View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by ScholarWorks@UMass Amherst

University of Massachusetts Amherst

ScholarWorks@UMass Amherst

Doctoral Dissertations

Dissertations and Theses

October 2018

Data Stream Algorithms for Large Graphs and High Dimensional Data

Hoa Vu

Follow this and additional works at: https://scholarworks.umass.edu/dissertations_2

Part of the Theory and Algorithms Commons

Recommended Citation

Vu, Hoa, "Data Stream Algorithms for Large Graphs and High Dimensional Data" (2018). *Doctoral Dissertations*. 1404.
https://scholarworks.umass.edu/dissertations_2/1404

This Open Access Dissertation is brought to you for free and open access by the Dissertations and Theses at ScholarWorks@UMass Amherst. It has been accepted for inclusion in Doctoral Dissertations by an authorized administrator of ScholarWorks@UMass Amherst. For more information, please contact scholarworks@library.umass.edu.

DATA STREAM ALGORITHMS FOR LARGE GRAPHS AND HIGH DIMENSIONAL DATA

A Dissertation Presented

by

HOA T. VU

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 2018

College of Information and Computer Sciences

© Copyright by Hoa T. Vu 2018

All Rights Reserved

DATA STREAM ALGORITHMS FOR LARGE GRAPHS AND HIGH DIMENSIONAL DATA

A Dissertation Presented

by

HOA T. VU

Approved as to style and content by:

Andrew McGregor, Chair

Barna Saha, Member

Arya Mazumdar, Member

Marco Duarte, Member

James Allan, Chair
College of Information and Computer Sciences

ACKNOWLEDGMENTS

I was very fortunate to work with my advisor Andrew McGregor. This thesis would not be possible without his help and guidance. Andrew has always been an understanding and patient advisor.

I would like to thank Marco Duarte, Arya Mazumda and Barna Saha for agreeing to be in my committee and for their time and suggestions.

I am fortunate to work with and learn from great co-authors including Sofya Vorotnikova, David Tench, Branislav Kveton, Muthu Muthukrisnan, Michael Bender, Shikha Singh, Samuel McCauley, and Yikun Xian. I am especially grateful to Branislav Kveton and Muthu Muthukrisnan for being great mentors during and after my internship at Adobe Research.

Special thanks go to Leeanne Lerlerc, who has been helping me with paperwork and administrative issues during my years at UMass.

I would like to thank my friends for their help and support during my graduate study. I am particularly thankful to my friend Pham Thu Trang, who is always willing to listen to my random stories from time to time.

Finally, this thesis is dedicated to my parents and my sister who never understand what I am doing but always offer me constant support.

ABSTRACT

DATA STREAM ALGORITHMS FOR LARGE GRAPHS AND HIGH DIMENSIONAL DATA

SEPTEMBER 2018

HOA T. VU

B.Sc., THE OHIO STATE UNIVERSITY

M.Sc., UNIVERSITY OF MASSACHUSETTS, AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Andrew McGregor

In contrast to the traditional random access memory computational model where the entire input is available in the working memory, the data stream model only provides sequential access to the input. The data stream model is a natural framework to handle large and dynamic data. In this model, we focus on designing algorithms that use sublinear memory and a small number of passes over the stream. Other desirable properties include fast update time, query time, and post processing time.

In this dissertation, we consider different problems in graph theory, combinatorial optimization, and high dimensional data processing.

The first part of this dissertation focuses on algorithms for graph theory and combinatorial optimization. We present new results for the problems of finding the densest subgraph, counting the number of triangles, finding max cut with bounded components, and finding the maximum k set coverage.

The second part of this dissertation considers problems in high dimensional data streams. In this setting, each stream item consists of multiple coordinates corresponding to different attributes. We consider the problem of testing or learning about the relationships among the attributes, and the problem of finding heavy hitters in subsets of attributes.

TABLE OF CONTENTS

Page

ACKNOWLEDGMENTS .... **iv**

ABSTRACT .... **v**

CHAPTER

1. INTRODUCTION **1**

1.1 Related Data Stream Models 2
1.2 Graph Theory and Combinatorial Optimization 2
1.3 High Dimensional Data Streams Processing 6

2. BASIC BACKGROUND AND NOTATION **11**

2.1 Basic Notation 11
2.2 Graph Theory Notation and Convention 12
2.3 Concentration Bounds 12

3. FAST ℓ_p SAMPLING AND ITS APPLICATION TO FINDING THE APPROXIMATE DENSEST SUBGRAPH **14**

3.1 Introduction 14
3.2 Fast ℓ_p Sampling Algorithm 15
3.3 Application: Finding Approximate Densest Subgraph 26

4. TRIANGLE COUNTING IN THE ADJACENCY LIST MODEL **35**

4.1 Introduction and Related Work 35
4.2 One Pass and $\tilde{O}(\epsilon^{-2}m/\sqrt{T})$ Space 39
4.3 Two Passes and $\tilde{O}(\epsilon^{-2}m^{3/2}/T)$ Space 43

5. FINDING APPROXIMATE MAXIMUM COVERAGE IN THE STREAMING SET MODEL **47**

5.1 Introduction and Related Work 47
5.2 Algorithms for Maximum k-Set Coverage 53

5.2.1 $(1 - 1/e - \epsilon)$ Approximation in One Pass and $\tilde{O}(\epsilon^{-2}m)$ Space 54
5.2.2 $(1 - 1/e - \epsilon)$ Approximation in $O(\epsilon^{-1})$ Passes and $\tilde{O}(\epsilon^{-2}k)$ Space 57
5.2.3 Removing Assumptions via Guessing, Sampling, and Sketching 61
5.2.4 Other Algorithmic Results 64

5.3 Algorithms for Maximum k-Vertex Coverage 72
5.4 Lower Bounds 75

6. FINDING CAPACITATED MAXCUT IN THE ADJACENCY LIST MODEL 79

6.1 Introduction and Related Work 79
6.2 Algorithms for Capacitated MaxCut 82
6.3 New Algorithms for Non-monotone Submodular Maximization 88

7. TESTING BAYESIAN NETWORKS IN DATA STREAMS 94

7.1 Introduction and Related Work 94
7.2 Algorithms for Estimating $\mathcal{E}_p(G)$ 97
7.3 Lower Bound for Estimating $\mathcal{E}_p(G)$ 99

8. FINDING SUBCUBE HEAVY HITTERS IN DATA STREAMS 105

8.1 Introduction and Related Work 105
8.2 The Sampling Algorithm 110
8.3 Algorithms under The Near-Independence Assumption 113
8.4 Algorithms under The Naive Bayes Assumption 117

9. FUTURE WORK 124

9.1 Graph Theory and Combinatorial Optimization 124
9.2 High Dimensional Data Streams 125

BIBLIOGRAPHY 126

CHAPTER 1

INTRODUCTION

Most traditional RAM algorithms scale poorly to large datasets. For this purpose, new computational models such as the data stream model and distributed models have been introduced. The data stream model is perhaps the most popular since it captures two important restrictions of large datasets. Specifically, in the data stream model, we only have one-way access to the data since the random access memory is insufficient to store the entire input. Furthermore, it is a natural model to work with dynamic data. In addition to the memory restriction, the data stream model also considers other performance factors such as the number of passes, the update time, the query time and the post-processing time.

Our work focuses on developing data stream algorithms for massive graphs which arise in many applications. Some example include webpages and hyperlinks, papers and citations, social network graphs and telephone networks. We also study algorithms for massive hypergraphs which model the set-element relationships. Example applications of hypergraphs include sensor allocation, information retrieval and influence maximization.

In the second part of this thesis, we consider the model based approach in high dimensional data streams. In this setting, the stream consists of high dimensional items. Each dimension corresponds to an attribute. The first problem is to test a graphical model based on the observed data stream. We then consider the problem of identifying heavy hitters of subsets of attributes.

1.1 Related Data Stream Models

The graph stream model. In this model, the stream consists of edge insertions of a graph on n nodes. If the stream also allows edge deletions, we refer to this model as the *dynamic graph stream model.*

In the case there is no edge deletion, the *adjacency list model* assumes that edges incident to the same node appear consecutively whereas the *arbitrary order model* assumes that edges arrive in an arbitrary order. For weighted graphs, an edge insertion or deletions provides the weight of the corresponding edge.

The streaming set model. This model is a natural extension of the graph stream model. Given a universe $U = \{1, 2, \ldots, n\}$, the stream consists of sets that are subsets of U. In particular, the stream consists of m sets $S_1, \ldots, S_m$ and each S_i is encoded as the list of elements in that set. This model extends the graph stream model to hypergraphs since sets can also be viewed as hyperedges of the graph on n nodes.

High dimensional data streams. In this model, the stream consists of d-dimensional items $x_1, x_2, \ldots, x_m$ of a database where each $x_i \in [n]^d$. This model captures databases where items have many attributes.

1.2 Graph Theory and Combinatorial Optimization

Finding the densest subgraph in dynamic graph streams. In dynamic graph streams, Bahmani et al. [20] showed that a constant approximation algorithm requires $\Omega(n)$ space. One of our main results is to show that it is possible to obtain a $1-\epsilon$ approximation of the densest subgraph in dynamic graph streams using $\tilde{O}(\epsilon^{-2}n)$ space. Our result improves upon the work of Bhattacharya et al. [25]. They presented two algorithms that use similar space to our algorithm and process updates in polylog(n) amortized time. The first algorithm returns a $(1/2-\epsilon)$ approximation of the maximum density of the final graph while the second outputs a $(1/4-\epsilon)$ approximation of the current maximum density after every update while still using only polylog(n) time per-update.

For our algorithms, we present a fast polylog(n) update time in the worst case. For constant ϵ, our algorithm is optimal up to polylogarithmic factors. The design of algorithm consists of two steps. First, we need to argue that uniformly sample the edges using ℓ_p sampling obtains a good estimate for the densest subgraph. Regarding the update time, we design a fast ℓ_p sampling algorithm that improves the update time from $\Omega(n)$ to $O(\operatorname{polylog} n)$.

Counting triangles in the adjacency list model. Counting triangles is a canonical problem in both the RAM model and the data stream model. In the data stream model, it has been shown that $\Omega(n^2)$ space is required just to test if the graph is triangle-fre in the worst case [21]. Therefore, the problem has been studied based by parameterizing of the number of triangles T (or a lower bound of T) in the graph. We present two main algorithms that $(1+\epsilon)$ estimate the number of triangles for the adjacency list order model where one is suitable for processing graphs with many triangles (in particular, when $T \geq m$) and the other is suitable for processing graphs with fewer triangles (i.e., $T \leq m$). Specifically, we present

1. A single-pass algorithm using $\tilde{O}(\epsilon^{-2}m/\sqrt{T})$ space and
2. A two-pass algorithm using $\tilde{O}(\epsilon^{-2}m^{3/2}/T)$ space.

Note that $m/\sqrt{T}$ space has become natural goal in the context of estimating the number of triangles. In particular, any constant pass algorithm in the arbitrary order model required this amount of space when $m = \Theta(n\sqrt{T})$ and there is an existing two-pass algorithm that returns a 3-approximation using this amount of space [51]. Furthermore, Jha et al. [81] showed that this space was sufficient for additively approximating T. Unfortunately, Braverman et al. [29] showed it was insufficient for achieving multiplicative approximation via a *single-pass* algorithm in the arbitrary order model. The significance of our results is showing that $\approx m/\sqrt{T}$ space is sufficient for $1+\epsilon$ approximation if we are given a single pass over a stream in adjacency list order or two passes over a stream in arbitrary order.

However, it is possible to improve upon $m/\sqrt{T}$ space when T is large and other algorithm do just this. At a high level, the main difference between the two types of algorithms we present is as follows. The $m/\sqrt{T}$ dependence arises when we focus on distinguishing between edges that are involved in many triangles and those that are not, whereas the $m^{3/2}/T$ dependence arises when we distinguish between high and low degree nodes. Our algorithms can also be implemented in arbitrary order model by using additional passes [114].

Approximating max capacitated cut in the adjacency list model. We consider the streaming capacitated max $(t+1)$-cut problem where t parts are bounded. This problem and its variations have been studied previously in various work [2, 65, 68, 142]. The goal is to find t disjoint sets $S_1, S_2, \ldots, S_t$ such that $|S_i| \leq k$ and the number of edges across the parts is maximized. The optimal solution is defined as follows.

$$\mathrm{OPT} := \underset{\substack{\text{disjoint } S_1, S_2, \ldots, S_t \subset V \\ |S_i| \leq k}}{\arg\max} \left| \{(u,v) \in E : |\{u,v\} \cap S_i| = 1\} \right| .$$

In particular, the algorithm's output is t disjoint sets of nodes $S_1, \ldots, S_t$. We study this problem in the adjacency list model. We want to design algorithms, with constant approximations, that use space depending only on k. Our main results are:

1. For the case $t = 1$, we present a) a single-pass, $\tilde{O}(k^2)$-space algorithm that finds a $0.4 - o(1)$ approximation and b) a two-pass, $\tilde{O}(k^3/\epsilon)$-space algorithm that finds a $0.5 - \epsilon$ approximation.

2. As for the more general case $t > 1$, we present a single-pass, $\tilde{O}(k^2)$ space algorithm that finds a $6/11 - o(1)$ approximation.

Motivated by the case $t = 1$, we design new algorithms for non-monotone submodular maximization under a cardinality constraint. By allowing more passes or space, we improve the approximation given by Chekuri et al. [41].

Finding the maximum k set coverage in the streaming set model. The *maximum set coverage problem* is a classic NP-Hard problem that has a wide range of applications including facility and sensor allocation [101], information retrieval [8], influence maximization in marketing strategy design [93], and the blog monitoring problem where we want to choose a small number of blogs that cover a wide range of topics [128]. In this problem, we are given a set system of m sets that are subsets of a universe $[n] := \{1, \ldots, n\}$. The goal is to find the k sets whose union covers the largest number of distinct elements.

It is well-known that the greedy algorithm, which greedily picks the set that covers the most number of uncovered elements, is a $1 - 1/e$ approximation algorithm. Furthermore, unless $P = NP$, this approximation factor is the best possible [62].

For the maximum set coverage problem, Saha and Getoor [128] gave a swap based 1/4 approximation algorithm that uses a single pass and $\tilde{O}(kn)$ space. Recently, Badanidiyuru et al. [17] gave a generic single-pass algorithm for maximizing a monotone submodular function on the stream's items subject to the cardinality constraint that at most k objects are selected. A careful adaptation to the maximum set coverage problem uses $\tilde{O}(\epsilon^{-1}n)$ space.

Our main goal is to find constant approximations using sublinear $o(mn)$ space. In particular, we present different algorithms and a lower bound giving evidence that our algorithms are either optimal or near optimal.

1. A polynomial time data stream algorithms that achieve a $1 - 1/e - \epsilon$ approximation for arbitrarily small ϵ. The first algorithm uses one pass and $\tilde{O}(\epsilon^{-2}m)$ space whereas the second algorithm uses $O(\epsilon^{-1})$ passes and $\tilde{O}(\epsilon^{-2}k)$ space.
2. A lower bound of $\Omega(m/k^2)$ space for any constant pass (randomized) algorithm to achieve an approximation factor better than $1 - (1 - 1/k)^k$ with probability at least 0.99; this holds even if the algorithm is permitted exponential time
3. With exponential time and $\tilde{O}(\epsilon^{-2}m \cdot \min(k, \epsilon^{-1}))$ space we observe that a $1 - \epsilon$ approximation is possible in a single pass.

For this problem, we also consider different constraints such as the budgeted constraint and the group cardinality constraint (partition matroid). Finally, we also consider a special case of this problem which is the maximum k-vertex coverage problem. This problem asks for k nodes that covers the most number of edges. We, however, study this problem in the dynamic graph stream model. We show a matching upper bound and lower bound, up to polylogarithmic factors, of $\Theta(N)$ for a constant approximation where N is the number of nodes in the graphs.

