0,1)$, consider the graph in Fig. 3.1, and let A be the stream of the $\hat{u}$ edges (which are placed in greedy matching) followed the u edges; B are the $\hat{v}$ edges, and C are the z edges. Then $\hat{m}(AB) = n$, $\hat{m}(B) = (1-\beta)n = \hat{m}(BC)$, and $m(ABC) = (3-\beta)n$.

Proof of Lemma 3.3.2. Let $M(X)$ and $\hat{M}(X)$ be the set of edges in an optimal matching on X and a maximal matching on X. We say that an edge in a matching *covers* the two nodes which are its endpoints.

We first note that every edge in $M(ABC)$ covers at least one node which is covered by $\hat{M}(AB) \cup \hat{M}(BC)$; otherwise, the edge could have been added to $\hat{M}(AB)$ or $\hat{M}(BC)$ or both. Since $M(ABC)$ is a matching, no two of its edges can cover the same node. Thus $m(ABC)$ is at most the number of nodes covered by $\hat{M}(AB) \cup \hat{M}(BC)$.

The number of nodes covered by $\hat{M}(AB) \cup \hat{M}(BC)$ is clearly at most $2\hat{m}(AB) + 2\hat{m}(BC)$. But this over-counts edges in $\hat{M}(B)$. Every edge in $\hat{M}(B)$ is clearly in $\hat{M}(BC)$; also, every edge in $\hat{M}(B)$ shares at least one node with an edge in $\hat{M}(AB)$ since the construction was greedy. Thus we find

$$\begin{aligned} m(ABC) &\leqslant 2\hat{m}(BC) + 2\hat{m}(AB) - \hat{m}(B) \\ &\leqslant 2\hat{m}(BC) + \frac{2}{1-\beta}\hat{m}(B) - \hat{m}(B) \\ &= 2\hat{m}(BC) + \frac{1+\beta}{1-\beta}\hat{m}(B) \\ &\leqslant \left(3 + \frac{2\beta}{1-\beta}\right)\hat{m}(BC) \ . \end{aligned}$$

where the second inequality follows from the assumption $\hat{m}(B) \geqslant (1-\beta)\hat{m}(AB)$. □

Theorem 3.3.3. *There exists a sliding-window algorithm for maintaining a $(3+\epsilon)$ approximation of the maximum cardinality matching using $O(\epsilon^{-1} n \log^2 n)$ space.*

Proof. We now use the smooth histograms technique to estimate the maximum matching size. The algorithm maintains maximal matchings over various buckets $B_1, \ldots, B_k$ where each bucket comprises of the edges in some suffix of the stream. Let W be the set of updates within the window. The buckets will always satisfy $B_1 \supseteq W \supsetneq B_2 \supsetneq \cdots \supsetneq B_k$, and thus $m(B_1) \geqslant m(W) \geqslant m(B_2)$.

Within each bucket B, we will keep a greedy matching whose size we denote by $\hat{m}(B)$. To achieve small space usage, whenever two nonadjacent buckets have greedy matchings of similar size, we will delete any buckets between them. Lemma 3.3.2 tells us that if the greedy matchings of two buckets have ever been close, then the smaller bucket's greedy matching is a good approximation of the size of the maximum matching on the larger bucket.

When a new edge e arrives, we update the buckets $B_1, \ldots, B_k$ and greedy matchings $\hat{m}(B_1), \ldots, \hat{m}(B_k)$ as follows where $\beta = \epsilon/4$:

1. Create a new empty bucket B_{k+1}.

2. Add e to the greedy matching within each bucket if possible.

3. For $i = 1 \dots k-2$:

   (a) Find the largest $j > i$ such that $\hat{m}(B_j) \geqslant (1-\beta)\hat{m}(B_i)$

   (b) Delete B_t for any $i < t < j$ and renumber the buckets.

4. If B_2 contains the entire active window, delete B_1 and renumber the buckets.

3.3.1.3 Space Usage

Step 3 deletes "unnecessary" buckets and therefore ensures that for all $i \leqslant k-2$ then $\hat{m}(B_{i+2}) < (1-\beta)\hat{m}(B_i)$. Since the maximum matching has size at most n, this ensures that the number of buckets is $O(\epsilon^{-1}\log n)$. Hence, the total number of bits used to maintain all k greedy matchings is $O(\epsilon^{-1} n \log^2 n)$.

3.3.1.4 Approximation Factor

We prove the invariant that for any $i < k$, either $\hat{m}(B_{i+1}) \geqslant m(B_i)/(3+\epsilon)$ or $|B_i| = |B_{i+1}| + 1$ (i.e., B_{i+1} includes all but the first edge of B_i) or both. If $|B_i| \neq |B_{i+1}| + 1$, then we must have deleted some bucket B which $B_i \subsetneq B \subsetneq B_{i+1}$. For this to have happened it must have been the case that $\hat{m}(B_{i+1}) \geqslant (1-\beta)\hat{m}(B_i)$ at the time. But then Lemma 3.3.2 implies that we currently satisfy:

$$m(B_i) \leqslant \left(3 + \frac{2\beta}{1-\beta}\right)\hat{m}(B_{i+1}) \leqslant (3+\epsilon)\hat{m}(B_{i+1}) \ .$$

Therefore, either $W = B_1$ and $\hat{m}(B_1)$ is a 2-approximation for $m(W)$, or we have

$$m(B_1) \geqslant m(W) \geqslant m(B_2) \geqslant \hat{m}(B_2) \geqslant \frac{m(B_1)}{3+\epsilon}$$

and thus $\hat{m}(B_2)$ is a $(3+\epsilon)$-approximation of $m(W)$. □

3.3.2 Weighted Matching

In the paper where we first presented these algorithms [33], results of Epstein et al. [40] were adapted to provide a 9.027 approximation for maximum matching on *weighted* graphs in the sliding window model. These results have been superceded by our later work, presented in Chapter 4, which provides an $6 + \epsilon$ approximation in space $O(\frac{1}{\epsilon} n \log^2 n)$ bits.

3.4 Minimum Spanning Tree

We next consider the problem of maintaining a minimum spanning forest in the sliding-window model. We show that it is possible to maintain a spanning forest that is at most a factor $(1 + \epsilon)$ from optimal but that maintaining the exact minimum spanning tree requires $\Omega(\max(n^2, L))$ space where L is the length of the sliding window.

The approximation algorithm is based on an idea of Chazelle et al. [27] where the problem is reduced to finding maximal acyclic subgraphs, i.e., spanning forests, among edges with similar weights. If each edge weight is rounded to the nearest power of $(1 + \epsilon)$, it can be shown that the minimum spanning tree of the union of these subgraphs is a $(1 + \epsilon)$ approximation of the minimum spanning tree of the original graph. The acyclic subgraphs can be found in the sliding-window model using the connectivity algorithm we presented earlier. The proof of the next theorem is almost identical to those in [2, Lemma 3.4].

Theorem 3.4.1. *There exists a sliding-window algorithm for maintaining a* $(1 + \epsilon)$ *approximation for the minimum spanning tree using* $O(\epsilon^{-1} n \log^2 n)$ *bits of space.*

In the "unbounded"/non-sliding-window stream model, it was possible to compute the exact minimum spanning tree via a simple algorithm: 1) add the latest edge to an acyclic subgraph that is being maintained, 2) if this results in a cycle, remove the heaviest weight edge in the cycle. However, the next theorem shows that maintaining an exact minimum spanning tree in the sliding-window model is not possible in sublinear space.

Theorem 3.4.2. *Maintaining an exact minimum spanning forest in the sliding-window model requires* $\Omega(\min(L, n^2))$ *space.*

Proof. Let $p = \min(L, n^2/4)$. The proof is by a reduction from the communication complexity of the two-party communication problem $\textsc{index}(p)$ where Alice holds a binary string $\vec{a} = a_1 a_2 \ldots a_p$ and Bob has an index $k \in \{1, \ldots, p\}$. If Alice sends a single message to Bob that enables Bob to output a_k with probability at least 2/3, then Alice's message must contain at $\Omega(p)$ bits [77].

Alice encodes her bits on the edges of a complete bipartite graph, writing in order the edges (u_1, v_1), (u_1, v_2), (u_1, v_3), $\ldots$, $(u_1, v_{\sqrt{p}})$, (u_2, v_1), $\ldots$, $(u_2, v_{\sqrt{p}})$, $\ldots$, $(u_{\sqrt{p}}, v_{\sqrt{p}})$ where the ith edge weight $2i + a_i$. Note that all these edges are in the current active window. Suppose she runs a sliding-window algorithm for exact MST on this graph and sends the memory state to Bob. Bob continues running the algorithm on an arbitrary set of $L - p + k - 1$ edges each of weight $2p + 2$. At this point any minimum spanning forest in the active window must contain the edge of weight $2k + a_k$ since it is the lowest-cost edge in the graph. Bob can thus learn a_k and hence the algorithm must have used $\Omega(p)$ bits of memory. Note that if Bob can only determine *what* the MST edges are, but not their weights, he can add an alternative path of weight $2k + 1/2$ to the node in question. □

3.5 Graph Spanners

In the unbounded stream model, the following greedy algorithm constructs a $2t-1$ stretch spanner with $O(n^{1+1/t})$ edges [5, 45]. We process the stream of edges in order; when seeing each edge (u, v), we add it to the spanner if there is not already a path from u to v of length $2t - 1$ or less. Any path in the original graph then increases by a factor of at most $2t-1$, so the resulting graph is a $(2t-1)$-spanner. The resulting graph has girth at least $2t+1$, so it has at most $O(n^{1+1/t})$ edges [20].

For graphs G_1, G_2 on the same set of nodes, let $G_1 \cup G_2$ denote the graph with the union of edges from G_1 and G_2. We will need the following simple lemma.