1.3 High Dimensional Data Streams Processing

Testing Bayesian networks. In this setting, the stream consists of m items that are n-tuples (i.e., each item has n coordinates) that empirically defines a joint distribution of n random variables $X_1, X_2, \ldots, X_n$ where each X_i has range $[k] := \{1, 2, \ldots k\}$. The empirical joint probability mass function of these variables is defined as

$$\begin{aligned}\mathcal{P}(x_1, \ldots, x_n) &= \Pr\left[X_1 = x_1 \text{ and } X_2 = x_2 \text{ and } \ldots \text{ and } X_n = x_n\right] \\ &:= \frac{c(x_1, x_2, \ldots, x_n)}{m} \ ,\end{aligned}$$

where $c(x_1, x_2, \ldots, x_n)$ is the *count* of the number of tuples equal to $(x_1, x_2, \ldots, x_n)$. This is also the probability of a random stream item is the tuple $(x_1, \ldots, x_n)$. Another terminology for the above definition is the *frequency ratio* of the tuple (or joint values) $(x_1, \ldots, x_n)$. The marginal probability of a subset of variables $\{X_j : j \in S\}$ for an arbitrary $S \subset \{1, 2, \ldots, n\}$ is

$$\Pr\left[X_j = x_j \quad \text{for all } j \in S\right] := \sum_{x_\ell \in [k] \text{ for all } \ell \notin S} \Pr\left[X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n\right] .$$

A Bayesian network is an acyclic graph G with a node X_i corresponding to each variable X_i along with a set of directed edges E that encode a factorization of the joint distribution. Specifically, if $\mathrm{Pa}(X_i) = \{X_j : (X_j \to X_i) \in E\}$ are the parents of X_i in G then the Bayesian network represents the assertion that for all $x_1, x_2, \ldots, x_n$, the joint distribution can be factorized as follows:

$$\Pr\left[X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n\right] = \prod_{i=1}^{n} \Pr\left[X_i = x_i \ \mid \ X_j = x_j \ \forall \ X_j \in \mathrm{Pa}(X_i)\right] .$$

For example, $E = \emptyset$ corresponds to the assertion that the X_i are fully independent whereas the graph on nodes $\{X_1, X_2, X_3\}$ with directed edges $X_1 \to X_2, X_1 \to X_3$ corresponds to the assertion that X_2 and X_3 are independent conditioned on X_1. We consider the problem of evaluating how well the observed data fits a Bayesian network. The data stream of tuples in $[k]^n$ and a Bayesian network G defines an empirical distribution $\mathcal{P}_G$:

$$\mathcal{P}_G(x_1, \ldots, x_n) := \prod_{i=1}^{n} \Pr\left[X_i = x_i | X_j = x_j \ \forall \ X_j \in \mathrm{Pa}(X_i)\right] ,$$

where

$$\Pr\left[X_i = x_i \ \mid \ X_j = x_j \text{ for all } X_j \in \mathrm{Pa}(X_i)\right] = \frac{\Pr\left[X_i = x_i \wedge X_j = x_j \forall X_j \in \mathrm{Pa}(X_i)\right]}{\Pr\left[X_j = x_j, \ \forall X_j \in \mathrm{Pa}(X_i)\right]}$$

is just the fraction of tuples whose ith coordinate is x_i amongst the set of tuples whose jth coordinate is x_j for all $X_j \in \mathrm{Pa}(X_i)$. We then define the error of G to be the ℓ_p norm, for $p \in \{1, 2\}$, of the difference between the joint distribution $\mathcal{P}$ and the factorization $\mathcal{P}_G$. In particular, we exhibit:

1. A lower bound of $\Omega(kn^d)$ space for any constant pass algorithm that determines if the ℓ_p distance between $\mathcal{P}$ and $\mathcal{P}_G$ is zero.
2. A near-optimal upper bound that $(1+\epsilon)$-approximate the ℓ_p distance between $\mathcal{P}$ and $\mathcal{P}_G$.

Finding subcube heavy hitters. We study the problem of finding heavy hitters in high dimensional data streams. Formally, let us start with a one-dimensional stream of items $x_1, \ldots x_m$ where each $x_i \in \{1, 2, \ldots, n\}$. We can look at the count $c(v) = |\{i : x_i = v\}|$ or the frequency ratio $f(v) = c(v)/m$. A *heavy hitter* value v is one with $c(v) \geq \gamma m$ or equivalently $f(v) \geq \gamma$, for some constant γ. The standard *data stream model* is that we maintain data structures of size $\mathrm{polylog}(m, n)$ and determine if v is a heavy hitter with probability of success at least 3/4, that is, if $f(v) \geq \gamma$ output YES and output NO if $f(v) < \gamma/4$ for all v.[1] We note that if $\gamma/4 \leq f(v) < \gamma$, then either answer is acceptable.

Detecting heavy hitters on data streams is a fundamental problem that arises in guises such as finding elephant flows and network attacks in networking, finding hot trends in databases, finding frequent patterns in data mining, finding largest coefficients in signal analysis, and so on. Therefore, the heavy hitters problem has been studied extensively in theory, databases, networking and signal processing literature. See [49] for an early survey and [139] for a recent survey.

---

[1] The gap constant 4 can be narrowed arbitrarily and the success probability can be amplified to $1 - \delta$ as needed, and we omit these factors in the discussions.

We extend this problem to higher dimensional data streams. For this problem, we use d to denote the dimensionality of stream items. A *k-dimensional subcube* T is a subset of k distinct coordinates $\{T_1, \cdots, T_k\} \subseteq [d]$.

Our problem takes k, γ as parameters and the stream as the input and build data structures to answer:

- *Subcube Heavy Hitter*: Query(T, v), where $|T| = k$, and $v \in [n]^k$, returns an estimate if the frequency of the joint values v in coordinates T is at least γ. Specifically, output YES if the frequency is at least γ and NO if the frequency is smaller than $\gamma/4$. The required success probability *for all* k-dimensional subcubes T and $v \in [n]^k$ is at least 3/4.
- *All Subcube Heavy Hitters*: AllQuery(T) outputs all joint values v that return YES to Query(T, v). This is conditioned on the algorithm used for Query(T, v).

It is important to emphasize that the stream is presented (in a single pass or constant passes) to the algorithm before the algorithm receives any query.

The problem we address is directly related to frequent itemset mining studied in the data mining community. In fact, the frequent itemset mining is a special case of our problem where each dimension is binary ($n = 2$), and we only asks questions about joint values that are all 1. Let $\mathbf{U}_k$ denote the k-tuple of all 1. Recently, Liberty et al. showed that any constant-pass streaming algorithm answering Query$(T, \mathbf{U}_k)$ requires $\Omega(kd/\gamma \cdot \log(d/k))$ space.

We observe a simple approach using Reservoir sampling [137] solves subcube heavy hitters problems more efficiently compared to the approaches mentioned above. Our analysis shows that the space we use is within polylogarithmic factors of the lower bound shown in [108] for binary dimensions and query vector $\mathbf{U}_k$, which is a special case of our problem. Therefore, the subcube heavy hitters problem can be solved using $\tilde{O}(kd/\gamma)$ space.

We further avoid this quadratic bottleneck (i.e., when k is large, the memory becomes d^2) for finding subcube heavy hitters. We adopt the notion that there is an underlying probabilistic model behind the data, and in the spirit of the Naive Bayes model, we assume that the dimensions are nearly mutually independent given an observable latent dimension. This could be considered as a low rank factorization of the dimensions. In particular, one could formalize this assumption by bounding the total variational distance between the data's joint distribution and that derived from the Naive Bayes formula. This assumption is common in statistical data analysis and highly prevalent in machine learning. Following this modeling, we make two main contributions:

1. A two-pass, $\tilde{O}(d/\gamma)$-space streaming algorithm for answering $\mathrm{Query}(T, v)$. This improves upon the kd factor in the space complexity from sampling, without assumptions, to just d with the Naive Bayes assumption, which would make this algorithm practical for large k.
2. A fast algorithm for answering $\mathrm{AllQuery}(T)$ in $\tilde{O}((k/\gamma)^2)$ time. The naive procedure would take exponential time $\Omega((1/\gamma)^k)$ by considering the Cartesian product of the heavy hitters in each dimension. Our approach, on the other hand, uses the structure of the Naive Bayes assumption to iteratively construct the subcube heavy hitters one dimension at a time.

Our work develops the direction of model-based data stream analysis. Model-based data analysis has been effective in other areas. For example, in compressed sensing, realistic signal models that include dependencies between values and locations of the signal coefficients improve upon unconstrained cases [55]. In statistics, using tree constrained models of multidimensional data sometimes improves point and density estimation. In high dimensional distribution testing, model based approach has also been studied to overcome the curse of dimensionality [54].

CHAPTER 2

BASIC BACKGROUND AND NOTATION

In this chapter, we present basic notation and background the the rest of this thesis.

2.1 Basic Notation

We say y is a $1+\epsilon$ approximation of x if

$$(1-\epsilon)\cdot x \le y \le (1+\epsilon)\cdot x$$

which is often denoted by $y=(1\pm\epsilon)x$.

We frequently use $[n]$ to denote the set of the first n natural numbers $\{1,2,\ldots,n\}$.

Vector Notation. If x is a vector of n entries, we use x_i to denote the ith entry of x. Furthermore, the pth norm of x is

$$\|x\|_p := \sqrt[p]{\sum_{i=1}^{n} x_i^p}\ .$$

The pth frequency moment of x is simply $\|x\|_p^p$. In the case $p=0$, we define $\|x\|_0 := \left|\{i\in[n] : x_i \neq 0\}\right|$ which is the number of non-zero entries.

Tuples and joint values. We use $(x_1,x_2,\ldots,x_k)\in[n]^k$ to denote an k-tuple where each $x_i\in[n]$. We also use joint values and tuple interchangably.

Asymptotic notation. We use polylog(n) to denote $\log^c n$ for some constant c. We use $\tilde{O}$ to suppress polylog factors. For example, $O(n \log^2 n) = \tilde{O}(n)$.

We also often rely on the observation that $\log_{1+\epsilon} n = O(\epsilon^{-1} \log n)$.

Probability and statistics notation. We often use $I[A]$ to denote the indicator variable for the event A. Let X be a random variable, $\mathrm{E}[X]$ and $\mathrm{V}[X]$ denote the expectation and variance of X respectively.

2.2 Graph Theory Notation and Convention

We use the common convention notation that V and E are the sets of nodes of and edges respectively. Additionally, we let n and m be the number of nodes and edges respectively. Furthermore, we often refer to "high probability" as $1 - 1/\operatorname{poly}(m, n)$ in the corresponding problem.

The induce subgraph of a subset of nodes $S \subseteq V$ is often denoted by G_S. The set of edges of G_S is denoted by $E(G_S)$. We also often use $\deg(v)$ to denote the degree of a node v.

We use node and vertex interchangeably in this thesis.

2.3 Concentration Bounds

We shall rely on some standard concentration bounds: Markov bound, Chebyshev bound and Chernoff bound.

Theorem 1 (Markov bound). *Let X be a non-negative random variable. Then,*

$$\Pr\left[X \geq \alpha \cdot \mathrm{E}[X]\right] \leq \frac{1}{\alpha} .$$

Theorem 2 (Chebyshev bound). *Let X be a random variable. Then,*

$$\Pr\left[|X - \mathrm{E}[X]| \geq \alpha\right] \leq \frac{\mathrm{V}[X]}{\alpha^2} .$$

Finally, we will frequently use the following version of Chernoff bound. We say that X_i are *negatively correlated* if for all X_i and $S \subset \{1, 2, \ldots, n\}$ where $i \notin S$, we have

$$\Pr\left[X_i = 1 \;\mid\; X_j = 1 \text{ for all } j \in S\right] \leq \Pr\left[X_i = 1\right] \enspace .$$

But first, let $X_1, \ldots, X_n$ be random binary variables. We will mostly appeal to the following Chernoff bound.

Theorem 3 (Chernoff bound)**.** *Let $X_1, X_2, \ldots, X_n$ be random binary variables that are either mutually independent or negatively correlated and let $\mu = \mathrm{E}\left[\sum_{i=1}^{n} X_i\right]$. Then, for any $\epsilon > 0$,*

$$\Pr\left[\left|\sum_{i=1}^{n} X_i - \mu\right| \geq \epsilon\mu\right] \leq \exp\left(\frac{-\min\{\epsilon^2, \epsilon\} \cdot \mu}{3}\right) \enspace .$$

CHAPTER 3

FAST ℓ_p SAMPLING AND ITS APPLICATION TO FINDING THE APPROXIMATE DENSEST SUBGRAPH

3.1 Introduction

In this chapter, we develop a fast ℓ_p sampling algorithm. We then apply this algorithm to the problem of finding the densest subgraph in dynamic graph streams.

ℓ_p sampling. In the *turnstile data stream model*, the stream is a sequence of m additive updates on entries of an underlying vector x of length n. Specifically, each update has the form

$$x_i \leftarrow x_i + \Delta \ .$$

An ℓ_p sampler (see [9, 85, 120]) is a data structure that takes one pass over a turnstile stream and with high probability returns a pair of an index j and an estimate y_j where $y_j = (1 \pm \epsilon)x_j$ and the probability that j is equal to i is proportional to x_i^p up to a $1 + \epsilon$ factor. In particular,

$$\Pr\left[j = i\right] = (1 \pm \epsilon) \cdot \frac{x_i^p}{\|x\|_p^p} \ .$$

If $p = 0$, the ℓ_p sampling data structure returns a uniformly random index j where x_j is non-zero. We have the promise that the ℓ_p sampler succeeds with probability at least $1 - n^{-c}$ where c is a constant. The constant c hides in the space use by the ℓ_p sampler. The update time of a sampler is the time to process an update to the vector x.

Lemma 4 ([9,85,120]). *There exists an ℓ_p sampler that uses $\tilde{O}(\epsilon^{-\max\{1,\ p\}})$ space for $0 < p \leq 2$. For $p = 0$, there exists an ℓ_0 sampler that uses $\tilde{O}(1)$ space. In both cases, the update time is $\tilde{O}(1)$.*

Densest subgraph problem. In the densest subgraph problem, the goal is to identify the subgraph that has the maximum (weighted) average degree. Somewhat surprisingly, there are polynomial time algorithms to find the densest subgraph [40, 64, 72, 95] and more efficient approximation algorithms also exist [40].

Our goal is to find an approximate densest subgraph in the dynamic graph stream model. Two main ingredients of our algorithm are:

- For unweighted graphs, we will show that a subsampled graph formed by $\tilde{O}(\epsilon^{-2}n)$ random edges preserves the densest density up to a $1 + \epsilon$ factor. For weighted graphs, we can sample the edges with replacement based on their weights.
- The above observation can be translated into a natural streaming algorithm that uses $\tilde{O}(\epsilon^{-2}n)$ space, i.e., we can sample the edges using ℓ_0 or ℓ_1 sampling. A major drawback of this implementation is that the update time is $\Omega(n)$. We show that $\tilde{O}(1)$ update time is possible by designing a fast ℓ_p sampling algorithm.

3.2 Fast ℓ_p Sampling Algorithm

Suppose we want to draw s independent ℓ_p samples. The naive implementation that maintains s different ℓ_p samplers in parallel would require $\Omega(s)$ update time. This section focuses on providing a faster update time. We prove the following result.

Theorem 5. *In the turnstile model, we have the following algorithms.*

- *For $p \in (0, 2]$, there exists a single-pass algorithm that, with high probability, outputs s independent ℓ_p samples using $\tilde{O}(s \cdot \epsilon^{-\max\{1,\ p\}})$ space and $\tilde{O}(1)$ update time.*
- *For $p = 0$, there exists a single-pass algorithm that, with high probability, outputs s independent ℓ_0 samples using $\tilde{O}(s)$ space and $\tilde{O}(1)$ update time.*

The theorem above implies $\operatorname{polylog} n$ update time regardless of s. This is most significant when $s = o(n)$ and $s = \omega(\operatorname{polylog} n)$.

Approach. We hash the coordinates of x into w groups and for each of these groups we maintain a small number of local ℓ_p samplers restricting to the corresponding coordinates. To draw an ℓ_p sample, we randomly pick a group with probability proportional to its mass contribution to the pth frequency moment $\|x\|_p^p$ and draw an ℓ_p sample from that group using a local ℓ_p sampler. The main challenge is ensuring that each group's contribution is small so that we only need to maintain a small number of samplers in each group. To do this, we separate the heavy coordinates into one group using the Heavy-Hitters algorithm.

Algorithm for $p \in (0, 2]$. We rely on the following Heavy-Hitters result (see [85], Lemma 1 and Section 4.4) .

Lemma 6 ([85])**.** *For $p \in (0, 2]$, there exists a single-pass, $\tilde{O}(\epsilon^{-p}\phi^{-1})$-space and $\tilde{O}(1)$-update time algorithm that with high probability returns a subset of indices $A \subseteq [n]$ and the set of the corresponding estimates $B = \{y_i : y_i = (1 \pm \epsilon)x_i$ and $i \in A\}$ such that: If $x_i^p \geq \phi \cdot \|x\|_p^p$, then i is in A and $x_i^p < \phi/8 \cdot \|x\|_p^p$, then i is not in A.*

We consider a set of pairwise independent hash functions

$$h_i : [n] \to [w], \text{ for } i = 1, 2, \ldots, d \ ,$$

where $d = c \cdot \log n$ and $w = c \cdot s$ for some sufficiently large constant c. We define a group $A_{i,j}$ as the set of indices that are hashed to j by the hash function h_i. Specifically,

$$A_{i,j} := \left\{ k \in [n] : h_i(k) = j \right\} \ .$$

Let $a^{(i,j)}$ be the vector that have the same entries of x except $a^{(i,j)}_k = 0$ if $h_i(k) \neq j$. Specifically,

$$a^{(i,j)}_k := \begin{cases} x_k & \text{, if } h_i(k) = j \\ 0 & \text{, otherwise.} \end{cases}$$

Finally, the set of heavy coordinates is defined as

$$H := \left\{ k \in [n] : x^p_k \geq \frac{\|x\|^p_p}{s} \right\} \ .$$

We construct a superset $\mathcal{H}$ of H by running the Heavy-Hitters algorithm with $\phi = s^{-1}$. Moreover, for each $k \in \mathcal{H}$, the Heavy-Hitter algorithm also returns an estimate

$$y^p_k = (1 \pm \epsilon) x^p_k \ .$$

Furthermore, we maintain the followings.

During the stream, maintain:

1. $r = (1 \pm \epsilon) \|x\|^p_p$.
2. $\alpha^{(i,j)} = (1 \pm \epsilon) \left\|a^{(i,j)}\right\|^p_p$ for all i and j .
3. A heavy hitter data structure with $\phi = 1/s$.
4. $O(\log n)$ different ℓ_p samplers for each $a^{(i,j)}$.

One can use a frequency moment approximation algorithm (such as [79]) that uses $\tilde{O}(\epsilon^{-2})$ space and $\tilde{O}(1)$ update time to maintain r and each $\alpha^{(i,j)}$. We observe that a stream update to a coordinate k involves the following steps.

1. The update time for the data structure maintaining the estimate r of $\|x\|_p^p$ and the heavy hitters data structure is $\tilde{O}(1)$.
2. We need to compute $h_i(k)$ for each $i = 1, 2, \ldots, d$. For each of d groups $A_{i,j}$ which the index k is hashed to, the algorithm needs to update the data structure maintaining the estimate $\alpha^{(i,j)}$ of $\left\|a^{(i,j)}\right\|_p^p$ and $O(\log n)$ data structures of the ℓ_p samplers for the corresponding vectors $a^{(i,j)}$.

Therefore, the total update time is $\tilde{O}(1)$. At the end of the stream, we compute the pth norm of the heavy entries

$$\beta := \sum_{k \in \mathcal{H}} y_k^p \ .$$

We define

$$G := \left\{ (i,j) : \alpha^{(i,j)} < \frac{10r}{s} \right\}$$

and compute

$$\alpha := \sum_{(i,j) \in G} \alpha^{(i,j)} \ .$$

We say group $A_{i,j}$ is *good* if (i,j) is in G. Furthermore, we also consider $\mathcal{H}$ as a good group. The number of good groups that an index k belongs to is denoted by

$$g(k) := \left| \left\{ (i,j) \in G : k \in A_{i,j} \right\} \right| + I[k \in \mathcal{H}] \ .$$

We use $g(k)$ in the rejection probability to avoid bias toward the coordinates that appear in many good groups. Repeat the following trial until we get s independent random ℓ_p samples.

1. Toss a fair coin.
2. If head, then randomly pick an index $k \in \mathcal{H}$ where $\Pr\left[k = u\right] = \frac{y_u^p}{\beta}$.
   (a) Reject the current trial with probability $1 - \frac{\beta}{2dr \cdot g(k)}$.
   (b) Otherwise, add k to the sample set.
3. If tail, then randomly pick (i, j) in G where $\Pr\left[(i, j) = (u, v)\right] = \frac{\alpha^{(u,v)}}{\alpha}$.
   (a) Reject the current trial with probability $1 - \frac{\alpha}{2dr}$.
   (b) Otherwise, use the next ℓ_p sampler for $a^{(i,j)}$ to retrieve (k, y_k) .
   (c) Reject the current trial with probability $1 - \frac{1}{g(k)}$.
   (d) Otherwise, add k to the sample set.

It is important to note that the rejection probability is valid. Since each coordinate is in at most d good groups $A_{i,j}$, for sufficiently small ϵ, we have

$$\alpha \leq (1 + \epsilon) \sum_{(i,j) \in G} \left\|a^{(i,j)}\right\|_p^p \leq (1 + \epsilon) d \cdot \|x\|_p^p < 2dr \ .$$

It is also obvious that

$$\beta \leq (1 + \epsilon) \sum_{k \in \mathcal{H}} x_k^p \leq (1 + \epsilon) \|x\|_p^p < 2dr \ .$$

Next, we need to show that each index is sampled with the desired probability. The first step is to show that $g(k) > 0$ for all indices k.

Lemma 7. *For all indices $k \notin \mathcal{H}$, we must have that k belongs to at least one good group with probability at least $1 - n^{-c+1}$.*

Proof. Observe that if k that is not in $\mathcal{H}$, then $x_k^p \leq \|x\|_p^p / s$. Let us fix i and suppose $h_i(k) = j$. By pairwise independence,

$$\mathrm{E}\left[\left\|a^{(i,j)}\right\|_p^p\right] \leq x_k^p + \sum_{z \neq k} \frac{x_z^p}{w} \leq \frac{(1+c)\left\|x\right\|_p^p}{cs} \leq \frac{2\left\|x\right\|_p^p}{s} \ .$$

Applying Markov bound,

$$\Pr\left[\left\|a^{(i,j)}\right\|_p^p > \frac{8\left\|x\right\|_p^p}{s}\right] \leq \frac{1}{4} \ .$$

Hence,

$$\begin{aligned} \Pr\left[A_{i,j} \text{ is good}\right] &\geq \Pr\left[(1+\epsilon)\left\|a^{(i,j)}\right\|_p^p < \frac{10(1-\epsilon)\left\|x\right\|_p^p}{s}\right] \\ &\geq \Pr\left[\left\|a^{(i,j)}\right\|_p^p < \frac{8\left\|x\right\|_p^p}{s}\right] \geq \frac{3}{4} \end{aligned}$$

for sufficiently small ϵ. Thus, the probability that there is no good group for k is at most $4^{-d} = 4^{-c\log n} \leq n^{-c}$. The lemma follows by taking the union bound over all k in $[n]$. □

Let $S_p(k)$ denote the event of adding k to the sample set. Then, the probability of successfully retrieving a sample is

$$S_p(\text{success}) = \bigcup_{k \in [n]} S_p(k) \ .$$

We first lower bound the probability of successfully retrieving a sample.

$$\Pr\left[S_p(\text{success})\right] = \sum_{k\in\mathcal{H}} \Pr\left[S_p(k) \;\mid\; \mathsf{head}\right] \Pr\left[\mathsf{head}\right] + \sum_{\substack{(i,j)\in G\\ k\in A_{i,j}}} \Pr\left[S_p(k) \;\mid\; \mathsf{tail}\right] \Pr\left[\mathsf{tail}\right] \ .$$

The first summation can be expressed as

$$\begin{aligned}\sum_{k\in\mathcal{H}} \Pr\left[S_p(k) \;\mid\; \mathsf{head}\right] \Pr\left[\mathsf{head}\right] &= \frac{1}{2}\sum_{k\in\mathcal{H}} \frac{y_k^p}{\beta}\cdot\frac{\beta}{2dr\cdot g(k)}\\ &= \frac{1}{4}\sum_{k\in\mathcal{H}} \frac{y_k^p}{dr\cdot g(k)} \ .\end{aligned}$$

Next, we simplify the second summation

$$\begin{aligned}\sum_{\substack{(i,j)\in G\\ k\in A_{i,j}}} \Pr\left[S_p(k) \;\mid\; \mathsf{tail}\right] \Pr\left[\mathsf{tail}\right] &= \frac{1}{2}\sum_{\substack{(i,j)\in G\\ k\in A_{i,j}}} \frac{\alpha^{(i,j)}}{\alpha}\cdot\frac{\alpha}{2dr}\cdot\frac{y_k^p}{\alpha^{(i,j)}}\cdot\frac{1}{g(k)}\\ &= \frac{1}{4}\sum_{\substack{(i,j)\in G\\ k\in A_{i,j}}} \frac{y_k^p}{dr\cdot g(k)} \ .\end{aligned}$$

First note that

$$\sum_{\substack{(i,j)\in G\\ k\in A_{i,j}}} \frac{y_k^p}{g(k)} = (1\pm\epsilon)\left\|x\right\|_p^p \ .$$

We then observe that $r = (1\pm\epsilon)\left\|x\right\|_p^p$ and $\alpha^{(i,j)} = (1\pm\epsilon)\left\|a^{(i,j)}\right\|_p^p$ to yield

$$\Pr\left[S_p(\text{success})\right] = \frac{1\pm 2\epsilon}{4d} = \Theta\left(\frac{1}{\log n}\right) \ .$$

We also have

$$\Pr\left[S_p(k) \text{ and } S_p(\text{success})\right] = \frac{1}{4d}\cdot\frac{y_k^p}{r} \ .$$

Therefore,

$$\Pr\left[S_p(x_k) \;\mid\; S_p(\text{success})\right] = \frac{(1\pm 4\epsilon)x_k^p}{\left\|x\right\|_p^p} \ .$$

Thus, by re-parameterizing ϵ, we have proved the following.

Lemma 8. *The probability of sampling k in a successful trial is*

$$\Pr\left[S_p(k) \ \mid \ S_p(\text{success})\right] = \frac{(1 \pm \epsilon)x_k^p}{\|x\|_p^p} \ .$$

Furthermore, each trial succeeds with probability $\Omega(1/\log n)$.

Finally, we show that it suffices to maintain $O(\log n)$ ℓ_p samplers on each vector $a^{(i,j)}$.

Lemma 9. *With high probability, repeating $O(s \log^2 n)$ trials, we obtain at least s independent ℓ_p samples and we need to draw $O(\log n)$ different ℓ_p samples from each group.*

Proof. As shown above, $\Pr\left[S_p(\text{success})\right] = \Omega(1/\log n)$. Thus, the first claim follows immediately from Chernoff bound. On the other hand, we draw a sample from $a^{(i,j)}$ if and only if (i,j) is in G which happens if and only if $\alpha^{(i,j)} \leq 10r/s$. Therefore, for each trial,

$$\Pr\left[\text{draw an } \ell_p \text{ sample from a good group } A_{i,j}\right] = \frac{\alpha^{(i,j)}}{2dr} = O\left(\frac{1}{s \log n}\right) \ .$$

For appropriate choice of constants, appealing to Chernoff bound again, the probability that a good group needs more than $O(\log n)$ ℓ_p samples is less than $1/\operatorname{poly}(n)$. Finally, appealing to the union bound over $O(s \log n)$ good groups, we conclude the second claim. □

Algorithm for $p = 0$. The case $p = 0$ is simpler since we do not need to separate the heavy coordinates. We can assume $s = o(\|x\|_0)$; otherwise, we can reconstruct x

via standard sparse recovery algorithms (e.g., see [69]). We again consider a set of pairwise independent hash functions

$$h_i : [n] \to [w], \text{ for } i = 1, 2, \ldots, d \ ,$$

where $d = c \cdot \log n$ and $w = c \cdot s$ for some sufficiently large constant c. Following the same approach for the case $p \in (0, 2]$, we define a group $A_{i,j}$ as the set of indices that are hashed to j by the hash function h_i. Specifically,

$$A_{i,j} := \Big\{ k \in [n] : h_i(k) = j \Big\} \ .$$

Similarly, we define the vectors $a^{(i,j)}$ that the same entries of x except $a_k^{(i,j)} = 0$ if $h_i(k) \neq j$.

$$a_k^{(i,j)} := \begin{cases} x_k & \text{, if } h_i(k) = j \\ 0 & \text{, otherwise.} \end{cases}$$

We first assume that all insertions and deletions are "atomic" such that x_i is either 0 or 1 at all points. This is true for the characteristic vector of the set of edges E in a dynamic graph stream. This means that we can maintain $\alpha^{(i,j)} = \|a^{(i,j)}\|_0$ exactly. The data structure during the stream is as follows.

During the stream, maintain the following:

1. $\alpha^{(i,j)} = \|a^{(i,j)}\|_0$.
2. $r = \|x\|_0$.
3. $O(\log n)$ different ℓ_0 samplers for each $a^{(i,j)}$.

Again, it is easy to see that the update time is $\tilde{O}(1)$. We say that a group $A_{i,j}$ is good if

$$\left\|a^{(i,j)}\right\|_0 \leq \frac{8r}{w} \ .$$

We again let $g(k)$ denote the number of good groups the index k belongs to and show that $g(k) > 0$ for all indices k.

Lemma 10. *For all indices k, with high probability, k belongs to a good group.*

Proof. Fix a hash function h_i and let $h(k) = j$. In expectation,

$$\mathrm{E}\left[\alpha^{(i,j)}\right] \leq 1 + \frac{\|x\|_0}{w} \leq \frac{cs + \|x\|_0}{w} \leq \frac{2\|x\|_0}{w} \ .$$

Appealing to Markov inequality, for sufficiently small ϵ, we deduce that

$$\Pr\left[A_{i,j} \text{ is good}\right] = \Pr\left[\alpha^{(i,j)} > \frac{8r}{w}\right] \geq \frac{3}{4} \ .$$

The second inequality follows from the assumption that $s = o(\|x\|_0)$. Therefore, the probability that k does not belong to a good group is at most $4^{-d} \leq n^{-10}$ since $d = c \log n$ for some large constant c. By taking the union bound over all k, we deduce the lemma. □

In post-processing, let $g(k)$ be the number of good group the index k belongs to. Let

$$G := \left\{(i,j) : A_{i,j} \text{ is good}\right\}$$

and

$$\alpha := \sum_{(i,j) \in G} \alpha^{(i,j)} \ .$$

We repeat the following trial until s samples are retrieved.

1. Pick a random (i,j) with probability $\dfrac{\alpha^{(i,j)}}{\alpha}$.
2. Use the next ℓ_0 sampler for $a^{(i,j)}$, retrieve an ℓ_0 sample k of $a^{(i,j)}$.
3. Reject the current trial with probability $1 - 1/(2 \cdot g(k))$.
4. Otherwise, add k to the sample set.

Similar to the case $p \in (0, 2]$, we want to show that

Lemma 11. *The probability of sampling k in a successful trial is*

$$\Pr\left[S_0(k) \;\mid\; S_0(\text{success})\right] = \frac{1}{\|x\|_0} \; .$$

Furthermore, each trial succeeds with probability $\Omega(1/\log n)$.

Proof. The success probability of retrieving a sample is

$$\Pr\left[S_0(\text{success})\right] = \sum_{(i,j)\in G} \sum_{\substack{k\in A_{i,j} \\ x_k \neq 0}} \frac{\alpha^{(i,j)}}{\alpha} \cdot \frac{1}{\|a^{(i,j)}\|_0} \cdot \frac{1}{2g(k)} = \frac{\|x\|_0}{2\alpha} \geq \frac{1}{2c\log n} \; .$$

The last inequality follows from the fact that each index belongs to at most $d \leq c\log n$ good groups. It is easy to see that the probability of sampling k is

$$\Pr\left[S_0(k) \;\mid\; S_0(\text{success})\right] = \frac{1}{2\alpha} \cdot \frac{2\alpha}{\|x\|_0} = \frac{1}{\|x\|_0}$$

as required. ☐

We now show that we need to draw $O(\log n)$ samples from each good groups.

Lemma 12. *With high probability, repeating $O(s\log^2 n)$ trials gives us s independent ℓ_0 samples and we need to draw $O(\log n)$ different ℓ_0 samples from each group.*

Proof. Appealing to Chernoff bound, with probability at least $1 - n^{-10}$, we need to perform $O(s\log^2 n)$ trials to attain s samples with high probability. For each trial, we draw a sample from a good group $A_{i,j}$ with probability

$$\Pr\left[\text{draw an } \ell_0 \text{ sample from a good group } A_{i,j}\right] = \frac{\alpha^{(i,j)}}{2\alpha} \leq \frac{8r}{wd\|x\|_0} \leq \frac{8}{c^2 s\log n} \; .$$

The second inequality follows from $\alpha \leq d\|x\|_0$ since each entry belongs to at most d good groups. Therefore, we draw $O(s\log n)$ samples from each good group $A_{i,j}$ with probability at least $1 - n^{-10}$. Taking the union bound over $O(s \log n)$ good groups concludes the claim.

$\square$

Remark. In the case that the updates do not guarantee $x_i \in \{0, 1\}$ at all points, then we can use F_0 approximation algorithm (e.g., [50]) to find $\alpha^{(i,j)} = (1 + \epsilon)\|a^{(i,j)}\|_0$ and $r = (1 \pm \epsilon)\|x\|_0$, then the probability of sampling an index k entails a $1 \pm \epsilon$ factor. Specifically,

$$\Pr\left[S_0(k) \ \mid \ S_0(\text{success})\right] = \frac{1 \pm \epsilon}{\|x\|_0} \ .$$

3.3 Application: Finding Approximate Densest Subgraph

Problem description. We consider the densest subgraph problem in the dynamic graph stream model. Let G_U be the induced subgraph of graph $G = (V, E)$ on nodes U. Then the *density* of G_U is defined as

$$d(G_U) := \frac{|E(G_U)|}{|U|} \ ,$$

where $E(G_U)$ is the set of edges in the induced subgraph. In weighted graphs, the density of G_U is

$$d(G_U) := \frac{w\left(E(G_U)\right)}{|U|} \ ,$$

where $w\left(E(G_U)\right)$ is the total weight of the edges in $E(G_U)$.

We define the *maximum density* as

$$d^* := \max_{U \subseteq V} d(G_U) \ .$$

and say that the corresponding subgraph is the *densest subgraph*. The densest subgraph can be found in polynomial time [40, 64, 72, 95] and more efficient approximation algorithms have been designed [40]. Finding dense subgraphs is an important primitive when analyzing massive graphs; applications include community detection in social networks and identifying link spam on the web, in addition to applications on financial and biological data. See [105] for a survey of applications and existing algorithms for the problem.

Our contributions. We present a single-pass algorithm that returns a $(1-\epsilon)$ approximation[1] with high probability. For a graph on n nodes, the algorithm uses the following resources:

- *Space:* $O(\epsilon^{-2} n \operatorname{polylog} n)$. The space used by our algorithm matches the lower bound of Bahmani et al. [20] up to a poly-logarithmic factor for constant ϵ.
- *Per-update time:* $\operatorname{polylog}(n)$. We note that this is the worst-case update time rather than amortized over all the edge insertions and deletions.
- *Post-processing time:* $\operatorname{poly}(n)$. This will follow by using any exact algorithm for densest subgraph [40, 64, 72] on the subgraph generated by our algorithm.

Related work. The most relevant previous results for the problem were established recently by Bhattacharya et al. [25]. They presented two algorithms that use similar space to our algorithm and process updates in $\operatorname{polylog}(n)$ amortized time. The first algorithm returns a $(1/2-\epsilon)$ approximation of the maximum density of the final graph while the second (the more technically challenging result) outputs a $(1/4-\epsilon)$ approximation of the current maximum density after every update while still using only $\operatorname{polylog}(n)$ time per-update. Our algorithm improves the approximation factor to $(1-\epsilon)$ while keeping the same space and update time. It is possible to modify our algorithm to output a $(1-\epsilon)$ approximation to the current maximum density after each update but the simplest approach would require the post-processing step to be run after every edge update and this would not be efficient.

---

[1] We adopt the convention that for maximization problems, an α approximation (where $\alpha \le 1$) is a solution that is at least $\alpha\,\mathrm{OPT}$.