Lemma 3.5.1. *Let G_1 and G_2 be graphs on the same set of nodes, and let H_1 and H_2 be α-spanners of G_1 and G_2 respectively. Then $H_1 \cup H_2$ is an α-spanner of $G_1 \cup G_2$.*

Proof. Let $G = G_1 \cup G_2$ and $H = H_1 \cup H_2$. For arbitrary nodes u, v, consider a path of length $d_G(u,v)$. Each edge in this path is in G_1 or G_2 (or both). There is thus a path between the edge's endpoints in the corresponding H_1 or H_2 which is of length at most α. Thus, there is a path of length at most $\alpha d_G(u,v)$ in $H = H_1 \cup H_2$. □

Theorem 3.5.2. *There exists a sliding-window algorithm for maintaining a $(2t-1)$ spanner using $O(\sqrt{Ln^{1+1/t}})$ space.*

Proof. We batch the stream into blocks E_1, E_2, E_3, $\ldots$, where each consists of s edges. We buffer the edges in each block until it has been read completely, marking each edge with its arrival time. We then run the greedy spanner construction on each block in reverse order, obtaining a spanner S_i. By Lemma 3.5.1, the union of the spanners S_i and the edges in the current block, restricted to the active edges, is a spanner of the edges in the active window. This algorithm requires space s to track the edges in the current block. Each spanner S_i has $O(n^{1+1/t})$ edges, and at most L/s past blocks are within the window. The total number of edges stored by the algorithm is thus $s + (L/s)O(n^{1+1/t})$. Setting $s = \sqrt{Ln^{1+1/t}}$ gives $O(\sqrt{Ln^{1+1/t}})$ edges. □

We can achieve a space/approximation trade-off by appling the construction of Theorem 3.5.2 recursively (using that an α-spanner of an α-spanner of G is an α^2-spanner of G):

Theorem 3.5.3. *For any integer $d \geqslant 1$ there exists a sliding-window algorithm for maintaining a $(2t-1)^d$ spanner using $O\Big(L^{\frac{1}{d+1}} n^{\left(1+\frac{1}{t}\right)\left(1-\frac{1}{d+1}\right)}\Big)$ space.*

Proof. We will determine parameters $s_1, s_2, \ldots, s_d$. The algorithm keeps s_1 edges per block in the first level, when a block of s_1 edges are complete, it computes and stores a $(2t-1)$-spanner (in reverse order, as in Theorem 3.5.2 above). When the algorithm acquires s_2 of these first-level blocks, it computes and stores a $(2t-1)$-spanner of them in reverse order, and so on.

The algorithm will store s_1 single edges; $s_2 + s_3 + \cdots + s_d$ spanners at the intermediate levels; and $L/(s_1 s_2 s_3 \cdots s_d)$ edges at the top level. Each spanner contains $O(n^{1+1/t})$ edges.

The total space used is thus

$$\frac{Ln^{1+1/t}}{s_1 s_2 \cdots s_d} + (s_2 + s_3 + \cdots + s_d) n^{1+1/t} + s_1 \qquad (3.4)$$

For $s_1 = L^{\frac{1}{d+1}} n^{1-\frac{1}{d+1}}$, $s_2 = s_3 = \cdots = s_d = s_1/n$, this yields space usage $O\Big(L^{\frac{1}{d+1}} n^{\left(1+\frac{1}{t}\right)\left(1-\frac{1}{d+1}\right)}\Big)$. □

3.6 Conclusions

We initiate the study of graph problems in the well-studied sliding-window model. We present algorithms for a wide range of problems including testing connectivity and constructing combinatorial sparsifiers; constructing minimum spanning trees; approximating weighted and unweighted matchings; and estimating graph distances via the construction of spanners. Open problems include reducing the space required to construct graph spanners and improving the approximation ratio when estimating matching size.

CHAPTER 4
MATCHING

4.1 Introduction

The problem of finding maximum-weight matchings in the semi-streaming model has seen a great deal of attention since it was first introduced in [45], which gave a 6-approximation. This was improved to a 5.828-approximation in [81]; a 5.585-approximation in the [104]; and finally the current best, a $4.911 + \epsilon$-approximation in [40]. Other work has looked at generalizations of matching to submodular-function matching [25], and at maximum weighted matching in the map reduce model [78] and the sliding-window stream model [33].

We study a generalization of the problem of finding maximum weighted matching on graphs: finding maximum-weight independent sets in p-systems. p-systems are a type of independence system which generalize both matching on p-bounded hypergraphs and intersections of p matroids. We show that maximum-weight independent sets can be approximated by finding approximate solutions to maximum-cardinality independent sets. This reduction works in the streaming model and in several related models.

The structure of our reduction is related to [40], a streaming algorithm for maximum weighted matching. That algorithm partitions incoming edges into multiplicatively spaced weight classes and calculates a greedy matching on the edges within each weight class. At the end of the stream, they greedily merge the greedy matchings from largest to smallest.

Our algorithm uses weight classes that have lower bounds but not upper bounds, so classes that admit smaller edges are subsets of all the "more exclusive" classes. This gives rise to a better approximation with a more broadly applicable proof, allowing us to use the algorithm for arbitrary p-systems.

In §4.2 we define our model more specifically and state our main result. In §4.3 we present the general algorithm. In §4.4 we show that the general algorithm can be applied to improve the best known algorithms for weighted matching and weighted independent set problems in the streaming, sliding window, and map-reduce models. In §4.6 we comment on improvements for specific cases and outline future work.

Work in this chapter was originally done with Daniel Stubbs and is currently in submission.

4.2 Definitions and Results

4.2.1 Independence Systems

An *independence system* is a pair (S, I) comprising a finite set S and a set I of subsets of S (the "independent sets") such that

1. $\varnothing \in I$
2. For $X \subseteq X'$, $X' \in S \Rightarrow X \in S$.

An independence system is called a p-system if, for any $A \subseteq S$, the ratio between the largest and smallest maximal independent subsets of A is at most p. Graph matching forms a 2-system where S is the set of edges and where a set of edges is independent if no two edges share an endpoint. More generally, p-hypergraph matching is a p-system, as is the intersection of p matroids. For more detail on p-systems, see e.g. [70].

Given a p-system (S, I), the *maximum-cardinality independent set* problem is the problem of finding an independent set with the largest number of elements. Given a weight function $w : S \to \mathbb{R}_{\geqslant 0}$, the *maximum-weight independent set* problem is

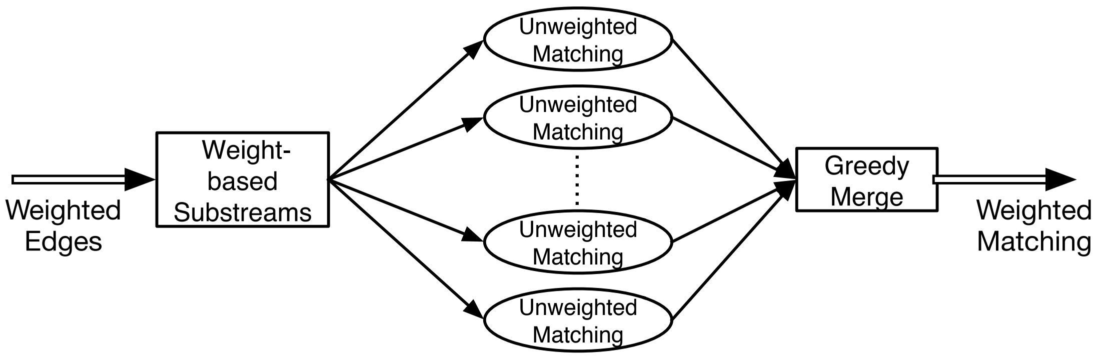


Figure 4.1: Block diagram of the weighted matching algorithm.

the problem of finding an independent set $X \in I$ which maximizes $\sum_{x \in X} w(x)$. These problems naturally extend the problems of finding maximum matchings on unweighted or weighted graphs; on first reading, the reader is encouraged to read "independent set" as "matching" and p as 2.

4.2.2 Streaming Reductions

Our algorithm will operate by transforming a maximum-weight independent set problem into a polylogarithmic number of maximum-cardinality independent set problems (Figure 4.1). This is an example of a class of reductions which we believe may be of particular interest in big data models: Turing reductions which make a polylogarithmically-bounded number of queries and which are "non-adaptive" (in the sense that the input to one query cannot depend on the output of another query). In this model, a reduction from problem A to problem B consists of processing the input to problem A into the inputs to polylog instances of B; solving the B instances; then processing the B outputs into the output to problem A. These models form a restricted class of the approximation-preserving streaming reductions used in [16].[1]

[1]Other papers introducing reductions in the streaming model defined many-one reductions between decision problems via a generalization of string homomorphisms [80, 13]; it is not readily apparent how to apply this work to approximation problems.

These reductions are natural choices because the resource classes being studied are typically closed under polylogarithmic blowup. Requiring the subproblems to be nonadaptive allows them to be easily parallelized. The resources used by the preprocessing and postprocessing steps can be restricted to preserve the classes of interest; in our case, the preprocessing step is merely testing the weight of each edge, and the postprocessing step is a greedy merge. Many existing streaming algorithms operate by reducing to polylog nonadaptive subproblems, including the precision sampling framework [7] and Indyk/Woodruff ℓ_p norm estimators [68].

We say that a reduction from A to B is p-approximate if, for any $\alpha \geqslant 1$, given an α-approximate solution to each B subproblem we can generate an αp-approximate solution to the A problem.

4.2.3 Main Result

Section 4.3 presents the proof of our main result:

Theorem 4.2.1. *Let (S, I) be a p-system. Then there is a $p(1+\epsilon)$-approximate non-adaptive Turing reduction from the problem of maximum-weight independent set on (S, I) to the problem of maximum-cardinality independent set on (S, I). The reduction uses $O(\frac{1}{\epsilon}\log n)$ copies of maximum-cardinality independent set.*

The reduction in Theorem 4.2.1 uses extremely minimal preprocessing (separating edges by weight) and minimal postprocessing (performing a greedy merge of the independent sets).

From the definition of p-systems, a greedily maximized set is always a p-approximate maximum cardinality matching. From Theorem 4.2.1 we thus immediately find:

Corollary 4.2.2. *We can perform a $p^2(1+\epsilon)$ approximation to maximum-weight independent set on any p-system, using $O(\frac{1}{\epsilon}\log n)$ times the resources necessary to greedily compute a maximal independent set on that p-system.*

The consequences of Corollary 4.2.2 in specific models are described in §4.4.

4.3 Algorithm

In this section we present a proof of Theorem 4.2.1.

Consider a p-system (S, I). Let $n = |S|$. Let the input E be a stream of elements from S, where each $e \in E$ is annotated with its weight $w(e)$.

For $i \in \mathbb{Z}$, we define substreams E_i, each containing all edges with weight above threshold $(1+\epsilon)^i$:

$$E_i \triangleq \{e \in E \mid w(e) \geqslant (1+\epsilon)^i\} \tag{4.1}$$

Note that i can be negative, but we assume that the range of possible weights $w(e)$ is polynomially bounded in n, so that we only need to consider substreams for $O(\frac{1}{\epsilon}\log n)$ values of i.[2]

To perform the reduction, assume that for $\alpha > 1$, we have for each E_i some independent set $C_i \subseteq E_i$ which contains at least $\frac{1}{\alpha}$ times as many elements as the maximum-cardinality independent set on E_i. We then greedily construct an independent set T by considering the elements in C_i in descending order of i. We output T. The top-level structure of the algorithm is summarized in Figure 4.1.

Consider a fixed maximum-weight independent set $\textsc{Opt}$ on E.

Lemma 4.3.1. *For each i, $|\textsc{Opt} \cap E_i| \leqslant \alpha p |T \cap E_i|$.*

Proof. For each class E_i let $\textsc{Opt}_i$ be the maximum-weight independent set on E_i. Our oracle returns C_i with $|\textsc{Opt}_i| \leqslant \alpha |C_i|$, and thus with $|\textsc{Opt} \cap E_i| \leqslant \alpha |C_i|$.

E_i is a p-system, with independent sets formed by any valid matching. Now consider maximal independent sets on $C_i \cup T$ (recall $C_i \cup T \subseteq E_i$). We have that C_i is a maximal independent set of size $|C_i|$. Thus, by the definition of p-systems, no maximal independent subset of $C_i \cup T$ can have size less than $\frac{1}{p}|C_i|$.

[2] If we do not have this guarantee, we can keep track of the highest-weight item seen so far, and discard any items with less than $2\epsilon/n$ times that weight. A matching made entirely of these discarded edges is then at most an ϵ fraction of the output weight (since the weight we output is at least the weight of the largest edge).

The greedy merge can thus always add elements from C_i to T until $|T \cap E_i| \geqslant \frac{1}{p}|C_i|$. Combining this with the above we have $|\textsc{Opt} \cap E_i| \leqslant \alpha p|T \cap E_i|$.

□

Lemma 4.3.2. *There exists a function f from* Opt *to T such that for each $e \in \textsc{Opt}$, $w(e) \leqslant (1+\epsilon)w(f(e))$ and for each $t \in T$, there are at most αp elements $e \in \textsc{Opt}$ such that $f(e) = t$.*

Proof. We define f inductively, considering $\textsc{Opt} \cap E_i$ in descending order by i and picking $f(e)$ from among the elements of $T \cap E_i$ that have fewer than αp edges already associated with them. This restriction will guarantee that $f(e)$ is from at least as high a class as e, which gives us that $w(e) \leqslant (1+\epsilon)w(f(e))$. By Lemma 4.3.1, there are always enough elements in $T \cap E_i$ to avoid overcrowding. □

Lemma 4.3.2 leads immediately to a charging argument which proves Theorem 4.2.1: every edge $e \in \textsc{Opt}$ is an element of the preimage $f^{-1}(t)$ for some t, and for each t

$$\sum_{e \in f^{-1}(t)} w(e) \leqslant |f^{-1}(t)|(1+\epsilon)w(t) \leqslant \alpha p(1+\epsilon)w(t) \tag{4.2}$$

so we have

$$w(\textsc{Opt}) = \sum_{e \in \textsc{Opt}} w(e) = \sum_{t \in T} \sum_{e \in f^{-1}(t)} w(e) \leqslant \sum_{t \in T} \alpha p(1+\epsilon)w(t) = \alpha p(1+\epsilon)w(T) \tag{4.3}$$

4.4 Extensions

The general result of Corollary 4.2.2 improves the best known algorithms for many matching problems. These are summarized in Table 4.1.

Maximum weighted graph matching (MWM) is a 2-system; since any maximal matching is a 2-approximation of unweighted maximum matching, our algorithm can thus provide a $4+\epsilon$ approximation by keeping a greedy unweighted matching in each

Problem	Model	Previous	Ref	This Work
MWM	One-pass streaming	4.911	[40]	4
MWM	One-pass sliding window	9.027	[33]	6
MWM	Map-reduce	8	[78]	4
3-MWM	One-pass streaming	9.899	[25]	9
2-MWIS	One-pass streaming	8	[25]	4
3-MWIS	One-pass streaming	9.899	[25]	9

Table 4.1: Approximation factor improvements over previous results. ϵ factors have been omitted.

E_i class. This provides an approximation guarantee in any big data model where we are capable of performing greedy matching on weight-based substreams of the data. Several of these applications are explained below; each of these is an improvement of the previous best results in these models. Results are summarized in Figure 4.1.

The semi-streaming model (defined in [45]) allows one-way access to a stream of weighted edges on a machine limited to $O(n \operatorname{polylog} n)$ memory. A series of papers in this model have provided improved approximation guarantees in this model [45, 81, 104, 40]; the current best is a $4.911 + \epsilon$ approximation [40]. Keeping a maximal matching in the semistreaming model is trivial (see e.g. [45]) and the machine has enough memory space to store one maximal matching for each of the $O(\log n)$ many weight classes, thus we find

Corollary 4.4.1. *There is a $4 + \epsilon$ approximation algorithm for maximum weighted matching in the semistreaming model.*

Chakribarti et al. 2013 [25] extended semistreaming matching algorithms to the more general cases of weighted matching on hypergraphs of degree p (p-MWM) and of finding maximum-weight independent sets in p-intersection systems (p-MWIS). Our algorithm improves their approximation ratios for the most practical $p = 2$ and $p = 3$ cases (see Figure 4.1).

Corollary 4.4.2. *There is a semistreaming algorithm for finding the Maximum-Weight Independent Set on a p-system with approximation ratio* $p^2 + \epsilon$.

Corollary 4.4.3. *There is a semistreaming algorithm for finding the Maximum-Weight Matching on a degree p hypergraph with approximation ratio* $p^2 + \epsilon$.

In the related semi-streaming "sliding window" graph model [35, 33], there is a fixed window length $L \in \omega(n \operatorname{polylog} n)$, and we are interested in maintaining (at all times) a maximum matching over the most recent L edges. We are again limited to $O(n \operatorname{polylog} n)$ memory space. In this model, only a $3+\epsilon$-approximation to unweighted matching is known [33], and we thus find:

Corollary 4.4.4. *There is a* $6 + \epsilon$ *approximation algorithm for maximum weighted matching in the semi-streaming sliding window model.*

The class $\mathcal{MRC}^0$ [75] is a theoretical model for MapReduce computations achievable with a constant number of rounds. In this model, even though the edge set does not fit on any individual processor, it is possible to find a maximal matching [78] (and thus a 2-approximation of the maximum unweighted matching). This immediately yields an improvement over the previous best-known 8-approximation algorithm for maximum weighted matching [78], with no additional communication cost (since the merge can be performed on a single processor).

Corollary 4.4.5. *There is a constant-round* $4 + \epsilon$ *approximation algorithm for maximum weighted matching in the MapReduce model* $\mathcal{MRC}^0$.

4.5 Lower Bounds for Graph Matching

In this section we consider the case of maximum weighted graph matching, and we present graph constructions which prove lower bounds on the approximation ratios achievable by our algorithm. These constructions extend to a general family of

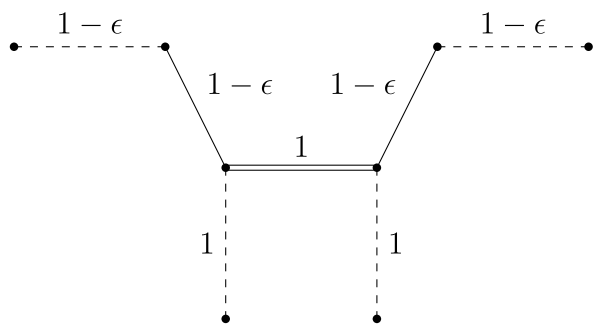


Figure 4.2: Graph with output weight 1 and optimum matching weight $4-2\epsilon$. In the weight class $[1,\infty)$ the double-lined edge is remembered; in the weight class $[1-\epsilon,\infty)$ the two single-lined edges are remembered. The double-lined edge is output. The dashed edges are the optimum matching (and are not remembered in either class).

techniques which also includes previous weight-class-based approaches to maximum weighted matching.

The algorithm presented in §4.3 computes its output matching by performing a greedy matching on remembered edges, in decreasing order of weight. Our analysis showed that this was a 4-approximation. In Fig. 4.2 we present a graph where the algorithm's approximation ratio is $4-2\epsilon$, showing that our analysis is tight to within $1+\epsilon$ factors.

In the graph of Fig. 4.2, the greedy matching on the remembered edges has weight 1, but the maximum weight matching on remembered edges has weight $2-2\epsilon$. In practice, many applications may be able to spend the post-processing time necessary to find the maximum-weight matching on the remembered edges (which are, after all, a sparse subgraph of the original graph). An obvious question is whether this post-processing can provide a stronger approximation guarantee.

The graph of Fig. 4.3 shows that our algorithm cannot achieve better than a 3.5 approximation, even when we output the maximum-weight matching on all of the remembered edges. Only remembered edges are drawn. Edges arrive in increasing order by weight, with the remembered edges appearing before other edges of the same weight. When the graph of Fig. 4.3 is extended upwards, any algorithm

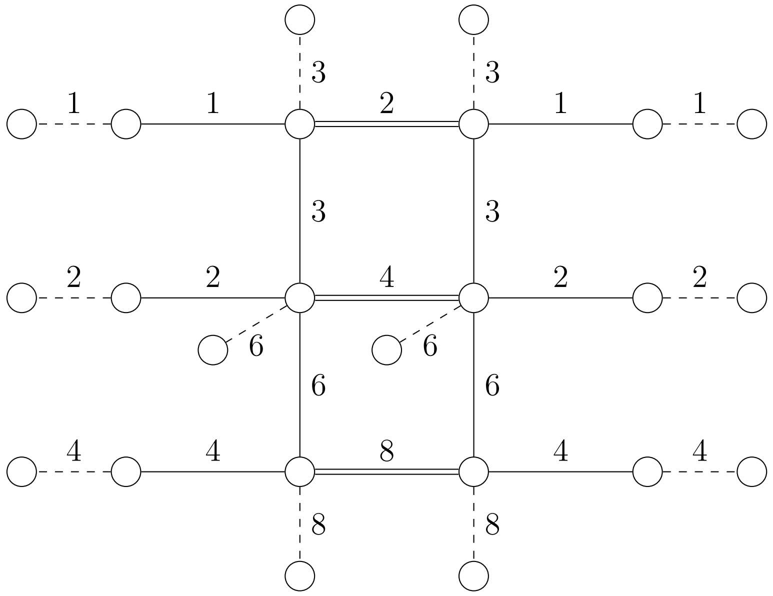


Figure 4.3: A graph which, when extended upwards, approaches approximation ratio 3.5. The dotted-line edges form the optimal matching, but are not remembered in any weight class. Solid edges are remembered but not output; double-lined edges are remembered and output. These remembered edges are produced by a stream where the edges arrive in order of increasing weight, with to-be-remembered edges arriving first. The reader can verify that within each weight class, the set of remembered edges is maximal.
On this graph, the output has weight 14, and the optimum matching has weight 48, for an approximation ratio of ≈3.429. The graph can be extended upwards, with each new layer including a single new output edge which decreases in weight by a power of 2; the approximation ratio quickly approaches 3.5.

which uses greedy matchings on weight-based substreams cannot do better than a 3.5-approximation, because it is incapable of remembering any edge from Opt. This class of algorithms includes our algorithm and also the previous best algorithm of [40].