Bhattacharya et al. were one of the first to combine the space restriction of graph streaming with the fast update and query time requirements of fully-dynamic algorithms from the dynamic graph algorithms community. Epasto, Lattanzi, and Sozio [60] present a fully-dynamic algorithm that returns a $(1/2 - \epsilon)$ approximation of the current maximum density. Other relevant work includes papers by Bahmani, Kumar, and Vassilvitskii [20] and Bahmani, Goel, and Munagala [19]. The focus of these papers is on designing algorithms in the MapReduce model but the resulting algorithms can also be implemented in the data stream model if we allow multiple passes over the data.

Our approach. For unweighted graphs, our algorithm requires maintaining $\tilde{O}(\epsilon^{-2}n)$ random edges. For weighted graphs, our algorithm samples with replacement $\tilde{O}(\epsilon^{-2}n)$ edges based on their weights. We then need to argue, via Chernoff bound, that the subsampled graph approximately preserves the densest subgraph.

In both cases, a naive implementation requires $\tilde{O}(\epsilon^{-2}n)$ update time. To reduce the update time to $\tilde{O}(1)$, we use fast ℓ_p-sampling algorithm that we developed in the previous section. We first consider the unweighted case.

Subsampling approximately preserves maximum density. In this section, we consider properties of a random subgraph of an unweighted input graph G. Specifically, let G' be the graph formed by sampling each edge in G independently with probability p where

$$p = c\epsilon^{-2} \log n \cdot \frac{n}{m}$$

for some sufficiently large constant $c > 0$ and $0 < \epsilon < 1/2$. We may assume that m is sufficiently large such that $p < 1$ because otherwise we can reconstruct the entire graph in the allotted space using standard results from the sparse recovery literature [69].

We will prove that, with high probability, the maximum density of G can be estimated up to factor $(1-\epsilon)$ given G'. While it is easy to analyze how the density of a specific subgraph changes after the edge sampling, we will need to consider all 2^n possible induced subgraphs and prove properties of the subsampling for all of them.

The next lemma shows that $d(G'_U)$ is roughly proportional to $d(G_U)$ if $d(G_U)$ is "large" whereas if $d(G_U)$ is "small" then $d(G'_U)$ will also be relatively small.

Lemma 13. *Let U be an arbitrary set of k nodes. Then,*

$$\Pr\left[d(G'_U) \geq pd^*/10\right] \leq n^{-10k} \qquad \text{if } d(G_U) \leq d^*/60$$

$$\Pr\left[|d(G'_U) - pd(G_U)| \geq \epsilon pd(G_U)\right] \leq 2n^{-10k} \qquad \text{if } d(G_U) > d^*/60 \; .$$

Proof. We start by considering the density of the entire graph $d(G) = m/n$ and therefore conclude that the maximum density, d^*, is at least m/n. Hence, $p \geq (c\epsilon^{-2}\log n)/d^*$.

Let X be the number of edges in G'_U and note that $\mathrm{E}[X] = pkd(G_U)$. First assume $d(G_U) \leq d^*/60$. Then, by an application of the Chernoff Bound (e.g., [119, Theorem 4.4]), we observe that

$$\Pr\left[d(G'_U) \geq pd^*/10\right] = \Pr\left[X \geq pkd^*/10\right] \leq 2^{-pkd^*/10} < 2^{-(ck\log n)/10}$$

and this is at most n^{-10k} for sufficiently large constant c.

Next assume $d(G_U) > d^*/60$. Hence, by an application of an alternative form of the Chernoff Bound (e.g., [119, Theorem 4.4 and 4.5]), we observe that

$$\begin{aligned}\Pr\left[|d(G'_U) - pd(G_U)| \geq \epsilon pd(G_U)\right] &= \Pr\left[|X - pkd(G_U)| \geq \epsilon pkd(G_U)\right] \\ &\leq 2\exp(-\epsilon^2 pkd(G_U)/3) \\ &\leq 2\exp(-\epsilon^2 pkd^*/180) \\ &\leq 2\exp(-ck(\log n)/180) \; .\end{aligned}$$

and this is at most $2n^{-10k}$ for sufficiently large constant c. □

Corollary 14. *With high probability, for all $U \subseteq V$:*

$$d(G'_U) \geq (1-\epsilon)pd^* \quad \Longrightarrow \quad d(G_U) \geq \frac{1-\epsilon}{1+\epsilon} \cdot d^* \ .$$

Proof. There are $\binom{n}{k} \leq n^k$ subsets of V that have size k. Hence, by appealing to Lemma 13 and the union bound, with probability at least $1 - 2n^{-9k}$, the following two equations hold,

$$\begin{aligned} d(G'_U) \geq pd^*/10 \quad &\Longrightarrow \quad d(G_U) > d^*/60 \\ d(G_U) > d^*/60 \quad &\Longrightarrow \quad d(G_U) \geq \frac{d(G'_U)}{p(1+\epsilon)} \end{aligned}$$

for all $U \subseteq V$ such that $|U| = k$. Since $(1-\epsilon)pd^* \geq pd^*/10$, together these two equations imply

$$d(G'_U) \geq (1-\epsilon)pd^* \quad \Longrightarrow \quad d(G_U) \geq \frac{d(G'_U)}{p(1+\epsilon)} \geq \frac{1-\epsilon}{1+\epsilon} \cdot d^*$$

for all sets U of size k. Taking the union bound over all values of k establishes the corollary. □

We next show that the densest subgraph in G' corresponds to a subgraph in G that is almost as dense as the densest subgraph in G.

Theorem 15. *Let $U' = \arg\max_U d(G'_U)$. Then with high probability,*

$$\frac{1-\epsilon}{1+\epsilon} \cdot d^* \leq d(G_{U'}) \leq d^* \ .$$

Proof. Let $U^* = \arg\max_U d(G_U)$. By appealing to Lemma 13, we know that $d(G'_{U^*}) \geq (1-\epsilon)pd^*$ with high probability. Therefore

$$d(G'_{U'}) \geq d(G'_{U^*}) \geq (1-\epsilon)pd^* \ ,$$

and the result follows by appealing to Corollary 14. □

Implementation in dynamic graph streams. First, we observe a dynamic graph stream is just a turnstile stream on the characteristic vector of the set of edges E. Therefore, we can use the ℓ_0 sampler to sample a random edge.

Sampling each edge independently with probability p can be simulated via a two-step procedure below.

- *Fix the number X of edges to sample:* Let $X \sim \mathbf{Bin}(g,p)$ where g is the number of edges in the relevant group.
- *Fix which X edges to sample:* We then randomly pick X edges without replacement.

The following lemma follows immediately from Chernoff bound.

Lemma 16. *With high probability, $X = O(\epsilon^{-2} n \log n)$.*

If we draw $O(\epsilon^{-2} n \log^2 n)$ independent ℓ_0 samples from the set of edges E, with high probability, we obtain $\Omega(\epsilon^{-2} n \log n)$ distinct random edges. But we know that we could do so in $\tilde{O}(\epsilon^{-2} n)$ space and $\tilde{O}(1)$ update time according to Theorem 5. We summarize our result as the following theorem.

Theorem 17. *There exists a single-pass algorithm that finds a $1-\epsilon$ approximation of the densest subgraph in the dynamic graph stream model which uses $\tilde{O}(\epsilon^{-2} n)$ space and $\tilde{O}(1)$ update time.*

Weighted densest subgraph. We show that the weighted densest subgraph problem can also be solved in the dynamic graph stream model using ℓ_1 sampling. We sample $t = c\epsilon^{-2} n \log n$ random edges with replacement based on their weights using ℓ_1 sampling. In particular, we sample an edge e with probability:

$$\Pr\left[\text{sample edge } e\right] = (1 \pm \epsilon)\frac{w(e)}{W} \ .$$

where W is the total weight of the edges in the graph. Let the multi-set of the sampled edges be $S = \{s_1, s_2, \ldots, s_t\}$, and from S we construct a multi-graph G'. The weight of an edge in the subsampled graph G' is the number of times it appears in S. Another interpretation of G' is a multi-graph with edges in S. We will show that the densities in the newly constructed graph, in expectation, is scaled by a factor $p = t/W$. We observe that $p = (c\epsilon^{-2} n \log n)/W \geq (c\epsilon^{-2} \log n)/d^*$ since $d^* \geq W/n$. The next lemma shows that $d(G'_U)$ is roughly $p \cdot d(G_U)$ if $d(G_U)$ is "large" whereas if $d(G_U)$ is "small" then $d(G'_U)$ will also be relatively small.

Lemma 18. *Let U be an arbitrary set of k nodes. Then,*

$$\begin{aligned} \Pr\left[d(G'_U) \geq pd^*/10\right] &\leq n^{-10k} && \textit{if } d(G_U) \leq d^*/100 \\ \Pr\left[\left|d(G'_U) - pd(G_U)\right| \geq \epsilon pd(G_U)\right] &\leq 2n^{-10k} && \textit{if } d(G_U) > d^*/100 \ . \end{aligned}$$

Proof. We first observe that if $d(G_U) < d^*/100$, then for sufficiently small ϵ,

$$\mathrm{E}\left[\sum_{i\in[t]} I[s_i \in E(G'_U)]\right] = (1 \pm \epsilon)\frac{t \cdot w(E(G_U))}{W} = (1 \pm \epsilon)\frac{t|U|d(G_U)}{W} \leq \frac{t|U|d^*}{60W} \ .$$

We observe that

$$d(G'_U) = \frac{\sum_{i\in[t]} I[s_i \in E(G'_U)]}{|U|} \ .$$

Then, we appeal to the Chernoff bound

$$\begin{aligned}\Pr\left[d(G'_U) \geq p \cdot \frac{d^*}{10}\right] &= \Pr\left[\sum_{i\in[t]} I[s_i \in E(G'_U)] \geq \frac{t|U|d^*}{10W}\right]\\ &\leq \exp\left(\frac{-t|U|d^*}{10W}\right) \leq \exp\left(\frac{-t|U|}{10n}\right) < n^{-10|U|}\ .\end{aligned}$$

The second inequality follows from $d^* \geq W/n$. These steps hold for sufficiently large c and sufficiently small ϵ.

Next, we consider the case $d(G_U) \geq d^*/100$. We again appeal to the Chernoff bound

$$\begin{aligned}&\Pr\left[\left|d(G'_U) - p \cdot d(G_U)\right| \geq \epsilon p \cdot d(G_U)\right]\\ =&\Pr\left[\left|\sum_{i\in[t]} I[s_i \in E(U)] - \frac{t|U|}{W} \cdot d(G_U)\right| \geq \frac{\epsilon t|U|}{W} \cdot d(G_U)\right]\\ \leq&\exp\left(\frac{-\epsilon^2 t|U|d(G_U)}{3W}\right)\\ \leq&\exp\left(-\frac{c|U|\log n}{180}\right)\\ \leq&\ n^{-10|U|}\ .\end{aligned}$$

We use the assumption that $d(G_U) \geq d^*/100 \geq 1/100 \cdot W/n$ to get the inequality in the third step. By taking a union bound over at most $n^{|U|}$ subgraphs of size $|U|$, the above guarantees holds for all of those subgraphs with high probability. □

Taking another union bound over all n possible values of $|U|$, we have the following corollary.

Corollary 19. *With high probability, for all subgraphs U,*

$$d(G'_U) \geq (1-\epsilon)pd^* \quad \Longrightarrow \quad d(G_U) \geq \frac{1-\epsilon}{1+\epsilon} \cdot d^*\ .$$

Thus, if we find the densest weighted subgraph of the described graph G', we have the following.

Theorem 20. *Let* $U' = \arg\max_U d(G'_U)$*. Then with high probability,*

$$\frac{1-\epsilon}{1+\epsilon} \cdot d^* \leq d(G_{U'}) \leq d^* \ .$$

Proof. Let $U^* = \arg\max_U d(G_U)$ be the densest subgraph in G. By appealing to Lemma 18, we know that $d(G'_{U^*}) \geq (1-\epsilon)pd^*$ with high probability. Therefore

$$d(G'_{U'}) \geq d(G'_{U^*}) \geq (1-\epsilon)pd^* \ ,$$

and the result follows by appealing to Corollary 19. □

Finally, recall that we can sample $t = c\epsilon^{-2}n \log n$ random edges with replacement based on their weights using ℓ_1 sampling given by Theorem 5. We therefore proved the following.

Theorem 21. *There exists a single-pass algorithm that finds a* $1-\epsilon$ *approximation of the weighted densest subgraph in the dynamic graph stream model which uses* $\tilde{O}(\epsilon^{-2}n)$ *space and* $\tilde{O}(1)$ *update time.*

CHAPTER 4

TRIANGLE COUNTING IN THE ADJACENCY LIST MODEL

4.1 Introduction and Related Work

Estimating the number of triangles in a graph is a canonical problem in the data stream model of computation. The problem was first considered by Bar-Yossef et al. [21] nearly fifteen years ago and a significant body of work has since been devoted to designing more efficient and ingenious algorithms for the problem in both the single-pass [5, 7, 21, 32, 33, 81, 84, 103, 109, 122, 125, 134] and multi-pass models [29, 51, 98].

There appears to be two main reasons for the high level of interest in the problem. First, the number of triangles in a network and related quantities such as the *transitivity* or *global clustering coefficient* (the fraction of length two paths that are included in a triangle) play an important role in the analysis of real-world networks. Popular examples include motif detection in protein interaction networks [117], uncovering hidden thematic structure in the web graph [56], analysis of social networks [138], and the evaluation of large graph models [106]. Following Kutzkov et al. [31, 103], we direct the interested reader to Tsourakakis et al. [136] for an excellent overview of these and other applications. Second, the problem has a rich theory. The best exact algorithm in the RAM model runs in $O(m^{2\omega/(\omega+1)})$ time [7] where $\omega \leq 2.3728$ is the matrix multiplication exponent and m is the number of edges. Recently, Eden et al. [57] designed the first sub-linear time algorithm. Finally, there are connections to a range of important problems in the field of fine-grained complexity [83]. Since many of the real world graphs of interest are massive, it is natural that the problem has also been studied in the appropriate computation models, e.g., the MapReduce model [133] and other parallel models [12, 24], external memory models [11], and the data stream model (see above for the long stream of references).

Two data stream models have been considered in the literature on triangle counting: the *arbitrary order* model in which the stream consists of the edges of the graph in arbitrary order and the *adjacency list order* model in which all edges incident to the same node appear consecutively.[1] Of the algorithms designed in both models, some are suitable when there are many triangles whereas others dominate if there are only a few triangles. We next discuss the state-of-the-art results and these trade-offs in the context of our new results.

Our results and related work. In discussing our results and the related work we use n to denote the number of nodes in the input graph, m to be the number of edges, and T is the number of triangles in the graph.

We present two main algorithms that $(1+\epsilon)$-estimate the number of triangles for the adjacency list order model where one is suitable for processing graphs with many triangles (in particular, when $T \geq m$) and the other is suitable for processing graphs with fewer triangles (i.e., $T \leq m$).[2] Specifically, we present

1. A single-pass algorithm using $\tilde{O}(\epsilon^{-2}m/\sqrt{T})$ space and
2. A two-pass algorithm using $\tilde{O}(\epsilon^{-2}m^{3/2}/T)$ space.

We show that the space can be further reduced if we only need to distinguish triangle-free graphs from those with at least T triangles.

---

[1] The adjacency list order model is closely related to the vertex arrival model that has been considered in the context of finding matchings in the data stream model [70, 89] and row-order arrival model consider in the context of linear algebra problems [48, 67].

[2] For context, it can be shown that $T = O(m^{3/2})$ and there are graphs where $T = \Omega(m^{3/2})$ [7].

It can be argued that using only $\approx m/\sqrt{T}$ space has become a natural goal in the context of estimating the number of triangles. In particular, Cormode and Jowhari [51] showed that any constant pass algorithm in the arbitrary order model required this amount of space when $m = \Theta(n\sqrt{T})$ and there is an existing two-pass algorithm that returns a 3-approximation using this amount of space. Furthermore, Jha et al. [81] showed that this space was sufficient for additively approximating T. Unfortunately, Braverman et al. [29] showed it was insufficient for achieving multiplicative approximation via a *single-pass* algorithm in the arbitrary order model. The significance of our results is showing that $\approx m/\sqrt{T}$ space is sufficient for $1 + \epsilon$ approximation if we are given a single pass over a stream in adjacency list order or two passes over a stream in arbitrary order.

However, it is possible to improve upon $\approx m/\sqrt{T}$ space when T is large and our second algorithm does just this. At a high level, the main difference between the two types of algorithms we present is as follows. The $m/\sqrt{T}$ dependence arises when we focus on distinguishing between edges that are involved in many triangles and those that are not, whereas the $m^{3/2}/T$ dependence arises when we distinguish between high and low degree nodes. The idea of distinguishing heavy and light edges or nodes is an important idea in the non-streaming work by Alon et al. [7], Eden et al. [57], Chiba and Nishizeki [44], among others. The main challenge in our work arises from the constraints of the data stream model. This necessitates new algorithms and new notions of heavy and light that may also depend on the ordering of the stream.

Notation and preliminaries It will be convenient to assume the node set of the graph is $[n] = \{1, 2, \ldots, n\}$. Let $\Gamma(v)$ denote the neighbors of a node v and so $\deg(v) = |\Gamma(v)|$. We write (undirected) edges as sets of two nodes $\{u, v\}$ and write ordered pairs of nodes as uv. For example, $\{u, v\} = \{v, u\}$ but $uv \neq vu$. We use Δ to denote the set of triangles in the input graph and so $T = |\Delta|$. For a random variable

X, we denote the expectation and variance as $\mathrm{E}[X]$ and $\mathrm{V}[X]$ respectively. $\mathbf{Bin}(n, p)$ denotes the binomial distribution with parameters n and p.

To simplify the presentation of our algorithms we adopt two main conventions that we explain here. Following Braverman et al. [29], we restrict our attention to bounding the expected space use of our randomized algorithm rather than bounding the space with high probability. Note that if the algorithm satisfies its accuracy guarantee with probability 99/100, for example, then it is straight-forward to show that the algorithm satisfies its accuracy guarantee *and* doesn't exceed its expected space use by more than a factor 100 with probability at least 49/50. Hence, by running a logarithmic number of copies of the algorithm in parallel, terminating any that exceed their space bound, and taking the median of the remaining estimates ensures an accurate answer with only a logarithmic space increase with $1 - 1/\operatorname{poly}(n)$ probability. Secondly, we parameterize our algorithms in terms of the actual number of triangles T in the graph and various quantities in the algorithm will depend on T. Obviously, we do not know T in advance (otherwise we wouldn't be trying to estimate it) but this convention is widely adopted in the literature. A natural way to formalize this is to phrase the problem as distinguishing between graphs with at most t triangles from those with at least $(1+\epsilon)t$ triangles where t is an input parameter. In practice, the quantities in the algorithm would be initialized based on a lower or upper bound (as appropriate) for the unknown quantities.

In this model, we may assume that the stream consists of a sequence of ordered pairs xy. For each edge $\{x, y\}$, both xy and yx will be present in the stream. The promise on the ordering is that all tuples with the same first node appear consecutively in the stream. Aside from that constraint, the stream is ordered arbitrarily. For example, for the graph consisting of a cycle on three nodes $V = \{v_1, v_2, v_3\}$, a possible ordering of the stream could be

$$\langle v_3v_1, v_3v_2, v_1v_2, v_1v_3, v_2v_3, v_2v_1\rangle \ .$$

In this example, we say that the adjacency list for v_3 came first, then the adjacency list for v_1, and finally the adjacency list for v_2.

Algorithms in arbitrary order model. Some of the ideas we present here can be applied to the arbitrary order model with the use of one or few more passes through the stream. We refer to [23, 114] for detailed algorithms in the arbitrary order model.

4.2 One Pass and $\tilde{O}(\epsilon^{-2}m/\sqrt{T})$ Space

Algorithm and intuition. Define a total ordering on nodes $<_s$ based on stream ordering where $x <_s y$ if the adjacency list of x is specified before the adjacency list of y in the stream. Define

$$R_{xy} = \begin{cases} |\{z : \{x,y,z\} \in \Delta \text{ and } x <_s z <_s y\}| & \text{if } x <_s y \\ 0 & \text{if } y <_s x \end{cases}$$

and note that $\sum_{x,y} R_{xy} = T$.

The basic outline of the algorithm comprises of two interlocking parts. In the first part, we will sample each edge xy with probability p when it arrives and, until yx arrives, we count all nodes z such that $\{x, y, z\}$ forms a triangle. If we do not observe yx after xy was sampled (i.e., yx came before xy in the stream ordering) this counter will never be used. Otherwise, the counter equals R_{xy} when yx arrives. Hence, by summing these counters, we get an estimator that equals $\sum_{xy} R_{xy} I[xy \text{ sampled}]$. In expectation it equals pT and has low variance if all R_{xy} are small.

The second part ensures that we estimate every R_{xy} if $R_{xy} \geq \sqrt{T}$ regardless of whether xy was sampled. This will allow us to restrict our attention to small R_{xy} in the first part of the algorithm (and hence get a good variance bound). The critical observation that allows us to estimate every large R_{xy} is as follows: when reading the neighbors of x, even if we did not sample xy, we will have probably sampled some of the edges in the set

$$\{xz : \{x, y, z\} \in \Delta \text{ and } x <_s z <_s y\}$$

if the number of edges in this set, i.e., R_{xy}, is large. Subsequently, each of these sampled edges form a triangle with the incident edges of y and the number of these triangles can be used to a) recognize R_{xy} is large and b) to estimate R_{xy}.

The algorithm maintains two sets of edges S_1 and S_2 where each is generated by sampling each element of the stream with probability p.

For each $xy \in S_1$, we maintain a counter $c(xy)$ that counts the number of triangles $\{x, y, z\}$ where $x <_s z <_s y$. For each $xy \in S_2$, we maintain a boolean $\text{order}(xy)$ that is initially 0 but is set to 1 when yx is observed; at this point we have deduced $x <_s y$.

The elements in S_2 will be used to determine whether each R_{xy} is large and, if so, to estimate it. The elements of S_1 will be used to estimate the contribution of all R_{xy} that are not large.

```
1. Initialize $A \leftarrow 0, S_1 \leftarrow \emptyset, S_2 \leftarrow \emptyset$ and $p \leftarrow \alpha \cdot \log n \cdot \epsilon^{-2}/T^{1/2}$ for some large constant $\alpha$.
2. On seeing edges adjacent to $v$ in the adjacency stream:
    (a) Update auxiliary information about sampled edges:
        i. For all $ab \in S_1$: If $a, b \in \Gamma(v)$ then $c(ab) \leftarrow c(ab) + 1$
        ii. For all $av \in S_2$: Let $\text{order}(av) = 1$
    (b) Sample additional edges and update estimator. For each incident edge $vu$:
        i. With probability $p$, $S_1 \leftarrow \{vu\} \cup S_1$ and set $c(vu) = 0$
        ii. With probability $p$, $S_2 \leftarrow \{vu\} \cup S_2$ and set $\text{order}(vu) = 0$
        iii. Define
             $c_1(uv) := \begin{cases} c(uv) & \text{if } uv \in S_1 \\ 0 & \text{otherwise} \end{cases}$
             $c_2(uv) := |\{z : uz \in S_2, \text{order}(uz) = 1, z \in \Gamma(v)\}|$
             and say $uv$ is heavy if $c_2(uv) \geq p\sqrt{T}$. Update the estimator as follows:
             $A \leftarrow A + \begin{cases} c_1(uv) & \text{if } uv \text{ is not heavy} \\ c_2(uv) & \text{if } uv \text{ heavy} \end{cases}$
3. Return $A/p$
```

Analysis. For the analysis, let $\mathcal{H}$ consist of all edges xy such that xy is defined as heavy by the algorithm. The final value of A can be written as $A = A_l + A_h$ where

$$A_l = \sum_{xy \notin \mathcal{H}} c_1(xy) \qquad \text{and} \qquad A_h = \sum_{xy \in \mathcal{H}} c_2(xy)$$

The next two lemmas establish that, with good probability, $A_l/p \approx \sum_{xy \notin \mathcal{H}} R_{xy}$ and $A_h/p \approx \sum_{xy \in \mathcal{H}} R_{xy}$.

Lemma 22. *With probability at least* 99/100*,*

$$A_h/p = \sum_{xy \in \mathcal{H}} R_{xy} \pm \epsilon T/2$$

and $R_{xy} \leq 2\sqrt{T}$ *for all* $xy \notin \mathcal{H}$*.*

Proof. First note that $c_2(xy) \sim \mathbf{Bin}(R_{xy}, p)$. If $R_{xy} \geq \sqrt{T}/2$, then by an application of the Chernoff bound,

$$\Pr\left[c_2(xy) = (1 \pm \epsilon/2) p R_{xy}\right] \geq 1 - 2e^{-\epsilon^2 p R_{xy}/12} \geq 1 - 1/n^{10} \ .$$

Alternatively, if $R_{xy} \leq \sqrt{T}/2$ then $c_2(xy) < p\sqrt{T}$ with probability at least $1 - 1/n^{10}$. Taking the union bound over all xy establishes the lemma since

$$\sum_{xy : c_2(xy) \geq p\sqrt{T}} c_2(xy) = (1 \pm \epsilon/2) p \sum_{xy \in \mathcal{H}} R_{xy} \ .$$

□

Lemma 23. *With probability at least* 99/100*,*

$$A_l/p = \sum_{xy \notin \mathcal{H}} R_{xy} \pm \epsilon T/2 \ .$$

Proof. First note that $c_1(xy) = R_{xy}$ with probability p and 0 otherwise. Furthermore, the $c_1(\cdot)$ values are independent because they each depend on whether a different tuple was sampled. Hence

$$\mathrm{E}\left[A_l\right] = p \sum_{xy \notin \mathcal{H}} R_{xy} \quad \text{and} \quad \mathrm{V}\left[A_l\right] \leq p \sum_{xy \notin \mathcal{H}} R_{xy}^2 \leq 4pT^{3/2} .$$

since $R_{xy} \leq 2\sqrt{T}$ for $xy \notin \mathcal{H}$. By an application of the Chebyshev bound,

$$\Pr\left[\left|A_l - \mathrm{E}\left[A_l\right]\right| \geq \epsilon pT/2\right] \leq \frac{4pT^{3/2}}{(\epsilon pT/2)^2} = \frac{16}{p\epsilon^2 T^{1/2}} \leq \frac{1}{100} .$$

□

We then use the above two lemmas to prove our first main result.

Theorem 24. *There exists a* $\tilde{O}(\epsilon^{-2}m/\sqrt{T})$*-space algorithm using one pass in the adjacency list model that returns a* $(1+\epsilon)$*-approximation of* T *with probability* 49/50.

Proof. The accuracy guarantee follows from Lemmas 22 and 23. The expected space use is $\tilde{O}(pm) = \tilde{O}(\epsilon^{-2}m/\sqrt{T})$ since each edge is sampled with probability p and $\tilde{O}(1)$ bits of auxilary data is collected for each sampled node. □

4.3 Two Passes and $\tilde{O}(\epsilon^{-2}m^{3/2}/T)$ Space

Algorithm and intuition. Define a total ordering on nodes $<_d$ based on degrees where $x <_d y$ if

$$\deg(x) < \deg(y) \;\text{ or }\; (\deg(x) = \deg(y) \text{ and } \mathrm{ID}(x) < \mathrm{ID}(x)) ,$$

i.e., $<_d$ is ordering the nodes by degree with ties broken by the id of the node (recall, we assume the nodes are labelled in $1, 2, \ldots, n$). For each edge $e = \{x, y\}$, define

$$R_{\{x,y\}} = |\{z : \{x, y, z\} \in \Delta \ , \ x <_d z, y <_d z\}|$$

and note that $\sum_{e \in E} R_e = T$.

The detailed algorithm proceeds as follows.

1. *First pass:* Let $S = S' = \emptyset$ and $p = 200\epsilon^{-2}\sqrt{m}/T$. On seeing edges adjacent to v in the stream:
    (a) For each $a \in \Gamma(v)$, with probability p let $S' \leftarrow S' \cup \{va\}$ and store $\deg(v)$.
    (b) For each $bv \in S'$, let $S \leftarrow S \cup \{\{b, v\}\}$ and store $\deg(v)$.
2. *Second pass:* Let $A = 0$. On seeing edges adjacent to v in the stream:
    (a) For each edge $\{a, b\} \in S$ such that $a <_d v$, $b <_d v$, and $a, b \in \Gamma(v)$, $A \leftarrow A + 1$.
3. *Output:* Return A/p.

After the first pass, S contains each edge in the graph sampled with probability p along with the degrees of the endpoints. In the second pass, we count the number of triangles $\{a, b, v\}$ where $\{a, b\} \in S$, $a <_d v$, and $b <_d v$.

The basic idea for the algorithm in this section is as follows: In the first pass, we generate a sample of edges S along with the degree of each endpoint of the sample edges. In the second pass, for each $e \in S$ we compute R_e and return $\sum_{e \in S} R_e$. This will equal pT in expectation and we will be able to bound the variance by first showing that $R_e \leq \sqrt{2m}$ for all $e \in E$.

Analysis. We first prove a bound on $\max R_e$ that will be required to bound the variance of our estimator.

Lemma 25. $\max R_e \leq \sqrt{2m}$.

Proof. Let $e = \{x, y\}$ and suppose $R_e = R_{\{x,y\}} > \sqrt{2m}$. Then $\deg(x) \geq \sqrt{2m}$. Furthermore, there exist at least $\sqrt{2m}$ nodes $z_1, z_2, \ldots$ such that $\{x, y, z_i\}$ is a triangle and $\deg(z_i) \geq \deg(x) > \sqrt{2m}$. But then

$$\deg(z_1) + \deg(z_2) + \ldots > \sqrt{2m} \cdot \sqrt{2m} = 2m$$

which is a contradiction since the sum of degrees of every node in the graph is $2m$. $\square$

Theorem 26. *There exists an $\tilde{O}(\epsilon^{-2}m^{3/2}/T)$-space algorithm using two passes in the adjacency list model that returns a $(1+\epsilon)$-approximation of T with probability* $99/100$.

Proof. Consider the above algorithm and note that each edge e is contained in S with probability p and $A = \sum_{e \in S} R_e$. Hence, by appealing to Lemma 25,

$$\mathrm{E}[A] = Tp \qquad \text{and} \qquad \mathrm{V}[A] < \sum_{e \in E} R_e^2 p \leq \sqrt{2m}Tp$$

Then, by the Chebyshev bound,

$$\begin{aligned} \Pr[|A/p - T| \geq \epsilon T] \leq & (\sqrt{2m}T/p)/(\epsilon^2 T^2) \\ = & p^{-1}\epsilon^{-2}\sqrt{2m}/T \leq 1/100. \end{aligned}$$

Hence the algorithm has the desired accuracy. The expected space use is $\tilde{O}(pm) = \tilde{O}(\epsilon^{-2}m^{3/2}/T)$. $\square$

Improved algorithm for testing triangle-freeness. We conclude this section by showing that if we are only trying to distinguish triangle-free graphs from those with at least T triangles, less space is sufficient.

Theorem 27. *There exists an* $\tilde{O}(m/T^{2/3})$*-space algorithm using two passes in the adjacency list model that distinguishes triangle-free graphs from those with at least* T *triangles with probability* $99/100$*.*

The basic observation is that a graph with T triangles has at least $T^{2/3}$ edges involved in these triangles. This follows because any graph with m edges can have at most $O(m^{3/2})$ triangles (see, e.g., [7]). Hence, if each edge is sampled with probability $p = \alpha/T^{2/3}$ for some large constant $\alpha > 0$ at least one edge $\{u, v\}$ in some triangle $\{u, v, z\}$ will the sampled. We do this sampling in the first pass. Then, in the second pass of the algorithm when the neighbors of z are observed we will identify a triangle by tracking which of the sampled edges have two endpoints in $\Gamma(z)$.

CHAPTER 5

FINDING APPROXIMATE MAXIMUM COVERAGE IN THE STREAMING SET MODEL

5.1 Introduction and Related Work

The *maximum set coverage problem* is a classic NP-Hard problem that has a wide range of applications including facility and sensor allocation [101], information retrieval [8], influence maximization in marketing strategy design [93], and the blog monitoring problem where we want to choose a small number of blogs that cover a wide range of topics [128]. In this problem, we are given a set system of m sets that are subsets of a universe $[n] := \{1, \ldots, n\}$. The goal is to find the k sets whose union covers the largest number of distinct elements. For example, in the application considered by Saha and Getoor [128], the universe corresponds to n topics of interest to a reader, each subset corresponds to a blog that covers some of these topics, and the goal is to maximize the number of topics that the reader learns about if she can only choose k blogs.

It is well-known that the greedy algorithm, which greedily picks the set that covers the most number of uncovered elements, is a $1 - 1/e$ approximation. Furthermore, unless $P = NP$, this approximation factor is the best possible [62].

The *maximum vertex coverage problem* is a special case of this problem in which the universe corresponds to the edges of a given graph and there is a set corresponding to each node of the graph that contains the edges that are incident to that node. For this problem, algorithms based on linear programming achieve a $3/4$ approximation for general graphs [1] and a $8/9$ approximation for bipartite graphs [34]. Assuming

$P \neq NP$, there does not exist a polynomial-time approximation scheme. Recent work has focused on finding purely combinatorial algorithms for this problem [26].

Streaming algorithms. Unfortunately, for both problems, the aforementioned greedy and linear programming algorithms scale poorly to massive data sets. This has motivated a significant research effort in designing algorithms that could handle large data in modern computation models such as the data stream model and the MapReduce model [17, 102]. In the data stream model, the k-set coverage problem and the related set cover problem have received a lot of attention in recent research [14,16,39,58,77,141].

Two variants of the data stream model are relevant to our work. In the *streaming-set model* [58, 74, 96, 126, 128, 132], the stream consists of m sets $S_1, \ldots, S_m$ and each S_i is encoded as the list of elements in that set along with a unique ID for the set. For simplicity, we assume that $\mathrm{ID}(S_i) = i$. In the dynamic graph stream model [3–6, 15, 25, 45, 74, 91, 92, 99, 112, 114], relevant to the maximum vertex coverage problem, the stream consists of insertions and deletions of edges of the underlying graph. For a recent survey of research in graph streaming, see [111]. Note that any algorithm for the dynamic graph stream model can also be used in the streaming-set model; the streaming-set model is simply a special case in which there is no deletion and edges are grouped by endpoint.

Related work. For the maximum set coverage problem, Saha and Getoor [128] gave a swap based 1/4 approximation algorithm that uses a single pass and $\tilde{O}(kn)$ space. At any point, their algorithm stores k sets explicitly in the memory as the current solution. When a new set arrives, based on a specific rule, their algorithm either swaps it with the set with the least contribution in the current solution or does nothing and moves on to the next set in the stream. Subsequently, Ausiello et al. [16] gave a slightly different swap based algorithm that also finds a 1/4 approximation using one pass and the same space. Yu and Yuan [141] claimed an $\tilde{O}(n)$ space, single-pass algorithm with an approximation factor around 0.3 based on the aid of computer simulation.