4.6 Conclusion

For specific systems of interest, we may be able to obtain stronger approximation guarantees, particularly by being more clever in our post-processing of memory. The case of one-pass streaming algorithms for graph matching is of particular interest. An obvious improvement to our algorithm is to calculate the maximum matching on all edges held in memory (via, e.g., the Blossom algorithm [85]) rather than performing a greedy matching on edges held. In future work we plan to analyze the approximation guarantees achievable by this algorithm. We conjecture that this improvement yields a $(3.5 + \epsilon)$-approximation, tight to the lower bound shown in Fig. 4.3.

CHAPTER 5
SAMPLE VS. SPACE COMPLEXITY

5.1 Introduction

In this chapter, we consider receiving a stream of IID samples from some unknown distribution D, and being tasked with estimating some function $f(D)$ of the distribution. Two natural questions arise, both of which have been studied extensively:

1. The statistics question is how to bound the *sample complexity*: how many samples are required to estimate $f(D)$ to some prescribed accuracy with high probability?

2. The data stream question is how to bound the *space complexity*: how much memory is required to compute or approximate the estimator for $f(D)$?

The focus of this chapter is investigating the trade-offs between these two quantities: If the stream is observed for time t, how much memory, s, is required to estimate $f(D)$. Obviously t needs to be at least the sample complexity t^* for the problem since, with less than t^* samples, insufficient information has been revealed about the data set and therefore even unlimited computation space and time cannot yield a good answer. However, in this chapter we observe that for two canonical problems, the space complexity to process these samples decreases as t/t^* increases.

Work in this chapter was originally done with Andrew McGregor and David Woodruff and was presented at the MASSIVE 2013 workshop.

5.1.1 Sufficient Statistics and Data Streams

The goal of space-efficiency in statistical estimation is not new. In the study of *sufficient statistics* [47] the goal is to prove that it suffices to maintain a small number of statistics about the input when estimating certain parameters of the source distribution D. For example, to estimate μ if $D \sim N(\mu, 1)$, it is sufficient to maintain the sum and count of the samples; other information can be discarded. However, for non-parametric problems sufficient statistics typically do not exist. Therefore, it makes sense to also consider "approximate sufficient statistics", i.e., statistics about the stream of samples that can be computed online that will suffice to estimate the relevant properties of the input with high probability.

In contrast to the majority of data stream research, in the above setting we do not need to consider adversarially-ordered streams since the assumption is that input stream is generated by a stochastic process. There is a growing body of work on randomly-ordered streams [56, 96, 90, 24, 84, 53, 23]. Some work has also explicitly considered streams of IID samples [28, 103, 55, 92]. There has also been work on hypothesis testing given limited space [76, 60].

5.1.2 Subsampling vs. Supersampling

Other related work includes a paper by an overlapping set of authors [83] (see also [98]) that considered the problem of processing data streams whose arrival rate was so high that it was not possible to observe every element of the stream. Consequently, it was presumed that the stream was first *subsampled* and then properties of the original stream had to be deduced from the samples. In contrast, in this work we essentially consider oversampling, or *supersampling* the data set. The motivation is two-fold. First, in many applications there is an abundance of redundant data, e.g., from sensors that a continually monitoring a static environment, and it makes sense to find a way to capitalize on this data. Second, it may be preferable from a computational point of view, to run a fast light-weight algorithm on a lot of data rather than a computationally expensive algorithm on a small amount of data.

5.1.3 Results

We study the trade-off between sample and space complexity for the canonical problems of undirected connectivity [43, 44, 97] and estimating frequency moments [4, 68]. Our results are as follows:

1. *Frequency Moments.* Suppose $D = D_p$ is a distribution $(p_1, \ldots, p_n)$ over $[n]$. Then, to estimate $F_k(D) = \sum_i p_i^k$ for $k \geqslant 2$ up to a factor $(1+\epsilon)$ it is sufficient[1] for
$$s \cdot t = \tilde{O}_\epsilon(n^{2-2/k}) \tag{5.1}$$
if $t \geqslant t^* = \Omega(n^{1-1/k})$. We also show the lower bound that $s = \tilde{\Omega}(n^{2.5(1-1/k)}/t^{1.5})$ space is necessary to return a constant approximation. We present the algorithmic result in §5.2 and the lower bound in §5.4.

2. *Graph Connectivity.* Suppose $D = D_G$ is the uniform distribution of the set of edges of an undirected, unweighted graph G. We show that it suffices for
$$s^2 \cdot t = \tilde{O}(|E| \cdot |V|^2)$$
if $t \geqslant t^* = \Omega(|E| \log |E|)$. We present these results in §5.3.

5.2 Frequency Moments

In this section we consider a stream of samples $\langle a_1, a_2, a_3, \ldots \rangle$ where each a_i is drawn independently from some unknown, discrete distribution p over $[n]$. Let $p_j = \mathbb{P}[a_i = j]$ and define $F_k = \sum_j p_j^k$. Our main theorem in this section is as follows:

[1]$\tilde{O}_\epsilon(f(n))$ omits $\mathrm{poly}(\log n, 1/\epsilon)$ terms.

Theorem 5.2.1. *For $k \geqslant 2$ and for $t = \Omega(n^{1-1/k})$, estimating F_k up to a factor $(1 \pm \epsilon)$ given t samples is possible in $\tilde{O}_\epsilon(n^{2-2/k}/t)$ space.*

We will prove the upper bound in this section and establish a lower bound in §5.4. We start with a simple algorithm for the $k = 2$ case and then extend to $k > 2$ via a variant of ℓ_2 sampling. Note that the condition that $t = \Omega(n^{1-1/k})$ was shown to be necessary by Bar-Yossef [15]; with less samples it is information-theoretically impossible to estimate F_k.

5.2.1 Warm-Up: F_2 Estimation

Consider the stream of samples as defining a sequence of pairs

$$\langle (a_1, a_2), (a_3, a_4), \ldots, (a_{t-1}, a_t) \rangle \ .$$

Note that the probability that a pair of values is identical is exactly F_2. Define $X_i \in \{0, 1\}$ where $X_i = \mathbb{1}[a_{2i-1} = a_{2i}]$. Let $X = \frac{2}{t} \sum_i X_i$.

Lemma 5.2.2. *If $t > 6\epsilon^{-2} n \ln(2/\delta)$ then $\mathbb{P}[|X - F_2| \geqslant \epsilon F_2] \leqslant \delta$.*

Proof. First note that $\mathbf{E}[X] = \sum_i p_i^2 = F_2$. Since each $X_i \in \{0, 1\}$ is independent, by an application of the Chernoff bound, $\mathbb{P}[|X - F_2| \geqslant \epsilon F_2] \leqslant 2e^{-\epsilon^2 t F_2/6}$. Hence this probability is less than δ if $t \geqslant 6 \ln(2/\delta)/(F_2 \epsilon^2)$. Since $F_2 \geqslant 1/n$ the result follows. □

5.2.2 Technique: ℓ_2-Sampling in the IID model

The main technique that will be used in our F_k estimation algorithm is a form of ℓ_2 sampling. Here we are given a sequence of samples $\langle a_1, a_2, a_3 \ldots \rangle$ from p and wish to generate a sample from q where $q_i = p_i^2/F_2(p)$. In the F_2-estimation algorithm above, we observed that returning a_{2i} for the smallest i such that $a_{2i-1} = a_{2i}$ yields a value drawn according to q. This required $O(\log n)$ space but potentially $\Omega(n)$ samples even in expectation.

We next consider a generalization of this idea: rather than taking two samples and hoping they are the same value, we take w samples and return the first value that occurs twice in the sequence. If w is small, it is possible there will be no duplicates. However, the next lemma establishes that i is sampled with probability approximately proportional to p_i^2.

Lemma 5.2.3. *Probability of returning $i \in [n]$ satisfies:*

$$\frac{cp_i^2}{1 + wp_i} \leqslant \mathbb{P}[\textit{returning } i] \leqslant cp_i^2$$

where c does not dependent on i. The probability of returning any value in $[n]$ is at least $\min(1/2, w^2/(50n))$.

Proof. To simplify notation let $i = n$. Let

$$A_k = \sum_{\text{distinct } i_1,\ldots,i_k \in [n]} p_{i_1} \ldots p_{i_k} \quad \text{and} \quad B_k = \sum_{\text{distinct } i_1,\ldots,i_k \in [n-1]} p_{i_1} \ldots p_{i_k}$$

Then, the probability of returning n can be written as $p_n^2(1 + 2B_1 + 3B_2 + \ldots + (w - 1)B_{w-2})$. Since $A_k = B_k + kp_nB_{k-1}$, we then have

$$\begin{aligned}
& 1 + 2B_1 + 3B_2 + \ldots + (w-1)B_{w-2} \\
\leqslant\ & 1 + 2A_1 + 3A_2 + \ldots + (w-1)A_{w-2} \\
=\ & (1 + 2p_n) + 2(1 + 3p_n)B_1 + 3(1 + 4p_n)B_2 + \ldots \\
& \qquad + (w-2)(1 + (w-1)p_n)B_{w-3} + (w-1)B_{r-2}
\end{aligned}$$

The first part of the lemma follows by setting $c = 1 + 2A_1 + 3A_2 + \cdots + (w-1)A_{w-2}$.

For the second part, observe that the probability of returning a value in $[n]$ is minimized if p is the uniform distribution over $[n]$. With probability at least $1/2$ there is at least one pair of duplicate samples after $5\sqrt{n}$ samples. If such a pair exists, the probability that a specific pair occurs within the first w samples is $\min(1, (w/5\sqrt{n})^2)$ since every ordering of the $5\sqrt{n}$ samples is equally likely. $\square$

5.2.2.1 Frequency Moments Algorithm and Analysis

The algorithm we present is inspired by the ℓ_2-sampling approach for frequency moment estimation [87, 29]. Specifically, suppose we had a sequence of $r = O(\epsilon^{-2} n^{1-2/k} \log \delta^{-1})$ samples and frequency estimations $(i_1, \tilde{p}_{i_1}), (i_2, \tilde{p}_{i_2}), \ldots (i_r, \tilde{p}_{i_r})$, where $\tilde{p}_{i_j} = (1 \pm \epsilon/2) p_{i_j}$ and where samples were chosen with probability $\mathbb{P}[i_j = \ell] = (1 \pm \epsilon/2) p_l^2 / F_2(p)$. Then a simple calculation and an application of the Chernoff bound would allow us to estimate $F_k(p)$ via

$$\Pr\left[\left|\frac{F_2(p)}{r} \sum_{j=1}^{r} \tilde{p}_{i_j}^{k-2} - F_k(p)\right| > \epsilon F_k(p)\right] \leqslant \delta \ .$$

We can estimate $F_2(p)$ in the sampling model as above. The remaining challenge is to approximate $\sum_{j=1}^{r} \tilde{p}_{i_j}^{k-2}$. We present an algorithm which approximates this by sampling r elements (with appropriate probabilities) and then approximating their frequencies.