Recently, Badanidiyuru et al. [17] gave a generic single-pass algorithm for maximizing a monotone submodular function on the stream's items subject to the cardinality constraint that at most k objects are selected. Their algorithm guarantees a $1/2 - \epsilon$ approximation. At a high level, based on a rule that is different from [16, 128] and a guess of the optimal value, their algorithm decides if the next item (which is a set in our case) is added to the current solution. The algorithm stops when it reaches the end of the stream or when k items have been added to the solution. In the maximum set coverage problem, the rule requires knowing the coverage of the current solution. As a result, a careful adaptation to the maximum set coverage problem uses $\tilde{O}(\epsilon^{-1}n)$ space. For constant ϵ, this result directly improves upon [16, 128]. Subsequently, Chekuri et al. [41] extended this work to non-monotone submodular function maximization under constraints beyond cardinality.

The set cover problem, which is closely related to the maximum set coverage problem, has been studied in [14, 39, 58, 77, 128]. See [14] for a comprehensive summary of results and discussion.

For the maximum vertex coverage problem, Ausiello et al. [16]. They first observed that simply outputting the k vertices with highest degrees is a $1/2$ approximation; this can easily be done in the streaming-set model. The main results of their work were $\tilde{O}(kN)$-space algorithms that have better approximation for special types of graph. Their results include a 0.55 approximation for regular graphs and a 0.6075 approximation for regular bipartite graphs. Note that their paper only considered the streaming-set model whereas our results for maximum vertex coverage will consider the more challenging dynamic graph stream model.

Our contributions *Maximum k-set coverage.* Our main goal is to achieve the $1 - 1/e$ approximation that is possible in the non-streaming or offline setting.

- We present polynomial time data stream algorithms that achieve a $1-1/e-\epsilon$ approximation for arbitrarily small ϵ. The first algorithm uses one pass and $\tilde{O}(\epsilon^{-2}m)$ space whereas the second algorithm uses $O(\epsilon^{-1})$ passes and $\tilde{O}(\epsilon^{-2}k)$ space. We consider both algorithms to be pass efficient but the second algorithm uses much less space at the cost of using more than one pass. We note that storing the solution itself requires $\Omega(k)$ space. Thus, we consider $\tilde{O}(\epsilon^{-2}k)$ space to be surprisingly space efficient.

- For constant k, we show that $\Omega(m)$ space is required by any constant pass (randomized) algorithm to achieve an approximation factor better than $1-(1-1/k)^k$ with probability at least 0.99; this holds even if the algorithm is permitted exponential time. To the best of our knowledge, this is the first non-trivial space lower bound for this problem. However, with exponential time and $\tilde{O}(\epsilon^{-2}m\cdot\min(k,\epsilon^{-1}))$ space we observe that a $1-\epsilon$ approximation is possible in a single pass.

For a slightly worse approximation, a $1/2-\epsilon$ approximation in one pass can be achieved using $\tilde{O}(\epsilon^{-3}k)$ space. This follows by building on the result of Badanidiyuru et al. [17]. However, we provide a simpler algorithm and analysis.

Our approach generalizes to the group cardinality constraint in which there are ℓ groups and only k_i sets from group i can be selected. This is also known as the partition matroid constraint. We give a $1/(\ell+1)-\epsilon$ approximation which improves upon [37,41] for the case $\ell=2$. Let $k=k_1+\ldots+k_\ell$. If $O(\epsilon^{-1}\log(k/\epsilon))$ passes are permitted, then we could achieve a $1/2-\epsilon$ approximation by adapting the greedy analysis in [43] to our framework.

Finally, we design a $1/3-\epsilon$ approximation algorithm for the budgeted maximum set coverage problem using one pass and $\tilde{O}(\epsilon^{-1}(n+m))$ space. In this version, each set S has a cost w_S in the range $[0,L]$. The goal is to find a collection of sets whose total cost does not exceed L that cover the most number of distinct elements. Khuller et al. [94] presented a polynomial time and $1-1/e$ approximation algorithm based on the greedy algorithm and an enumeration technique. Our results are summarized in Figure 5.1.

In an independent work that was published shortly after, Bateni et al. [22] also presented a polynomial-time, single-pass, $\tilde{O}(\epsilon^{-3}m)$ space algorithm that finds a $1-1/e-\epsilon$ approximation for the maximum k-set coverage problem. Furthermore, given unlimited post-processing time, their results also imply a $1-\epsilon$ approximation using a single-pass and $\tilde{O}(\epsilon^{-3}m)$ space. Recently, Assadi proved a space lower bound $\Omega(\epsilon^{-2}m)$ for any $1-\epsilon$ approximation for constant k [13]. We also note that our approach also works in their *edge arrival* model in which the stream reveals the set-element relationships one at a time.

Maximum k-vertex coverage. Compared to the most relevant previous work [16], we study this problem in a more general model, i.e., the dynamic graph stream model. We manage to achieve a better approximation and space complexity for general graphs even when comparing to their results for special types of graph. Our results are summarized in Figure 5.2. In particular, we show that

- $\tilde{O}(\epsilon^{-2}N)$ space is sufficient for a $1-\epsilon$ approximation (or a $3/4-\epsilon$ approximation if restricted to polynomial time) and arbitrary k in a single pass. The algorithms in [16] use $\tilde{O}(kN)$ space and achieve an approximation worse than 0.61 even for regular bipartite graphs.

- Any constant approximation in constant passes requires $\Omega(N)$ space for constant k.

- For regular graphs, we show that $\tilde{O}(\epsilon^{-3}k)$ space is sufficient for $1-\epsilon$ approximation in a single pass. We generalize this to an $\kappa-\epsilon$ approximation when the ratio between the minimum and maximum degree is bounded below by κ. We also extend this result to hypergraphs.

Bound	No. of passes	Space	Approximation	Constraint
U	$O\left(\epsilon^{-1}\right)$	$\tilde{O}(\epsilon^{-2}k)$	$1-1/e-\epsilon$	C
U	1	$\tilde{O}(\epsilon^{-3}k)$	$1/2-\epsilon$	C
U	1	$\tilde{O}(\epsilon^{-2}m)$	$1-1/e-\epsilon$	C
U	1	$\tilde{O}(\epsilon^{-2}m\cdot\min(k,\epsilon^{-1}))$	$1-\epsilon$	C
L	Constant	$\Omega(mk^{-2})$	$1-(1-1/k)^k+\epsilon$	C
U	1	$\tilde{O}(\epsilon^{-3}k)$	$1/(\ell+1)-\epsilon$	G
U	$O(\epsilon^{-1}\log(k/\epsilon))$	$\tilde{O}(\epsilon^{-2}k)$	$1/2-\epsilon$	G
U	1	$\tilde{O}(\epsilon^{-1}(n+m))$	$1/3-\epsilon$	B

Figure 5.1. Summary of results for MaxSetCoverage, U: upper bound, L: lower bound, C: cardinality, B: budget, G: group cardinality (partition matroid)

Bound	No. of passes	Space	Approximation
U	1	$\tilde{O}(\epsilon^{-2}N)$	$1-\epsilon$
U	1	$\tilde{O}(\epsilon^{-3}k)$	$\kappa-\epsilon$
L	1	$\Omega(N\kappa^3/k)$	$\kappa+\epsilon$

Figure 5.2. Summary of results for MaxVertexCoverage, U: upper bound, L: lower bound, κ is ratio of lowest degree to highest degree.

Our techniques. On the algorithmic side, our basic approach is a "guess, subsample, and verify" framework. At a high level, suppose we design a streaming algorithm for approximating the maximum k-set coverage that assumes a priori knowledge of a good guess of the optimal coverage. We show that it is a) possible to run same algorithm on a subsampled universe defined by a carefully chosen hash function and b) remove the assumption that a good guess was already known.

If the guess is at least nearly correct, running the algorithm on the subsampled universe results in a small space complexity. However, there are two main challenges. First, an algorithm instance with a wrong guess could use too much space. We simply terminate those instances. The second issue is more subtle. Because the hash function is not fully independent, we appeal to a special version of the Chernoff bound. The bound needs not guarantee a good approximation unless the guess is near-correct. To this end, we use the F_0 estimation algorithm to verify the coverage of the solutions. Finally, we return the solution with maximum estimate coverage. This framework allows us to restrict the analysis solely to the near-correct guess.

Some of our other algorithmic ideas are inspired by previous works. The "thresholding greedy" technique was inspired by [18, 39, 52]. However, the analysis is different for our problem. Furthermore, to optimize the number of passes, we rely on new observations.

Another algorithmic idea in designing one-pass space-efficient algorithm is to treat the sets differently based on their contributions. During the stream, we immediately add the sets with large contributions to the solution. We store the contribution of each remaining sets explicitly and solve the remaining problem offline. Har-Peled et al. [77] devised a somewhat similar strategy for the set-cover problem, but the details are different.

For the maximum k-vertex coverage problem, we show that simply running the streaming cut-sparsifier algorithm is sufficient and optimal up to a polylog factor. The novelty is to treat it as an interesting corner case of a more space-efficient algorithm for near regular graphs, i.e., κ is bounded below.

One of the novelties is proving the lower bound via a randomized reduction from the k-party set disjointness problem.

5.2 Algorithms for Maximum k-Set Coverage

In this section, we design various algorithms for approximating MaxSetCoverage in the data stream model. Our main algorithmic results in this section are two $1 - 1/e - \epsilon$ approximation algorithms. The first algorithm uses one pass and $\tilde{O}(\epsilon^{-2} m)$ space whereas the second algorithm uses $O(\epsilon^{-1})$ passes and $\tilde{O}(\epsilon^{-2} k)$ space. We also briefly explore some other trade-offs in a subsequent subsection.

Notation. If $\mathcal{A}$ is a collection of sets, then $\mathcal{C}(\mathcal{A})$ denotes the union of these sets.

5.2.1 $(1 - 1/e - \epsilon)$ Approximation in One Pass and $\tilde{O}(\epsilon^{-2}m)$ Space

Approach. The algorithm adds sets to the current solution if the number of new elements they cover exceeds some threshold. The basic algorithm relies on an estimate z of the optimal coverage OPT. The threshold for immediately including a new set in the solution is that it covers at least z/k new elements. Unfortunately, this threshold is too high to ensure that we selected sets that achieve the required $1 - 1/e - \epsilon$ approximation and we may want to revisit adding a set, say S, that was not added when it first arrived. To facilitate this, we will explicitly store the subset of S that were uncovered when S arrived in a collection of sets $\mathcal{W}$. Because S was not added immediately, we know that this subset is not too large. At the end of the pass, we continue augmenting out current solutions using the collection $\mathcal{W}$.

Technical details. For the time being, we suppose that the algorithm is provided with an estimate z such that $\text{OPT} \leq z \leq 4\,\text{OPT}$. We will later remove this assumption. The algorithm uses C to keep track of the elements that have been covered so far. Upon seeing a new set S, the algorithm stores $S \setminus C$ explicitly in $\mathcal{W}$ if S covers few new elements. Otherwise, the algorithm adds S to the solution and updates C immediately. At the end of the stream, if there are fewer than k sets in the solution, we use the greedy approach to find the remaining sets from $\mathcal{W}$.

The basic algorithm maintains $I \subseteq [m]$, $C \subseteq [n]$ where I corresponds to the ID's of the (at most k) sets in the current solution and C is the the union of the corresponding sets. We also maintain a collection of sets $\mathcal{W}$ described above. The algorithm proceeds as follows:

```
1. Initialize $C = \emptyset$, $I = \emptyset$, $\mathcal{W} = \emptyset$.
2. For each set $S$ in the stream:
    (a) If $|S \setminus C| < z/k$ then $\mathcal{W} \leftarrow \mathcal{W} \cup \{S \setminus C\}$.
    (b) If $|S \setminus C| \geq z/k$ and $|I| < k$, then $I \leftarrow I \cup \{ID(S)\}$ and $C \leftarrow C \cup S$.
3. Post-processing: Greedily add $k - |I|$ sets from $\mathcal{W}$ and update $I$ and $C$ appropriately.
```

Lemma 28. *There exists a single-pass, $O\left(k \log m + mz/k \cdot \log n\right)$-space algorithm that finds a $1 - 1/e$ approximation of* MaxSetCoverage.

Proof. We observe that storing the set of covered elements C requires at most $\mathrm{OPT} \log n = O(z \log n)$ bits of space. For each set S such that $S \setminus C$ is stored explicitly in $\mathcal{W}$, we need $O\left(z/k \cdot \log n\right)$ bits of space. Storing I requires $O(k \log m)$ space. Thus, the algorithm uses the space as claimed.

After the algorithm added the ith set S to the solution, let a_i be the number of new elements that S covers and b_i be the total number of covered elements so far. Furthermore, for $i \geq 0$, let $c_i = \mathrm{OPT} - b_i$. Define $a_0 := 0$ and $b_0 := 0$.

At the end of the stream, suppose $|I| = j$. Then,

$$c_j \leq \mathrm{OPT} - \frac{z \cdot j}{k} \leq \mathrm{OPT}\left(1 - \frac{j}{k}\right) \leq \mathrm{OPT}\left(1 - \frac{1}{k}\right)^j .$$

The last inequality holds when $k \geq 2$ and j is a non-negative integer. The case $k = 1$ is trivial since we can simply find the largest set in $\tilde{O}(1)$ space.

Now, we consider the sets that were added in post-processing. We then proceed with the usual inductive argument to show that $c_i \leq (1 - 1/k)^i \,\mathrm{OPT}$ for $i > j$. Before the algorithm added the $(i+1)$th set for $j \leq i \leq k-1$, there must be a set that covers at least c_i/k new elements. Therefore,

$$c_{i+1} = c_i - a_{i+1} \leq c_i \left(1 - \frac{1}{k}\right) \leq \text{OPT} \left(1 - \frac{1}{k}\right)^{i+1} .$$

The approximation follows since $c_k \leq \text{OPT}(1-1/k)^k \leq 1/e \cdot \text{OPT}$. □

Following the approach outlined in Section 5.2.3 we may assume $z = O(\epsilon^{-2} k \log m)$ and that $\text{OPT} \leq z \leq 4\,\text{OPT}$.

Theorem 29. *There exists a single-pass, $\tilde{O}(\epsilon^{-2} m)$ space algorithm that finds a $1 - 1/e - \epsilon$ approximation of* MaxSetCoverage *with high probability.*

Better approximation using more space and unlimited post-processing time. We observe that a slight modification of the above algorithm can be used to attain a $1 - 1/(4b)$ approximation for any $b > 1$ if we are permitted unlimited post-processing time and an extra factor of b in the space use. Specifically, we increase the threshold for when to add a set immediately to the solution from z/k to bz/k and then find the optimal collection of $k - |I|$ sets from $\mathcal{W}$ to add in post-processing. It is immediate that this algorithm uses $O(k \log m + mbz/k \cdot \log n)$ space.

Suppose a collection of y sets $\mathcal{S}_1$ were added during the stream. These y sets cover

$$|\,\mathcal{C}(\mathcal{S}_1)| \geq y \cdot \frac{bz}{k} \geq \text{OPT} \cdot \frac{yb}{k}$$

elements. On the other hand, the collection of sets $\mathcal{S}_2$ selected in post-processing covers at least $\frac{k-y}{k} \cdot (\text{OPT} - |\,\mathcal{C}(\mathcal{S}_1)|)$ new elements. Then,

$$\begin{aligned}
|\mathcal{C}(\mathcal{S}_1 \cup \mathcal{S}_2)| &\geq |\mathcal{C}(\mathcal{S}_1)| + \frac{k-y}{k} \cdot (\mathrm{OPT} - |\mathcal{C}(\mathcal{S}_1)|) \\
&= \left(1 - \frac{y}{k}\right) \mathrm{OPT} + \frac{y}{k} \cdot |\mathcal{C}(\mathcal{S}_1)| \\
&\geq \left(1 - \frac{y}{k}\right) \mathrm{OPT} + \frac{y}{k} \cdot \mathrm{OPT} \cdot \frac{yb}{k} \\
&= \mathrm{OPT} \left(1 - \frac{y}{k} + \left(\frac{y}{k}\right)^2 b\right) \\
&\geq \mathrm{OPT} \left(1 - \frac{1}{4b}\right)
\end{aligned}$$

where the last inequality follows by minimizing over y. Hence, we obtain a $1-\epsilon$ approximation by setting $b = 4/\epsilon$.

Theorem 30. *There exists a single-pass, $\tilde{O}(\epsilon^{-3}m)$ space algorithm that finds a $1-\epsilon$ approximation of* MaxSetCoverage *with high probability.*

5.2.2 $(1 - 1/e - \epsilon)$ Approximation in $O(\epsilon^{-1})$ Passes and $\tilde{O}(\epsilon^{-2}k)$ Space

Approach. Our second algorithm is based on the standard greedy approach but instead of adding the set that increases the coverage of the current solution the most at each set, we add a set if the number of new elements covered by this set exceeds a certain threshold. This threshold decreases with each pass in such a way that after only $O(\epsilon^{-1})$ passes, we have a good approximate solution but the resulting algorithm may use too much space. We will fix this by first randomly subsampling each set at different rates and running multiple instantiations of the basic algorithm corresponding to different rates of subsampling.

The basic "decreasing threshold" approach has been used before in different contexts [18, 39, 52]. The novelty of our approach is in implementing this approach such that the resulting algorithm uses small space and a small number of passes. For example, a direct implementation of the approach by Badanidiyuru and Vondrák [18] in the streaming model may require $O(\epsilon^{-1}\log(m/\epsilon))$ passes and $O(n)$ space[1].

Technical details. We will assume that we are given an estimate z of OPT such that $\text{OPT} \leq z \leq 4\,\text{OPT}$. We start by designing a $(1 - 1/e - \epsilon)$ approximation algorithm that uses $\tilde{O}(k + z)$ space and $O(\epsilon^{-1})$ passes. We will subsequently use a sampling approach to reduce the space to $\tilde{O}(\epsilon^{-2}k)$.