In what follows we assume all $p_i \leqslant \epsilon^3 t / n^{2-2/k}$ since we can identify all larger probabilities (there will be only $O(\epsilon^{-3} n^{2-2/k}/t)$ of them and they'll all appear among the first $O(\epsilon^{-3} \log n n^{2-2/k}/t)$ samples) and estimate their frequencies separately.

5.2.2.1.1 Algorithm. We consider the t samples divided into t/w contiguous segments

$$C_1, C_2, \ldots, C_{t/w} \ ,$$

each consisting of $w = \max(5\sqrt{n}, \alpha \epsilon^{-2} \log \delta^{-1} n^{2-2/k}/t)$ samples for some large constant α. In each segment C_i, the goal is to *collect* the first duplicate e_i . Given the assumption that $p_i \leqslant \epsilon/w$ for all i, Lemma 5.2.3 ensures that the element is chosen with probability $\propto(1 \pm \epsilon)p_{e_i}^2$ as required. Subsequently we *monitor* e_i, perhaps into future segments, until we have a good estimate of p_{e_i}.

1. $X \leftarrow 0$
2. *Collect:* For each segment C_i, let e_i be the first element to have a duplicate (if one exists).
3. *Monitoring:* Let c_i be the number of samples in $C_{i+1} \cup C_{i+2} \cup \ldots$ that appear before e_i has occurred $d = O(\epsilon^{-2} \log n)$ times. Let $X \leftarrow X + (d/c_i)^{k-2}/r$
4. Repeat until r samples e_i are found. Return $F_2 \cdot X$.

The following lemma establishes that we get a good approximation of the probability of each element collected.

Lemma 5.2.4. *For all $i \in \{1, 2, \ldots, r\}$, $d/c_i = (1 \pm \epsilon/2)p_i$*

Lemma 5.2.4 follows immediately from the following tail bounds for the negative binomial distribution:

Lemma 5.2.5. *Let $X \sim NB(r, p)$ be distributed according to the negative binomial distribution with parameters $r \in \mathbb{N}$ and $p \in (0, 1)$. Then,*

$$\mathbb{P}[|r/X - p| \geqslant \epsilon p|] \leqslant 2 \exp\left(\frac{-\epsilon^2 r}{3(1+\epsilon)}\right)$$

Hence, for some $c > 0$ and $r = c\epsilon^{-2} \log m$, $\mathbb{P}[|r/X - p| \geqslant \epsilon p|] \leqslant m^{-100}$.

Proof. Consider $Y \sim Bin(\frac{r}{p(1+\epsilon)}, p)$. Then

$$\mathbb{P}\left[X < \frac{r}{p(1+\epsilon)}\right] = \mathbb{P}[Y \geqslant r] = \mathbb{P}\Big[Y \geqslant (1+\epsilon)\,\mathbf{E}[Y]\Big] \leqslant \exp\left(\frac{-\epsilon^2 r}{3(1+\epsilon)}\right)$$

Similarly, let $Y \sim Bin(\frac{r}{p(1-\epsilon)}, p)$. Then

$$\mathbb{P}\left[X > \frac{r}{p(1-\epsilon)}\right] = \mathbb{P}[Y < r] = \mathbb{P}\Big[Y < (1-\epsilon)\,\mathbf{E}[Y]\Big] \leqslant \exp\left(\frac{-\epsilon^2 r}{3(1-\epsilon)}\right)$$

The result follows from the union bound. □

Lemma 5.2.6. *With high probability, the algorithm terminates before the end of the stream is reached.*

Proof. By Lemma 5.2.3, the probability of finding a duplicate during a segment is at least $\min(1/2, w^2/(50n))$. Hence the expected number of elements collected is

$$\min(1/2, w^2/(50n)) \cdot t/w = tw/(50n) \geqslant \alpha\epsilon^{-2}\log(\delta^{-1})n^{1-2/k}/50\ .$$

For sufficiently large $\alpha > 0$, this is at least r with high probability. □

The last lemma bounds the space use of the algorithm.

Lemma 5.2.7. *The algorithm uses $\tilde{O}_\epsilon(n^{2-2/k}/t)$ space.*

Proof. To find a duplicate in each segment only requires $w \leqslant n^{2-2/k}/t$ space. It remains to show that we are never monitoring too many items at the same time. During the j-th segment, we may be estimating p_i. By Lemma 5.2.5 we may assume that we have not been monitoring i for more than $O(\epsilon^{-2}\log n/p_i)$ time, or equivalently more than $O(\epsilon^{-2}\log n/(wp_i))$ segments. The probability that i is a duplicate during a segment is at most $\binom{w}{2}p_i^2$. Hence,

$$\begin{aligned}\mathbb{P}[\text{monitoring } i \text{ during } j\text{-th segment}] &= \binom{w}{2}p_i^2 \cdot O(\epsilon^{-2}\log n/(wp_i)) \\ &= O(n^{2-2/k}p_i\epsilon^{-2}\log n/t)\end{aligned}$$

Hence, the total expected number of items being monitored is $O(n^{2-2/k}\epsilon^{-2}\log n/t)$ as claimed. □

5.3 Connectivity

In this section, we consider the problem of determining whether a graph $G = (V, E)$ is connected given a sequence of t random samples (with replacement) from E. For notational convenience, denote the number of nodes by $n = |V|$ and the number of edges by $m = |E|$.

Our algorithm uses the sampled edges to simulate classical random walks on the graph. In §5.3.1, we discuss how to simulate (and tightly analyze) random walks in our model. The main technical difficulty is ensuring independence when simulating multiple random walks in parallel. Then, in §5.3.2, we adapt a connectivity algorithm of Feige [44] to achieve the required space/sample trade-off.

5.3.1 Technique: Emulating Classical Random Walks

Consider the following basic algorithm: given a node v, we sample edges until we receive an edge $\{v, u\}$ for some u. At this point, we move to node u, and repeat. We refer to this method as a *sampling walk*. Note that the expected time to leave v is $m/d(v)$ samples[2] where $d(v)$ is the degree of a node v, and so a single step of a classical random walk may require $\Omega(m)$ samples if v has low degree.

[2]This is because the number of samples is geometrically distributed with parameter $d(v)/m$.

5.3.1.1 An Inefficient Connectivity Algorithm

This basic algorithm already leads to a $O(\log n)$ space algorithm which uses $O(m^2n^2)$ samples in expectation. This follows by starting a sampling walk at node 1 and emulating a classical walk until it traverses nodes $2, 3, \dots, n$ in order. The expected length of this walk is $O(mn^2)$ because the cover time of G, i.e., the expected length of walk until it visits all nodes (see, e.g., [86]), is $O(mn)$ and there are $n - 1$ segments in the traversal. Hence, emulating the random walk takes $O(m^2n^2)$ samples in expectation. The space use is $O(\log n)$ bits because the algorithm just needs to remember the current node and the furthest node that has been reached in the sequence. In what follows, we will improve upon the number of samples required and generalize to algorithms that use more space.

5.3.1.2 The Loopy Graph and an Improved Analysis

The first improvement comes via a better analysis. At a node v with $d(v)$ neighbors, there are $d(v)$ possible samples which would result in a move, and $m - d(v)$ samples which would not. We can thus view our sampling-based walk on G as a classical random walk on a new graph H formed adding $m - d(v)$ self-loops to each vertex v in G. We call H the "loopy graph".

This view of the sampling walk illuminates its properties. Specifically, H's cover time is $O(mn^2)$ since there are mn edges and n nodes. Hence, the above "inefficient" algorithm actually only requires $(n - 1) \times \text{cover-time}(H) = O(mn^3)$ samples. We will also subsequently use the fact that since H is m-regular, its stable distribution is uniform across all nodes.

5.3.1.3 Multiple Independent Random Walks

Random walks experience dramatic speedups in cover time, hitting time, etc., when they are split into multiple shorter walks; [37] provides a recent survey and results. These speedups naturally require the walks to be independent. In this section, we consider performing $p \leqslant n$ random walks in parallel, with the starting point of each walk chosen independently and uniformly from the nodes (and thus according to the stationary distribution on H). Running these p walks will require $O(p \log n)$ space.

The main theorem of this section establishes that it is possible to efficiently perform p independent, parallel walks in H.

Theorem 5.3.1. *Given $p \leqslant n$ parallel random walks in H, each starting at an independently-chosen uniformly random node, we can simulate one independent step of each walk using $O(\log n / \log \log n)$ total samples.*

5.3.1.4 Issue 1: Multiple Walks can use a Sampled Edge

The first issue we encounter is that a single sample may be a valid move for multiple walks. If we allow multiple walks to use the same sample, we introduce obvious dependence; if we only allow one of our walks to use the sample, we are "slowing down" walks that have collisions, and again introducing dependence.

When multiple walks are at the same node, we will handle them independently in the following way. We partition the p walks into $B_1 \cup B_2 \cup \ldots \cup B_r$ where each B_i contains at most one walk at each node. We process each batch in turn and hence the total number of samples required equals the number of samples required for a batch multiplied by the number of batches. The next lemma establishes that it suffices to consider $r = O(\log n / \log \log n)$ batches.

Lemma 5.3.2. *With high probability, no node ever contains more than $O(\log n / \log \log n)$ walks.*

Proof. Consider a fixed node at a fixed time. Let Z be the number of walks in this node. Note that $Z \sim Bin(p, 1/n)$ since each walk is independent and is equally likely to be at any node. Hence $\mathbf{E}[Z] = p/n \leqslant 1$. By an application of the Chernoff bound,

$$\mathbb{P}[Z \geqslant c \log n / \log\log n] \leqslant n^{-10}$$

for some large constant c. The lemma follows by taking the union bound over the n nodes and $\operatorname{poly} n$ time-steps. $\square$

Henceforth, we assume that at most one walk is at each node, i.e., we analyze how many samples are required to process a single batch. The remaining case where a sampled edge may be valid for multiple walks is if there are walks at both endpoints. To solve this problem, we randomly orient each sampled edge so that it is valid for only one walk. This increases the expected number of samples required by a factor of 2.

5.3.1.5 Issue 2: Negative Correlation

We have reduced the problem to the following situation: we have p distinct nodes $u_1, \ldots, u_p$ and can sample arcs uv uniformly from the set $E^+ = \{uv : \{u, v\} \in E_G\}$, i.e., the set of arcs formed by bidirecting each each in E_G. Note that $|E^+| = 2|E_G|$. The goal is to generate a set of arcs $\{u_1v_1, \ldots, u_pv_p\}$ such that arc is chosen independently and for each i,

$$v_i = \begin{cases} v \in_R \Gamma(u_i) & \text{with probability } d_G(v_i)/|E^+| \\ u_i & \text{with probability } 1 - d_G(v_i)/|E^+| \end{cases} \tag{5.2}$$

where $\Gamma(u_i) = \{v : \{u, v\} \in E\}$ is the neighborhood of u_i in G.

Consider the following procedure: draw a single sample $uv \in_R E^+$ and, for each i, set $v_i = v$ if $u = u_i$, or $v_i = u_i$ otherwise. This procedure picks each v_i according to the desired distribution:

$$\mathbb{P}[v_i = u_i] = 1 - d_G(v_i)/|E^+|$$

and conditioned on $\{v_i \neq u_i\}$, v_i is uniformly chosen from $\Gamma(u_i)$. Unfortunately, the procedure obviously does not satisfy the independence requirement because the events $\{u_i = v_i\}$ and $\{u_j = v_j\}$ are negatively correlated. However, the following theorem establishes that, with only $O(1)$ samples from E^+ in expectation, it is possible to sample independently according to the desired distribution.

Theorem 5.3.3 (Efficient Parallel Sampling)**.** *There exists an algorithm that returns samples* $(v_1, \ldots, v_p)$ *drawn from the desired distribution* (5.2) *while using at most* $2e - 1$ *samples from* E^+ *in expectation.*

Proof. Our algorithm operates in rounds and each round uses at most 2 samples from E^+. At the beginning of a round, suppose we have already assigned values to $v_1, \ldots, v_i$ for some $i \geqslant 0$. Then the round proceeds as follows:

1. Sample $uv \in E^+$:
   (a) If $u \notin \{u_{i+1}, \ldots, u_p\}$ then set $v_{i+1} = u_{i+1}, \ldots, v_p = u_p$
   (b) If $u = u_j$ for some $j \in \{i+1, \ldots, p\}$ then sample an additional arc $wx \in E^+$
      i. If $w \in \{u_{i+1}, \ldots, u_{j-1}\}$ then set $v_{i+1} = u_{i+1}, \ldots, v_{j-1} = u_{j-1}, v_j = u_j$
      ii. If $w \notin \{u_{i+1}, \ldots, u_{j-1}\}$ then set $v_{i+1} = u_{i+1}, \ldots, v_{j-1} = u_{j-1}, v_j = v$

and we repeat the process until all $v_1, \ldots, v_p$ have been assigned.

To analyze the algorithm we define $T_j = \{u_j v : \{u_j, v\} \in E_G\}$ to be the set of $d_G(u_j)$ arcs leaving u_j and note that because $u_1, \ldots, u_p$ are distinct, $T_1, \ldots, T_p$ are disjoint. Also define A_j to be the event that $\{v_j \neq u_j\}$. Then, in any round in which v_j hasn't yet been assigned:

$$\begin{aligned} & \mathbb{P}[v_j \text{ is assigned and } A_j | v_j \text{ is assigned}] \\ = \; & \frac{|T_j|}{|E^+| - \sum_{k=i+1}^{j-1} |T_k|} \cdot \frac{|E^+| - \sum_{k=i+1}^{j-1} |T_k|}{|E^+|} = \frac{|T_j|}{|E^+|} \, . \end{aligned} \tag{5.3}$$

and hence v_j is chosen according the desired distribution.

We next show that each v_j is chosen independently. First observe that, conditioned on A_j, v_j is independent of $(v_1, \ldots, v_{j-1}, v_{j+1}, \ldots, v_p)$. Hence, it suffices to show that all A_j are independent. Note that the RHS of (5.3) does not depend on decisions made in previous rounds. Hence, we may deduce that A_j is independent of the outcome of rounds before v_j is assigned. Hence, for any $1 \leqslant i_1 < i_2 < \cdots < i_r \leqslant p$,

$$\begin{aligned}\mathbb{P}[A_{i_1} \cap \ldots \cap A_{i_r}] &= \mathbb{P}[A_{i_1}]\mathbb{P}[A_{i_2}|A_{i_1}] \ldots \mathbb{P}\big[A_{i_r}|A_{i_1} \cap \ldots \cap A_{i_{r-1}}\big] \\ &= \mathbb{P}[A_{i_1}]\mathbb{P}[A_{i_2}] \ldots \mathbb{P}[A_{i_r}] \ .\end{aligned}$$

The worst case for the expected number of samples is achieved when $p = |E^+|$ and each set is of size 1. For the algorithm not to terminate in a given round, we need $u \in \{u_{i+1}, \ldots, u_p\}$ and hence the index of the sampled u needs to strictly increase over previous rounds. The probability of this happening for r rounds is $\binom{m}{r}/m^r$ and the expected number of rounds which don't terminate is $\sum_{r=1}^{m} \binom{m}{r}/m^r \leqslant e - 1$. Because each non-terminating round involves two samples, the expected total number of samples is thus at most $2e - 1$. □

5.3.2 Connectivity Algorithm and Analysis

Our algorithm adapts a technique of Feige [43] for determining graph connectivity via a two step process. We first test whether G contains any connected components containing k or fewer nodes (for some $k < n$ to be chosen). If all of the connected components of G contain at least k nodes, we choose $O(n \log n/k)$ nodes at random, and verify that they are all connected to each other. Note that we can expect to have chosen a vertex from each connected component. If we find that all of our chosen vertices are connected, we conclude that G is connected; otherwise, we conclude that G is disconnected.

Our connectivity algorithm thus relies on algorithms for two problems: 1) determining whether the graph has any connected components below a certain size, and 2) determining whether a set of nodes is mutually connected in the graph. In the next two sections, we develop algorithms with sample/space tradeoffs for each of these two problems. We then use them in an algorithm for determining whether the graph G is connected.

5.3.2.1 Finding Small Components

Our first subproblem is to determine whether the graph has any connected components below a certain size. Given a node v, let the set of nodes in the connected component containing v be denoted $\mathrm{cc}(v)$.

Lemma 5.3.4. *Given a node v of G and a parameter r, we can distinguish between the case where $|\mathrm{cc}(v)| < r$ and the case where $|\mathrm{cc}(v)| > 2r$ with constant probability using $O(mr^2)$ samples and $\tilde{O}(1)$ space.*

Proof. We perform a sampling walk of length $O(mr^2)$ samples. While performing this walk, we maintain a 1.1-approximation of the number of distinct vertices visited using an F_0 estimator [73]. If the estimated number of vertices visited is at least $3r/2$, we conclude that $|\mathrm{cc}(v)| \geqslant r$; otherwise, we conclude that $|\mathrm{cc}(v)| \leqslant 2r$.

If $|\mathrm{cc}(v)| \leqslant r$, we will clearly visit at most r nodes. Our algorithm correctly concludes this so long as the F_0 estimator returns the promised approximation. If $|\mathrm{cc}(v)| \geqslant 2r$, we need to argue that in $O(mr^2)$ samples we will hit at least $2r$ distinct nodes (except with constant probability). This follows from a result by Barnes [17, Thm 1.3] that states that for any connected (multi-)graph, it takes $O(\mathcal{M}\mathcal{N})$ time in expectation to hit either $\mathcal{N}$ distinct nodes or $\mathcal{M}$ distinct edges. Using $\mathcal{M} = 2mr$ and $\mathcal{N} = 2r$ establishes the result. □

Theorem 5.3.5. *We can determine whether G has a connected component of size less than 2^k using $O(p)$ space and $\tilde{O}(2^k \cdot mn/p)$ samples for any $p \leqslant n$.*

Proof. Our algorithm has k rounds, each corresponding to a value $r = 1, 2, 4, \ldots, 2^{k-1}$. In each round we reach one of the following two conclusions: 1) G has no connected components with size in the range $[r, 2r]$ or 2) there exists a connected component of G of size $< 3r$. All graphs satisfy at least one of these conclusions. We then determine G has no connected component of size less than 2^k if we never reach the first conclusion.

At a given value of r, we choose $O(n \log n/r)$ nodes, so that we hit any connected component of at least r nodes with high probability. From each node, we perform $\tilde{O}(1)$ random walks of length $\tilde{O}(mr^2)$ samples; from Lemma 5.3.4 this will suffice to determine with high probability whether any of these nodes is in a connected component of size $\leqslant 2r$.

We choose p nodes at a time, and perform p walks in parallel. From Theorem 5.3.1 we can perform each set of p walks using $\tilde{O}(mr^2)$ samples. The number of samples required for each r value is then $O(\frac{n}{rp}mr^2) = O(mnr/p)$, and we thus require a total number of samples $O(mn2^k/p)$. $\square$

5.3.2.2 Checking Mutual Connectivity

The remaining subproblem is to determine whether a set of randomly-chosen nodes is mutually connected.

Lemma 5.3.6. *We can determine whether a set of $O(p)$ randomly-chosen nodes is mutually connected in G using $\tilde{O}(p)$ space and $\tilde{O}(mn^2/p^2)$ samples for any $p \leqslant n$.*

Proof. Feige [44] provides a method for testing whether two nodes s and t are connected using space $\tilde{O}(p)$ and using a total of $\tilde{O}(mn/p)$ random-walk steps. Their algorithm proceeds by choosing p "landmark" nodes; we then run $O(\log n)$ random walks from each landmark and from s and t. Each random walk is of length $\tilde{O}(mn/p^2)$. During these random walks we build up a union-find data structure indicating which sets of landmark nodes are connected. If at the end of the algorithm, s and t are in the same union-find component, we conclude that s and t are connected.

Since H is regular, the landmark selection process chooses each node with equal probability. Using Feige's algorithm on the p randomly-chosen landmarks determines whether this set of p nodes is mutually connected. The graph H has n nodes and mn edges, so from [44] each walk should be of length $\tilde{O}(mn^2/p^2)$. Using Theorem 5.3.1 we can simulate the p walks with total of $\tilde{O}(mn^2/p^2)$ samples. $\square$

We are now ready to prove our main connectivity result.

Theorem 5.3.7. *Given sampling access to a graph G, we can determine with high probability whether G is connected using $O(p \log n)$ space and $\tilde{O}(mn^2/p^2)$ samples, for any $p \leqslant n$.*

Proof. We use Theorem 5.3.5 with $2^k = n/p$ to verify that G has no connected components of size less than n/p. If it has such a component, then G is disconnected. If not, we choose $O(p \log n)$ random vertices, hitting each remaining component with high probability. Using Lemma 5.3.6, we test that these vertices are mutually connected. Since we have chosen enough vertices to hit every connected component, this suffices to show that the graph is connected. Each of the two subproblems requires $O(mn^2/p^2)$ samples and $\tilde{O}(p)$ space, so these are the sample and space requirements of our algorithm. $\square$

5.4 Lower Bound

Our lower bound result relies on a result by Andoni et al. [8] that implies that $\Omega(t/r^{2.5})$ space is required[3] to distinguish between the following two cases:

Case 1: We observe a sequence of t samples from a distribution p^{no} that is uniform on some subset $S \subseteq [t]$ of size $\Theta(t)$

[3]We note that an improvement to the work of Andoni et al. [8] was claimed in [54]; however, the proof given in [54] is incorrect and currently not known to be fixable.