As with the previous algorithm, the basic algorithm in this section also maintains $I \subseteq [m]$, $C \subseteq [n]$ where I corresponds to the ID's of the (at most k) sets in the current solution and C is the the union of the corresponding sets. The algorithm proceeds as follows:

```
1. Initialize $C = \emptyset$ and $I = \emptyset$
2. For $j = 1$ to $1 + \lceil \log_\alpha(4e) \rceil$ where $\alpha = 1 + \epsilon$:
    (a) Make a pass over the stream. For each set $S$ in the stream: If $|I| < k$ and
            $|S \setminus C| \geq \frac{z}{k(1+\epsilon)^{j-1}}$ ,
        then $I \leftarrow I \cup \{ID(S)\}$ and $C \leftarrow C \cup S$.
```

Lemma 31. *There exists an $O(\epsilon^{-1})$-pass, $O(k \log m + z \log n)$-space algorithm that finds a $1 - 1/e - \epsilon$ approximation of* MaxSetCoverage.

To analyze the algorithm, we introduce some notation. After the ith set was picked, let a_i be the number of new elements covered by this set and let b_i be the total number of covered elements so far. Furthermore, let $c_i = \text{OPT} - b_i$. We define $a_0 := 0$ and $b_0 := 0$.

---

[1] Note that their work addressed the more general problem of maximizing sub-modular functions.

Lemma 32. *Suppose the algorithm picks k' sets. For $0 \leq i \leq k'-1$, we have $a_{i+1} \geq c_i/(\alpha k)$.*

Proof. Suppose the algorithm added the $(i+1)$th set S during the jth pass. Consider the set of covered elements C just before the algorithm added the set S.

We first consider the case where $j = 1$. Then, the algorithm only adds S if

$$|S \setminus C| \geq \frac{z}{k} \geq \frac{\mathrm{OPT}}{k} \geq \frac{c_i}{k} \geq \frac{c_i}{\alpha k} \ .$$

Now, we consider the case where $j > 1$. Note that just before the algorithm added S, there must exist a set S' (which could be S) that had not been already added where $|S' \setminus C| \geq c_i/k$. This follows because the optimal collection of k sets covers at least c_i elements that are currently uncovered and hence one of these sets must cover at least c_i/k new elements. But since S' had not already been added, we know that S' was not added during the first $j-1$ passes and thus, $|S' \setminus C| < z/(k\alpha^{j-2})$. Therefore,

$$\frac{z}{k\alpha^{j-2}} > |S' \setminus C| \geq \frac{c_i}{k}$$

and in particular, $z/(k\alpha^{j-1}) > c_i/(k\alpha)$. Since the algorithm picked S, we have

$$a_{i+1} = |S \setminus C| \geq \frac{z}{k\alpha^{j-1}} > \frac{c_i}{k\alpha}$$

as required. □

Proof of Lemma 31. It is immediate that the number of passes is $O(\epsilon^{-1})$. The algorithm needs to store the sets I and C. Since $|C| \leq z$, the total space is $O(k \log m + z \log n)$.

To argue about the approximation factor, we first prove by induction that we always have $c_i \leq \left(1 - \frac{1}{\alpha k}\right)^i \mathrm{OPT}$ for $i \leq k'$. Trivially, $c_0 \leq \left(1 - \frac{1}{\alpha k}\right)^0 \mathrm{OPT}$. Suppose $c_i \leq (1 - \frac{1}{\alpha k})^i \mathrm{OPT}$. Then, according to Lemma 32, $a_{i+1} \geq c_i/(\alpha k)$. Thus,

$$c_{i+1} = c_i - a_{i+1} \leq c_i - \frac{c_i}{\alpha k} = c_i\left(1 - \frac{1}{\alpha k}\right) \leq \mathrm{OPT}\left(1 - \frac{1}{\alpha k}\right)^{i+1}.$$

Suppose the final solution contains k sets. Then

$$c_k \leq \left(1 - \frac{1}{\alpha k}\right)^k \mathrm{OPT} \leq e^{-1/\alpha}\,\mathrm{OPT} \leq \left(\frac{1}{e} + \epsilon\right)\mathrm{OPT}\,.$$

As a result, the final solution covers $b_k = \mathrm{OPT} - c_k \geq (1 - 1/e - \epsilon)\,\mathrm{OPT}$ elements.

Suppose the collection of sets $\mathcal{S}$ chosen by the algorithm contains fewer than k sets. We define $\tilde{S} := S \setminus \mathcal{C}(\mathcal{S})$ to be the set of elements in S that are not covered by the final solution. For each set S in the optimal solution $\mathcal{O}$, if S was not added, then $|\tilde{S}| \leq z/(4ek)$. Therefore,

$$\begin{aligned}\mathrm{OPT} = \left|\bigcup_{S\in\mathcal{O}} (S \cap \mathcal{C}(\mathcal{S}))\right| + \left|\bigcup_{S\in\mathcal{O}\setminus\mathcal{S}} \tilde{S}\right| \leq |\mathcal{C}(\mathcal{S})| + \sum_{S\in\mathcal{O}\setminus\mathcal{S}} \left|\tilde{S}\right| &\leq |\mathcal{C}(\mathcal{S})| + \frac{z}{4e} \\ &\leq |\mathcal{C}(\mathcal{S})| + \frac{\mathrm{OPT}}{e}\,.\end{aligned}$$

Hence, $|\mathcal{C}(\mathcal{S})| \geq (1 - 1/e)\,\mathrm{OPT}$. □

Following the approach outlined in Section 5.2.3 we may assume $z = O(\epsilon^{-2} k \log m)$ and that $\mathrm{OPT} \leq z \leq 4\,\mathrm{OPT}$.

Theorem 33. *There exists an $O(\epsilon^{-1})$-pass, $\tilde{O}(\epsilon^{-2}k)$ space algorithm that finds a $1 - 1/e - \epsilon$ approximation of* MaxSetCoverage *with high probability.*

5.2.3 Removing Assumptions via Guessing, Sampling, and Sketching

In this section, we address the fact that in the previous two sections we assumed a priori knowledge of a constant approximation of the maximum number of elements that could be covered and that this optimum was of size $O(\epsilon^{-2} k \log m)$.

Addressing both issues are interrelated and are based on a subsampling approach. The basic idea is to run the above algorithms on a new instance formed by removing occurrences of certain elements in $[n]$ from all the input sets. The goal is to reduce the maximum coverage to $\min(n, O(\epsilon^{-2} k \log m))$ while ensuring that a good approximation in the subsampled instance corresponds to a good approximation in the original instance. In the rest of this section we will assume that $k = o(\epsilon^2 n / \log m)$ since otherwise this bound is trivial.

In this section, we will need to use the following Chernoff bound for limited independent random variables.

Theorem 34 (Schmidt et al. [129])**.** *Let* $X_1, \ldots, X_n$ *be binary random variables. Let* $X = \sum_{i=1}^{n} X_i$ *and* $\mu = \mathrm{E}[X]$*. Suppose* $\mu \leq n/2$*. If* X_i *are* $\lceil \gamma\mu \rceil$*-wise independent, then*

$$\Pr[|X - \mu| \geq \gamma\mu] \leq \exp\left(-\lfloor \min(\gamma, \gamma^2) \cdot \mu/3 \rfloor\right) .$$

Subsampling. Assume we know a value v that satisfies $\mathrm{OPT}/2 \leq v \leq \mathrm{OPT}$. Let c be some sufficiently large constant and set $\lambda = c\epsilon^{-2} k \log m$. Let $h : [n] \to \{0, 1\}$ be drawn from a family of 2λ-wise independent hash functions where

$$p := \Pr[h(e) = 1] = \lambda/v.$$

The space to store h is $\tilde{O}(\epsilon^{-2} k)$. For any set S that is a subset of $[n]$, we define

$$S' := \{e \in S : h(e) = 1\}.$$

The next lemma and its corollary will allow us to argue that approximating the maximum coverage among the elements $\{e \in [n] : h(e) = 1\}$ gives only a slightly weaker approximation of the maximum coverage among the original set of elements.

Lemma 35. *With high probability*[2]*, for all collections of k sets $S_1, \ldots, S_k$ in the stream,* $|S'_1 \cup \ldots \cup S'_k| = |S_1 \cup \ldots \cup S_k| p \pm \epsilon v p$.

Proof. Fix any collection of k sets $S_1, \ldots, S_k$. Let $D = |S_1 \cup \ldots \cup S_k|$ and $D' = |S'_1 \cup \ldots \cup S'_k|$. We first observe that since $k = o(\epsilon^2 n / \log m)$, we may assume that $\lambda = o(n)$.

$$\mu := \mathrm{E}[D'] = pD \leq p\,\mathrm{OPT} < 2pv = 2\lambda \leq n/2.$$

Appealing to the Chernoff bound with limited independence (Theorem 34) with the binary variables $X_i = 1$ if and only if $i \in S_1 \cup \ldots \cup S_k$ and $h(i) = 1$, i.e, $D' = \sum_{i=1}^{n} X_i$, we have

$$\Pr[|D' - \mu| \geq \epsilon v p] = \Pr[|D' - \mu| \geq \gamma D p] \leq \exp\left(-\lfloor \min(\gamma, \gamma^2) \cdot \mu/3 \rfloor\right)$$

where $\gamma = \epsilon v / D$ since the hash function is $\lceil \gamma \mu \rceil = \lceil \epsilon v p \rceil$-wise independent. But note that

$$\begin{aligned} \exp\left(-\lfloor \min(\gamma, \gamma^2) \cdot \frac{\mu}{3} \rfloor\right) &= \exp\left(-\lfloor \min(1, \gamma) \cdot \frac{\epsilon v p}{3} \rfloor\right) \\ &\leq \exp\left(-\left\lfloor \frac{1}{2} \cdot \frac{ck \log m}{3} \right\rfloor\right) \leq \frac{1}{m^{10k}} \end{aligned}$$

where we use the fact that $\gamma = \epsilon v / D \geq \epsilon/2$ because $D \leq \mathrm{OPT} \leq 2v$. The lemma follows by taking the union bound over all $\binom{m}{k}$ collections of k sets. □

In particular, the following corollary establishes that a $1/t$ approximation when restricted to elements in $\{e \in [n] : h(e) = 1\}$ yields a $(1/t - 2\epsilon)$ approximation and at most $p\,\mathrm{OPT}(1 + \epsilon) = O(\epsilon^{-2} k \log m)$ of these elements can be covered by k sets.

---

[2] We consider $1 - 1/\operatorname{poly}(m)$ or $1 - 1/\operatorname{poly}(n)$ as high probability.

Corollary 36. *Let* OPT' *be optimal number of elements that can be covered from* $\{e \in [n] : h(e) = 1\}$*. Then, with high probability,*

$$p\,\mathrm{OPT}(1+\epsilon) \geq \mathrm{OPT}' \geq p\,\mathrm{OPT}(1-\epsilon)$$

Furthermore, with high probability, if $U_1, \ldots, U_k$ *satisfies* $|U_1' \cup \ldots \cup U_k'| \geq p\,\mathrm{OPT}(1-\epsilon)/t$ *for* $t \geq 1$ *then*

$$|U_1 \cup \ldots \cup U_k| \geq \mathrm{OPT}\left(\frac{1}{t} - 2\epsilon\right) .$$

Proof. The fact that $\mathrm{OPT}' \geq p\,\mathrm{OPT}(1-\epsilon)$ follows by applying Lemma 35 to the optimal solution. According to Lemma 35, for all collections of k sets $U_1, \ldots, U_k$, we have

$$|U_1' \cup \ldots \cup U_k'| = |U_1 \cup \ldots \cup U_k|p \pm \epsilon v p \leq p\,\mathrm{OPT}(1+\epsilon)$$

which implies the first inequality.

Now, suppose $|U_1' \cup \ldots \cup U_k'| \geq p\,\mathrm{OPT}(1-\epsilon)/t$. Since $|U_1' \cup \ldots \cup U_k'| - \epsilon v p \leq |U_1 \cup \ldots \cup U_k|p$, we deduce that $|U_1 \cup \ldots \cup U_k| \geq \mathrm{OPT}(1-\epsilon)/t - \epsilon v \geq \mathrm{OPT}(1/t - 2\epsilon)$. □

Hence, since we know v such that $\mathrm{OPT}/2 \leq v \leq \mathrm{OPT}$, then we know that

$$(1-\epsilon)\lambda \leq \mathrm{OPT}' \leq 2(1+\epsilon)\lambda \tag{5.1}$$

with high probability according to Corollary 36. Then, by setting $z = 2(1+\epsilon)\lambda$, we ensure that $\mathrm{OPT}' \leq z \leq 4\,\mathrm{OPT}'$.

Guessing v and F_0 sketching. We still need to address how to compute v such that $\mathrm{OPT}/2 \leq v \leq \mathrm{OPT}$. The natural approach is to make $\lceil \log_2 n \rceil$ guesses for v corresponding to $1, 2, 4, 8 \ldots$ since one of these will be correct.[3] We then perform multiple parallel instantiations of the algorithm corresponding to each guess. This increases the space by a factor of $O(\log n)$.

---

[3] The number of guesses can be reduced to $\lceil \log_2 k \rceil$ if the size of the largest set is known since this gives a k approximation of OPT. The size of the large set can be computed in one additional pass if necessary.

But how do we determine which instantiation corresponds to the correct guess? The most expedient way to deal with this question is to sidestep the issue as follows. Instantiations corresponding to guesses that are too small may find it is possible to cover $\omega(\epsilon^{-2}k \log m)$ elements so we will terminate any instantiation as soon as it covers more than $O(\epsilon^{-2}k \log m)$ elements. Note that by Corollary 36 and Equation 5.1, we will not terminate the instantiation corresponding to the correct guess.

Among the instantiations that are not terminated we simply return the best solution. To find the best solution we want to estimate $|\cup_{i\in I} S_i|$, i.e., the coverage of the corresponding sets *before* the subsampling. To compute this estimate in small space we can use the F_0-*sketching* technique. For the purposes of our application, we can summarize the required result as follows:

Theorem 37 (Cormode et al. [50])**.** *There exists an* $\tilde{O}(\epsilon^{-2} \log \delta^{-1})$*-space algorithm that, given a set* $S \subseteq [n]$*, can construct a data structure* $\mathcal{M}(S)$*, called an* F_0 sketch *of* S*, that has the property that the number of distinct elements in a collection of sets* $S_1, S_2, \ldots, S_r$ *can be approximated up to a* $1+\epsilon$ *factor with probability at least* $1-\delta$ *given the collection of* F_0 *sketches* $\mathcal{M}(S_1), \mathcal{M}(S_2), \ldots, \mathcal{M}(S_r)$*.*

For the algorithms in the previous section, we can maintain a sketch $\mathcal{M}(C)$ of the set of covered elements in $\tilde{O}(\epsilon^{-2} \log \delta^{-1})$ space and from this can estimate the desired coverage. We set $\delta \leftarrow \Theta(1/n^2)$ so that coverages of all non-terminated instances are estimated up to a factor $(1+\epsilon)$ with high probability.

5.2.4 Other Algorithmic Results

In this final subsection, we briefly review some other algorithmic results for MaxSetCoverage, either with different trade-offs or for a "budgeted" version of the problem.

$(1-\epsilon)$ **approximation in one pass and $\tilde{O}(\epsilon^{-2}mk)$ space.** In the previous subsection, we gave a single-pass $1-1/e-\epsilon$ approximation using $\tilde{O}(\epsilon^{-2}m)$ space. Here we observe that if we are permitted $\tilde{O}(\epsilon^{-2}mk)$ space and unlimited post-processing time then a $1-\epsilon$ approximation can be achieved directly from the F_0 sketches.

In particular, in one pass we construct the F_0 sketches of all m sets, $\mathcal{M}(S_1), \ldots, \mathcal{M}(S_m)$ where the failure probability of the sketches is set to $\delta = 1/(nm^k)$. Thus, at the end of the stream, one can $1+\epsilon$ approximate the coverage $|S_{i_1} \cup \ldots \cup S_{i_k}|$ for each collection of k sets $S_{i_1}, \ldots, S_{i_k}$ with probability at least $1-1/(nm^k)$. Since there are at most $\binom{m}{k} \leq m^k$ collections of k sets, appealing to the union bound, we guarantee that the coverages of all of the collections of k sets are preserved up to a $1+\epsilon$ factor with probability at least $1-1/n$. The space to store the sketches is $\tilde{O}(\epsilon^{-2}mk)$.

Theorem 38. *There exists a single-pass, $\tilde{O}(\epsilon^{-2}mk)$-space algorithm that finds a $1-\epsilon$ approximation of* MaxSetCoverage *with high probability .*

In comparison to the algorithm in Theorem 30, the algorithm above is non-adaptive and hence could be used in various distributed models. It also uses less space in the case where k is much smaller than ϵ^{-1}.

$(1/2-\epsilon)$ **approximation in one pass and $\tilde{O}(\epsilon^{-3}k)$ space** We next observe that it is possible to achieve a $1/2-\epsilon$ approximation using a single pass and $\tilde{O}(\epsilon^{-3}k)$ space. Consider the following simple single-pass algorithm that uses an estimate z of OPT such that $\mathrm{OPT} \leq z \leq (1+\epsilon)\,\mathrm{OPT}$. As with previous algorithms, the basic algorithm in this section also maintains $I \subseteq [m]$, $C \subseteq [n]$ where I corresponds to the ID's of the (at most k) sets in the current solution and C is the the union of the corresponding sets. The algorithm proceeds as follows:

1. Initialize $C = \emptyset$ and $I = \emptyset$.

2. For each set S in the stream:

(a) If $|S \setminus C| \geq z/(2k)$ and $|I| < k$ then $I \leftarrow I \cup \{ID(S)\}$ and $C \leftarrow C \cup S$.