Case 2: We observe a sequence of t samples from a distribution p^{yes} such that $p^{\text{yes}}_i = r/t$ for some $i \in [t]$ and uniform on some subset $T \subseteq [t] \backslash \{i\}$.

By combining this result with a hashing technique we establish the following result.

Theorem 5.4.1. *Any constant factor approximation of $F_k(D)$ given a sequence of t IID samples on $[n]$ requires $\Omega(n^{2.5(1-1/k)}/(\log^{2.5} n \cdot t^{1.5}))$ space.*

Proof. Let $h : [t] \to [n]$ be a fully-random hash function and consider the problem of distinguishing p^{no} and p^{yes} where we set $r = c \log n \cdot t \cdot n^{1/k-1}$ for some constant $c > 0$. By applying h on each distribution (i.e., applying h to each observed sample) we generate two new distributions q^{no} and q^{yes} over $[n]$ where:

$$q^{\text{no}}_i = \sum_{j:h(j)=i} p^{\text{no}}_i \quad \text{and} \quad q^{\text{yes}}_i = \sum_{j:h(j)=i} p^{\text{yes}}_i$$

Note that with high probability $\max_i q^{\text{no}}_i = O(\log n \cdot 1/n)$ and hence

$$F_k(q^{\text{no}}) \leqslant n \cdot O((\log n \cdot 1/n)^k) = O(\log^k n \cdot n^{1-k}) \ .$$

However, $\max_i q^{\text{yes}}_i \geqslant r/n$ and so

$$F_k(q^{\text{yes}}) \geqslant r^k/t^k = c^k \cdot \log^k n \cdot n^{1-k} \ .$$

Hence, for a sufficiently large value of the constant $c > 0$ we can ensure that any constant approximation of F_k distinguishes between q^{yes} and q^{no} and hence, also distinguishes between p^{yes} and p^{no}. However, by the result of Andoni et al. [8] we know that this requires $\Omega(t/r^{2.5}) = \Omega(n^{2.5(1-1/k)}/(\log^{2.5} n \cdot t^{1.5}))$ space. □

CHAPTER 6
CONCLUSIONS

In this thesis, we have introduced several new models of streaming computation. We used computational reductions to adapt existing algorithms to these models, and to move from a small number of new algorithmic "primitives" to easily develop new algorithms. Considering reductions as an important object of study led directly to the development of algorithms.

We hope that future work will study reductions between streaming problems in more detail. Many interesting streaming problems can be solved only by probabilistic approximation algorithms. Unfortunately, classical approximation complexity has largely considered nondeterministic classes rather than probabilistic models. We suspect that this is one reason that a more formal study of streaming complexity classes has been elusive.

BIBLIOGRAPHY

[1] Kook Jin Ahn and Sudipto Guha. Graph sparsification in the semi-streaming model. In *ICALP (2)*, pages 328–338, 2009. Referenced on pg. 45.

[2] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *SODA*, pages 459–467, 2012. Referenced on pp. 43, 45, 47, and 54.

[3] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *PODS*, pages 5–14, 2012. Referenced on pp. 43, 45, and 48.

[4] Noga Alon, Yossi Matias, and Mario Szegedy. The Space Complexity of Approximating the Frequency Moments. *Journal of Computer and System Sciences*, 58(1):137–147, February 1999. Referenced on pp. 5, 6, 28, 31, 38, 41, and 71.

[5] Ingo Althöfer, Gautam Das, David P. Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9:81–100, 1993. Referenced on pg. 55.

[6] Alexandr Andoni, Assaf Goldberger, Andrew McGregor, and Ely Porat. Homomorphic fingerprints under misalignments. In *STOC*, page 931, June 2013. Referenced on pg. 26.

[7] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Streaming algorithms via precision sampling. *FOCS*, 2011. Referenced on pp. 10 and 61.

[8] Alexandr Andoni, Andrew McGregor, Krzysztof Onak, and Rina Panigrahy. Better bounds for frequency moments in random-order streams. *CoRR*, abs/0808.2222, 2008. Referenced on pp. 85 and 86.

[9] Arvind Arasu and Gurmeet Singh Manku. Approximate counts and quantiles over sliding windows. In *PODS*, pages 286–296, 2004. Referenced on pp. 16 and 44.

[10] Ahmed Ayad and Jeffrey F. Naughton. Static optimization of conjunctive queries with sliding windows over infinite streams. In *SIGMOD Conference*, pages 419–430, 2004. Referenced on pp. 16 and 44.

[11] Brian Babcock, Mayur Datar, and Rajeev Motwani. Sampling from a moving window over streaming data. In *SODA*, pages 633–634, 2002. Referenced on pp. 16 and 44.

[12] Brian Babcock, Mayur Datar, Rajeev Motwani, and Liadan O'Callaghan. Maintaining variance and k-medians over data stream windows. In *PODS*, pages 234–243, 2003. Referenced on pp. 16 and 44.

[13] Ajesh Babu, Nutan Limaye, J Radhakrishnan, and Girish Varma. Streaming algorithms for language recognition problems. *Theoretical Computer Science*, 494:13–23, 2013. Referenced on pg. 60.

[14] Ajesh Babu, Nutan Limaye, and Girish Varma. Streaming algorithms for some problems in log-space. *Electronic Colloquium on Computational Complexity (ECCC)*, 17:94, 2010. Referenced on pg. 20.

[15] Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Sampling algorithms: lower bounds and applications. In *STOC*, pages 266–275, 2001. Referenced on pg. 72.

[16] Ziv Bar-Yossef, Ravi Kumar, and D Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *SODA*, pages 623–632, January 2002. Referenced on pp. 15 and 60.

[17] Greg Barnes and Uriel Feige. Short Random Walks on Graphs. *SIAM Journal on Discrete Mathematics*, 9(1):19, 1996. Referenced on pg. 83.

[18] Surender Baswana. Streaming algorithm for graph spanners—single pass and constant processing time per edge. *Information Processing Letters*, 2008. Referenced on pg. 45.

[19] Radu Berinde, Piotr Indyk, Graham Cormode, and Martin J Strauss. Space-optimal heavy hitters with strong error bounds. *ACM Transactions on Database Systems*, 35(4):1–28, November 2010. Referenced on pg. 11.

[20] B Bollobás. *Extremal graph theory.* Academic Press, New York, 1978. Referenced on pg. 56.

[21] Vladimir Braverman, Ran Gelles, and Rafail Ostrovsky. How to catch ℓ_2-heavy-hitters on sliding windows. In *COCOON*, pages 638–650, 2013. Referenced on pp. 16, 44, and 48.

[22] Vladimir Braverman and Rafail Ostrovsky. Smooth Histograms for Sliding Windows. In *FOCS*, pages 283–293. Ieee, October 2007. Referenced on pp. 16, 44, 48, and 50.

[23] Amit Chakrabarti, Graham Cormode, and Andrew McGregor. Robust lower bounds for communication and stream computation. In *STOC*, pages 641–650, 2008. Referenced on pg. 70.

[24] Amit Chakrabarti, T. S. Jayram, and Mihai Patrascu. Tight lower bounds for selection in randomly ordered streams. In *SODA*, pages 720–729, 2008. Referenced on pg. 70.

[25] Amit Chakrabarti and Sagar Kale. Submodular Maximization Meets Streaming: Matchings, Matroids, and More. In *IPCO*, September 2014. Referenced on pp. 17, 18, 58, and 64.

[26] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding Frequent Items in Data Streams. In *ICALP '02: Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, pages 693–703, London, UK, 2002. Springer-Verlag. Referenced on pg. 10.

[27] Bernard Chazelle, Ronitt Rubinfeld, and Luca Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM J. Comput.*, 34(6):1370–1379, 2005. Referenced on pg. 54.

[28] Steve Chien, Katrina Ligett, and Andrew McGregor. Space-efficient estimation of robust statistics and distribution testing. In *ICS*, pages 251–265, 2010. Referenced on pg. 70.

[29] Don Coppersmith and Ravi Kumar. An improved data stream algorithm for frequency moments. In *SODA*, pages 151–156, 2004. Referenced on pg. 74.

[30] Graham Cormode and Marios Hadjieleftheriou. Methods for finding frequent items in data streams. *The VLDB Journal*, 19(1):3–20, December 2010. Referenced on pg. 11.

[31] Graham Cormode and S. Muthukrishnan. An improved data stream summary: The Count-Min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005. Referenced on pg. 7.

[32] Michael Crouch and Andrew McGregor. Periodicity and cyclic shifts via linear sketches. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 158–170, 2011. Referenced on pg. 23.

[33] Michael Crouch, Andrew McGregor, and Daniel Stubbs. Dynamic Graphs in the Sliding-Window Model. *ESA*, 2013. Referenced on pp. 18, 43, 54, 58, 64, and 65.

[34] Artur Czumaj and Leszek Gccasieniec. On the Complexity of Determining the Period of a String. In Raffaele Giancarlo and David Sankoff, editors, *Combinatorial Pattern Matching*, volume 1848 of *Lecture Notes in Computer Science*, pages 412–422. Springer Berlin / Heidelberg, 2000. Referenced on pg. 23.

[35] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining Stream Statistics over Sliding Windows. *SIAM Journal on Computing*, 31(6):1794, 2002. Referenced on pp. 15, 16, 44, and 65.

[36] DP Dubhashi and A Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009. Referenced on pp. 20 and 21.

[37] Klim Efremenko and Omer Reingold. How Well Do Random Walks Parallelize? In Irit Dinur, Klaus Jansen, Joseph Naor, and José Rolim, editors, *APPROX-RANDOM*, volume 5687 of *Lecture Notes in Computer Science*, pages 476–489, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. Referenced on pg. 78.

[38] Michael Elkin. Streaming and fully dynamic centralized algorithms for constructing and maintaining sparse spanners. *ACM Transactions on Algorithms*, 7(2):1–17, March 2011. Referenced on pg. 45.

[39] D. Eppstein, Z. Galil, G.F. Italiano, and A. Nissenzweig. Sparsification: a technique for speeding up dynamic graph algorithms. *Journal of the ACM*, 44(5):669–696, 1997. Referenced on pg. 43.

[40] Leah Epstein, Asaf Levin, Julián Mestre, and Danny Segev. Improved Approximation Guarantees for Weighted Matching in the Semi-streaming Model. *SIAM Journal on Discrete Mathematics*, 25(3):1251–1265, January 2011. Referenced on pp. 17, 18, 43, 45, 54, 58, 64, and 67.

[41] Funda Ergün, Hossein Jowhari, and Mert Salam. Periodicity in Streams. *RANDOM*, 2010. Referenced on pp. v, 23, 25, 26, 28, 40, and 41.

[42] Funda Ergün, S. Muthukrishnan, and Cenk Sahinalp. Periodicity testing with sublinear samples and space. *ACM Transactions on Algorithms*, 6(2):1–14, March 2010. Referenced on pp. 23, 26, and 28.

[43] Uriel Feige. A fast randomized logspace algorithm for graph connectivity. *Theor. Comput. Sci.*, 169(2):147–160, 1996. Referenced on pp. 71 and 82.

[44] Uriel Feige. A Spectrum of TimeSpace Trade-offs for Undirected s-t Connectivity. *Journal of Computer and System Sciences*, 54(2):305–316, April 1997. Referenced on pp. 71, 77, 84, and 85.

[45] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2-3):207–216, December 2005. Referenced on pp. 5, 8, 17, 43, 45, 55, 58, and 64.

[46] Joan Feigenbaum, Sampath Kannan, and Jian Zhang. Computing diameter in the streaming and sliding-window models. *Algorithmica*, 41(1):25–41, 2004. Referenced on pp. 16 and 44.

[47] R. A. Fisher. On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 222:309–368, 1922. Referenced on pp. 18 and 70.

[48] Philippe Flajolet and G Nigel Martin. Probabilistic Counting. In *FOCS*, pages 76–82. IEEE, 1983. Referenced on pg. 4.

[49] Wai Shing Fung, Ramesh Hariharan, Nicholas J. A. Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. In *STOC*, pages 71–80, 2011. Referenced on pg. 48.

[50] Sumit Ganguly. Lower bounds on frequency estimation of data streams. In *Computer ScienceTheory and Applications*, pages 204–215. Springer, 2008. Referenced on pg. 20.

[51] Anna C. Gilbert and Piotr Indyk. Sparse Recovery Using Sparse Matrices. *Proceedings of the IEEE*, 98(6):937–947, June 2010. Referenced on pg. 7.

[52] Ashish Goel, Michael Kapralov, and Ian Post. Single pass sparsification in the streaming model with edge deletions. *CoRR*, abs/1203.4900, 2012. Referenced on pg. 45.

[53] Michael Greenwald and Sanjeev Khanna. Efficient online computation of quantile summaries. In *ACM International Conference on Management of Data*, pages 58–66, 2001. Referenced on pg. 70.

[54] Sudipto Guha and Zhiyi Huang. Revisiting the direct sum theorem and space lower bounds in random order streams. In *ICALP*, pages 513–524, 2009. Referenced on pg. 85.

[55] Sudipto Guha and Andrew McGregor. Space-efficient sampling. In *AISTATS*, pages 169–176, 2007. Referenced on pg. 70.

[56] Sudipto Guha and Andrew McGregor. Stream order and order statistics: Quantile estimation in random-order streams. *SIAM Journal on Computing*, 38(5):2044–2059, 2009. Referenced on pg. 70.

[57] G H Hardy and E M Wright. *An Introduction to The Theory of Numbers (Fourth Edition)*. Oxford University Press, 1960. Referenced on pg. 32.

[58] Juris Hartmanis, Neil Immerman, and Stephen R. Mahaney. One-Way Log-Tape Reductions. In *FOCS*, pages 65–72, 1978. Referenced on pg. 4.

[59] Juris Hartmanis and Stephen R. Mahaney. Languages Simultaneously Complete for One-Way and Two-Way Log-Tape Automata. *SIAM Journal on Computing*, 10(2):383, 1981. Referenced on pg. 4.

[60] Martin E Hellman and Thomas M Cover. Learning with finite memory. *The Annals of Mathematical Statistics*, pages 765–782, 1970. Referenced on pg. 70.

[61] Monika Rauch Henzinger and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM*, 46(4):502–516, 1999. Referenced on pg. 43.

[62] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001. Referenced on pg. 43.

[63] John Hopcroft and J. D. Ullman. Some Results on Tape-Bounded Turing Machines. *Journal of the ACM*, 16(1):168–177, January 1969. Referenced on pg. 4.

[64] Neil Immerman. *Descriptive Complexity.* Springer, 1999. Referenced on pg. 4.

[65] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53(3):307–323, 2006. Referenced on pg. 7.

[66] Piotr Indyk, N Koudas, and S. Muthukrishnan. Identifying representative trends in massive time series data sets using sketches. In *VLDB*, pages 363–372, 2000. Referenced on pg. 23.

[67] Piotr Indyk and Andrew McGregor. Declaring independence via the sketching of sketches. In Shang-Hua Teng, editor, *SODA*, pages 737–745. SIAM, 2008. Referenced on pp. 26 and 38.

[68] 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. Referenced on pp. 61 and 71.

[69] G.F. Italiano, D. Eppstein, and Z. Galil. Dynamic graph algorithms. *Algorithms and Theory of Computation Handbook, CRC Press*, 1999. Referenced on pg. 43.

[70] Thomas A Jenkyns. The efficacy of the greedy algorithm. *Proceedings of the 7th Southeastern Conference on Combinatorics, Graph Theory and Computing*, pages 341–350, 1976. Referenced on pg. 59.

[71] Hossein Jowhari, Mert Salam, and Gábor Tardos. Tight bounds for Lp samplers, finding duplicates in streams, and related problems. In *PODS*, pages 49–58. ACM Press, June 2011. Referenced on pp. 10 and 35.

[72] Daniel M Kane, Jelani Nelson, and David P Woodruff. An optimal algorithm for the distinct elements problem. In *PODS*, pages 41–52, 2010. Referenced on pg. 7.

[73] Daniel M Kane, Jelani Nelson, and David P Woodruff. On the Exact Space Complexity of Sketching and Streaming Small Norms. In Moses Charikar, editor, *SODA*, pages 1161–1178. SIAM, 2010. Referenced on pg. 83.

[74] Michael Kapralov. Better bounds for matchings in the streaming model. In *SODA*, pages 1679–1697, 2013. Referenced on pp. 43 and 50.

[75] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A Model of Computation for MapReduce. In *SODA*, pages 938–948, 2010. Referenced on pg. 65.

[76] Jack Koplowitz. Necessary and sufficient memory size for m-hypothesis testing. *Information Theory, IEEE Transactions on*, 21(1):44–46, 1975. Referenced on pg. 70.

[77] Eyal Kushilevitz and Noam Nisan. *Communication Complexity*, volume 2006. Cambridge University Press, 1997. Referenced on pg. 55.

[78] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. Filtering: a method for solving graph problems in MapReduce. In *SPAA*, New York, New York, USA, 2011. ACM Press. Referenced on pp. 17, 18, 58, 64, and 65.

[79] P. M. Lewis, R. E. Stearns, and Juris Hartmanis. Memory bounds for recognition of context-free and context-sensitive languages. In *6th Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1965)*, pages 191–202. IEEE, 1965. Referenced on pg. 4.

[80] Frédéric Magniez, Claire Mathieu, and Ashwin Nayak. Recognizing well-parenthesized expressions in the streaming model. In Leonard J Schulman, editor, *STOC*, pages 261–270. ACM, 2010. Referenced on pp. 19 and 60.

[81] Andrew McGregor. Finding graph matchings in data streams. *APPROX-RANDOM*, 2005. Referenced on pp. 17, 43, 58, and 64.

[82] Andrew McGregor. *Processing data streams.* PhD thesis, University of Pennsylvania, 2007. Referenced on pg. 4.

[83] Andrew McGregor, A. Pavan, Srikanta Tirthapura, and David P. Woodruff. Space-efficient estimation of statistics over sub-sampled streams. In *PODS*, pages 273–282, 2012. Referenced on pg. 70.

[84] Andrew McGregor and Paul Valiant. The shifting sands algorithm. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 453–458, 2012. Referenced on pg. 70.

[85] Silvio Micali and Vijay V. Vazirani. An O(sqrt(|V|) |E|) algorithm for finding maximum matching in general graphs. In *FOCS*, pages 17–27, 1980. Referenced on pg. 68.

[86] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, 2005. Referenced on pg. 78.

[87] Morteza Monemizadeh and David P Woodruff. 1-Pass Relative-Error L_p-Sampling with Applications. In *SODA*, 2010. Referenced on pp. 28, 35, and 74.

[88] Robert Morris. Counting large numbers of events in small registers. *Commun. ACM*, 21(10):840–842, October 1978. Referenced on pg. 4.

[89] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995. Referenced on pg. 40.

[90] J. Ian Munro and M.S. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12(3):315–323, November 1980. Referenced on pp. 4 and 70.

[91] S. Muthukrishnan. Data Streams: Algorithms and Applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2006. Referenced on pg. 5.

[92] S. Muthukrishnan. Stochastic data streams. In *MFCS*, page 55, 2009. Referenced on pg. 70.

[93] Jelani Nelson. *Sketching and Streaming High-Dimensional Vectors*. PhD thesis, Massachusetts Institute of Technology, 2011. Referenced on pg. 4.

[94] Noam Nisan. Pseudorandom Generators for Space-bounded Computation. *Combinatorica*, 12:449–461, 1992. Referenced on pg. 7.

[95] Benny Porat and Ely Porat. Exact and Approximate Pattern Matching in the Streaming Model. In *FOCS*, pages 315–323, October 2009. Referenced on pp. 23 and 40.

[96] Piyush Rai, Hal Daumé III, and Suresh Venkatasubramanian. Streamed learning: One-pass SVMs. In *IJCAI*, pages 1211–1216, 2009. Referenced on pg. 70.

[97] Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM*, 55(4):1–24, September 2008. Referenced on pg. 71.

[98] Florin Rusu and Alin Dobra. Sketching sampled data streams. In *ICDE*, pages 381–392, 2009. Referenced on pg. 70.

[99] R. E. Stearns, Juris Hartmanis, and P. M. Lewis. Hierarchies of memory limited computations. In *6th Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1965)*, pages 179–190. IEEE, 1965. Referenced on pg. 4.

[100] R.E. Tarjan. *Data Structures and Network Algorithms*. SIAM, Philadelphia, 1983. Referenced on pg. 46.

[101] Mikkel Thorup. Near-optimal fully-dynamic graph connectivity. In *STOC*, pages 343–350, 2000. Referenced on pg. 43.

[102] H Vollmer. *Introduction to circuit complexity: a uniform approach*. Springer-Verlag, 1999. Referenced on pg. 4.

[103] David P. Woodruff. The average-case complexity of counting distinct elements. In *ICDT*, pages 284–295, 2009. Referenced on pg. 70.

[104] Mariano Zelke. Weighted matching in the semi-streaming model. *Algorithmica*, pages 669–680, 2012. Referenced on pp. 17, 43, 58, and 64.