The described algorithm is a $1/2 - \epsilon$ approximation. To see this, if the solution consists of k sets, then the final solution obviously covers at least $z/2 \geq \mathrm{OPT}/2$ elements. Now we consider the case in which the collection of sets $\mathcal{S}$ chosen by the algorithm contains fewer than k sets. We define $\tilde{S} := S \setminus \mathcal{C}(\mathcal{S})$ to be the set of elements in S that are not covered by the final solution. For each set S in the optimal solution $\mathcal{O}$, if S is unpicked, then $|\tilde{S}| \leq z/(2k)$. Therefore,

$$\begin{aligned}\mathrm{OPT} = \left| \bigcup_{S \in \mathcal{O}} (S \cap \mathcal{C}(\mathcal{S})) \right| + \left| \bigcup_{S \in \mathcal{O} \setminus \mathcal{S}} \tilde{S} \right| \leq |\mathcal{C}(\mathcal{S})| + \sum_{S \in \mathcal{O} \setminus \mathcal{S}} \left| \tilde{S} \right| &\leq |\mathcal{C}(\mathcal{S})| + \frac{z}{2} \\ &\leq |\mathcal{C}(\mathcal{S})| + \frac{\mathrm{OPT}(1+\epsilon)}{2}\end{aligned}$$

and thus $|\mathcal{C}(\mathcal{S})| \geq \frac{1-\epsilon}{2}\,\mathrm{OPT}$.

We note that the above algorithm uses $O(k \log m + z \log n)$ space but we can use an argument similar to that used in Section 5.2.3 to reduce this to $\tilde{O}(\epsilon^{-3}k)$. The only difference is since we need z such that $\mathrm{OPT}' \leq z \leq (1+\epsilon)\,\mathrm{OPT}'$ we will guess v in powers of $1 + \epsilon/4$ and set $\lambda = 16c\epsilon^{-2}k \log m$. Then Eq. 5.1 becomes $(1 - \epsilon/4)\lambda \leq \mathrm{OPT}' \leq (1 + \epsilon/4)^2\lambda$ and hence $z = (1 + \epsilon/4)^2\lambda$ is a sufficiently good estimate.

Theorem 39. *There exists a single-pass, $\tilde{O}(\epsilon^{-3}k)$ space algorithm that finds a $1/2 - \epsilon$ approximation of* MaxSetCoverage *with high probability.*

Group cardinality constraint. We now consider a version of the problem where each set belongs to a group amongst ℓ disjoint groups $G_1, G_2, \ldots, G_\ell$ and we are allowed to pick at most k_1 sets from group G_1, k_2 sets from group G_2, and so on. This is also known as the partition matroid constraint. A $1 - 1/e$ approximation is possible for the offline version of this problem via linear programming [2, 131]. Furthermore, Chekuri and Kumar showed that the offline greedy algorithm is guaranteed to return a 1/2 approximation [43].

Single-pass algorithm. We first observe that by simply applying the previous $1/2-\epsilon$ approximation algorithm for each group and returning the best solution, we obtain a $(1/2-\epsilon)/\ell$ approximation. The main idea for the improved algorithm is to set a threshold for when to add a set that depends on the group to which this set belongs.

We now present an algorithm that returns a $1/(\ell+1)-\epsilon$ approximation which improves upon [37, 41] for the case $\ell = 2$. The basic algorithm maintains the sets I_i for each $i = 1, 2, \ldots, \ell$ where I_i corresponds to the IDs of the sets from group G_i that are in the current solution. Similar to previous algorithms, C is used to keep track of the current coverage. Finally, the algorithm also uses an estimate z of OPT such that $\text{OPT} \leq z \leq (1+\epsilon)\,\text{OPT}$. The detailed algorithm proceeds as follows:

1. Initialize $C = \emptyset$ and $I_i = \emptyset$ for each $i = 1, \ldots, \ell$.

2. For each set $S \in G_i$ in the stream: if

   $$|S \setminus C| \geq \frac{z}{(\ell+1)k_i} \text{ and } |I_i| < k_i,$$

   then $I_i \leftarrow I_i \cup \{ID(S)\}$ and $C \leftarrow C \cup S$.

If there exists a group G_i in which k_i sets are selected, then it is clear that the solution covers at least $z/(\ell+1)$ elements. On the other hand, suppose that for all groups G_i, fewer than k_i sets are selected. As before, $\mathcal{S}$ and $\mathcal{C}(\mathcal{S})$ are the collection of sets in the solution and their union respectively. Again, we define $\tilde{S} := S \setminus \mathcal{C}(\mathcal{S})$. Furthermore, let $\mathcal{O}_i$ denote the sets in G_i that are also in the optimal solution, i.e., $\mathcal{O}_i = \mathcal{O} \cap G_i$. We have

$$\mathrm{OPT} = \left| \bigcup_{S \in \mathcal{O}} (S \cap \mathcal{C}(\mathcal{S})) \right| + \left| \bigcup_{S \in \mathcal{O} \setminus \mathcal{S}} \tilde{S} \right| \le |\mathcal{C}(\mathcal{S})| + \sum_{i=1}^{\ell} \sum_{S \in \mathcal{O}_i \setminus \mathcal{S}} \left| \tilde{S} \right|$$

$$\le |\mathcal{C}(\mathcal{S})| + \sum_{i=1}^{\ell} \sum_{S \in \mathcal{O}_i \setminus \mathcal{S}} \frac{z}{(\ell+1)k_i} \le |\mathcal{C}(\mathcal{S})| + \frac{\ell \cdot z}{\ell+1} \ .$$

Therefore,

$$|\mathcal{C}(\mathcal{S})| \ge \mathrm{OPT} - \frac{\ell \cdot z}{\ell+1} \ge \left(\frac{1}{\ell+1} - \epsilon \right) \mathrm{OPT} \ .$$

Let $k = k_1 + k_2 + \ldots + k_\ell$. The above algorithm uses $O(k \log m + z \log n)$ space but we can use an argument similar to that used in Sections 5.2.3 and 5.2.4 to reduce this to $\tilde{O}(\epsilon^{-3}k)$. We summarize the result as the following theorem.

Theorem 40. *There exists a single-pass, $\tilde{O}(\epsilon^{-3}k)$ space algorithm that finds a $1/(\ell+1) - \epsilon$ approximation of group cardinality constraint* MaxSetCoverage *with high probability.*

We note the algorithm of [37, 41] combining with our subsampling framework in Sections 5.2.3 yields a $1/4 - \epsilon$ approximation in a single pass for matroid constraints. The algorithm above gives a better approximation for partition matroid constraint when the number of groups $\ell = 2$.

Multiple-pass algorithm. Next, we present a $1/2 - \epsilon$ approximation that uses $O(\epsilon^{-1} \log(k/\epsilon))$ passes. The idea is similar to the algorithm in Section 5.2.2 where we pick a set if its contribution is above a threshold. We decrease the threshold by a factor $(1+\epsilon)$ after each pass. The main difference is to not pick a set if that violates the group constraint. Here, we assume that $\mathrm{OPT} \le z \le 4\,\mathrm{OPT}$. The detailed algorithm proceeds as follows:

1. Initialize $C \leftarrow \emptyset$ and $I_i = \emptyset$ for each $i = 1, \ldots, \ell$.

2. For $j = 1$ to $\lceil \log_\alpha(10 \cdot k/\epsilon) \rceil$ where $\alpha = 1 + \epsilon$

    (a) Make a pass over the stream. For each set $S \in G_i$:

i. If $|S \setminus C| \geq z/\alpha^j$ and $|I_i| < k_i$, then $I_i \leftarrow I_i \cup \{ID(S)\}$ and $C \leftarrow C \cup S$.

Recall that $k = k_1 + \ldots + k_\ell$. Suppose the algorithm picks k sets $S_1, S_2, \ldots, S_k$ in that order. If the algorithm picks fewer than k sets, at the end, we could simply add dummy empty sets to the solution; thus, we can assume that the algorithm picks exactly k_i sets from each group G_i. Consider an optimal solution $\mathcal{O} = \{O_1, \ldots, O_k\}$ and a bijection $\pi : [k] \to [k]$ that satisfies the following:

- If $\pi(i) = j$, then S_i and O_j belong to the same group.
- If S_i is O_j, then $\pi(i) = j$.

When $S_i \in G_t$ was picked in the jth iteration for $j > 1$, by the second property of π, we deduce that $O_{\pi(i)}$ had not been picked in the $(j-1)$th iteration. Furthermore, the first property of π ensures that since we picked S_i in the jth iteration, we know that $O_{\pi(i)}$ was available to pick in the $(j-1)$th iteration; however, its contribution was smaller than $z/(k\alpha^{j-1})$. Let $\hat{S}_i := S_i \setminus (S_1 \cup \ldots S_{i-1})$ and $\tilde{O}_i := O_i \setminus \mathcal{C}(\mathcal{S})$.

Therefore,

$$\left|\tilde{O}_{\pi(i)}\right| < \frac{z}{\alpha^{j-1}} = \alpha \cdot \frac{z}{\alpha^j} \leq \alpha \left|\hat{S}_i\right| .$$

In the case $j = 1$, obviously, $\left|\hat{S}_i\right| \geq z/\alpha \geq \left|\tilde{O}_{\pi(i)}\right| \cdot 1/\alpha$.

Finally, in the case that S_i is a dummy set that was added at the end, then $O_{\pi(i)}$ was not picked during the last pass (even though it was available to pick). Hence $\left|\tilde{O}_{\pi(i)}\right| \leq \epsilon z/(10k)$. Suppose the algorithm picked y sets $S_1, \ldots, S_y$ that are not dummy sets. We have

$$\begin{aligned} |\mathcal{C}(\mathcal{O})| - |\mathcal{C}(\mathcal{S})| &\leq \sum_{i=1}^{y} \left|O_{\pi(i)} \setminus \mathcal{C}(\mathcal{S})\right| + \sum_{i=y+1}^{k} \left|O_{\pi(i)} \setminus \mathcal{C}(\mathcal{S})\right| \\ &\leq \sum_{i=1}^{y} \alpha \left|\hat{S}_i\right| + \epsilon \left|\mathcal{C}(\mathcal{O})\right| = (1+\epsilon) \left|\mathcal{C}(\mathcal{S})\right| + \epsilon \left|\mathcal{C}(\mathcal{O})\right| \end{aligned}$$

and so $(2+\epsilon)\,|\mathcal{C}(\mathcal{S})| \geq (1-\epsilon)\,|\mathcal{C}(\mathcal{O})|$. Therefore, $|\mathcal{C}(\mathcal{S})| \geq (1-\epsilon)\,\mathrm{OPT}/(2+\epsilon)$. Repeating the subsampling argument in Section 5.2.3, we have the following.

Theorem 41. *There exists a $\tilde{O}(\epsilon^{-2}k)$ space algorithm that finds a $1/2-\epsilon$ approximation of group cardinality constraint* MaxSetCoverage *in $O(\epsilon^{-1}\log(k/\epsilon))$ passes with high probability .*

Budgeted constraint. In this variation, each set S has a cost $w_S \in [0, L]$. The problem asks to find the collection of sets whose total cost is at most L that covers the most number of distinct elements. For $I \subseteq [n]$, we use $w(I)$ to denote $\sum_{i\in I} w_{S_i}$.

We present the algorithm assuming knowledge of an estimate z such that $\mathrm{OPT} \leq z \leq (1+\epsilon)\,\mathrm{OPT}$; this assumption can be removed by running the algorithm for guesses $1, (1+\epsilon), (1+\epsilon)^2, \ldots$ for z and returning the best solution found. The basic algorithm maintains $I \subseteq [m]$, $C \subseteq [n]$ where I corresponds to the ID's of the (at most k) sets in the current solution and C is the the union of the corresponding sets. The algorithm proceeds as follows:

1. Initialize $C = \emptyset$ and $I = \emptyset$
2. For each set S in the stream:
    (a) If
    $$|S \setminus C| \geq \frac{2z}{3} \cdot \frac{w_S}{L} \ ,$$
    then:
        i. If $w(I) + w_S > L$: Terminate and return:
        $$I \leftarrow \begin{cases} I & \text{if } |C| \geq |S| \\ \{ID(S)\} & \text{if } |C| < |S| \end{cases}$$
        ii. $I \leftarrow I \cup \{ID(S)\}$ and $C \leftarrow C \cup S$.

Lemma 42. *If the clause in line* 2*ai is never satisfied, then the algorithm returns a* $1/3-\epsilon$ *approximation.*

Proof. Suppose the collection of sets chosen by the algorithm is $\mathcal{S}$. As before, we define $\tilde{S} := S \setminus \mathcal{C}(\mathcal{S})$ to be the set of elements in S that are not covered by the final solution. For each set S in the optimal solution $\mathcal{O}$, if S is unpicked, then $|\tilde{S}| \le 2z/3 \cdot w_S/L$. Therefore,

$$\begin{aligned}\mathrm{OPT} = \left|\bigcup_{S\in\mathcal{O}} (S \cap \mathcal{C}(\mathcal{S}))\right| + \left|\bigcup_{S\in\mathcal{O}\setminus\mathcal{S}} \tilde{S}\right| \le |\mathcal{C}(\mathcal{S})| + \sum_{S\in\mathcal{O}\setminus\mathcal{S}} \left|\tilde{S}\right| &\le |\mathcal{C}(\mathcal{S})| + \frac{2z}{3} \\ &\le |\mathcal{C}(\mathcal{S})| + \frac{2\,\mathrm{OPT}(1+\epsilon)}{3}\ ,\end{aligned}$$

and thus $|\mathcal{C}(\mathcal{S})| \ge \frac{1-2\epsilon}{3}\,\mathrm{OPT}$. ☐

Lemma 43. *If the clause in line* 2*ai is satisfied at some point, then the algorithm returns a* $1/3$ *approximation.*

Proof. Suppose the clause is satisfied when the set S is being considered. Then

$$|S \setminus C| + |C| \ge \frac{2z}{3} \cdot \frac{w_S + w(I)}{L} \ge \frac{2z}{3}$$

where we used the fact that $w_S + w(I) > L$. The claim then follows immediately. ☐

The algorithm needs to store the IDs of the sets in the solution as well as the current coverage C. Therefore, it uses $\tilde{O}(m+n)$ space.

Theorem 44. *There exists a single-pass,* $\tilde{O}(\epsilon^{-1}(m+n))$*-space algorithm that finds a* $1/3-\epsilon$ *approximation of budgeted* MaxSetCoverage*.*

We note that the subsampling approach would not work for budgeted constraints since the number of sets in the optimal solution is no longer bounded by k.

5.3 Algorithms for Maximum k-Vertex Coverage

In this section, we present algorithms for the maximum k-vertex coverage problem. We present our results in terms of hypergraphs for full generality. The generalization to hypergraphs can also be thought of as a natural "hitting set" variant of maximum coverage, i.e., the stream consists of a sequence of sets and we want to pick k elements in such a way to maximize the number of sets that include a picked element.

Notation. Given a hypergraph G and a subset of nodes S, we define $\mathcal{C}_G(S)$ to be the number of edges that contain at least one node in S. Recall that the maximum k-vertex coverage problem is to approximate the maximum value of $\mathcal{C}_G(S)$ over all sets S containing k nodes. We use E_G and V_G to denote the set of edges and nodes of the hypergraph G respectively.

The size of a cut $(S, V \setminus S)$ in a hypergraph G, denoted as $\delta_G(S)$, is defined as the number of hyperedges that contain at least one node in both S and $V \setminus S$. In the case that G is weighted, $\delta_G(S)$ denotes the total weight of the cut. A core idea to our approach is to use *hypergraph sparsification*:

Definition 45 (ϵ-sparsifier)**.** *Given a hypergraph $G = (V, E)$, we say that a weighted subgraph $H = (V, E')$ is an ϵ-sparsifier for G if for all $S \subseteq V$, $(1-\epsilon)\delta_G(S) \leq \delta_H(S) \leq (1+\epsilon)\delta_G(S)$.*

Any graph on N nodes has an ϵ-sparsifier with only $\tilde{O}(\epsilon^{-2}N)$ edges [130]. Similarly, any hypergraph in which the maximum size of the hyperedges is bounded by d (rank d hypergraphs) has an ϵ-sparsifier with only $\tilde{O}(\epsilon^{-2}dN)$ edges. Furthermore, an ϵ-sparsifier can be constructed in the dynamic graph stream model using one pass and $\tilde{O}(\epsilon^{-2}dN)$ space [74, 91].

First, we show that it is possible to approximate all the coverages by constructing a sparsifier of a slightly modified graph. In particular, we construct the sparsifier H of the graph G' with an extra node v, i.e., $V_{G'} = V_G \cup \{v\}$, and for every hyperedge

$e \in E_G$, we put the hyperedge $e \cup \{v\}$ in $E_{G'}$. It is easy to see that for all S that is a subset of V_G, we have $\mathcal{C}_G(S) = \delta_{G'}(S)$. Therefore, it is immediate that we could $1+\epsilon$ approximate all the coverages in G by constructing the sparsifer of G'.

Theorem 46. *There exists a single-pass, $\tilde{O}(\epsilon^{-2}dN)$-space algorithm that finds a $1-\epsilon$ approximation of* MaxVertexCoverage *of rank d hypergraphs with high probability.*

The above theorem assumes unbounded post-processing time. If k is constant, the post-processing will be polynomial. For larger k, if we still require polynomial running time then, after constructing the ϵ-sparsifier H, we could either use the $(1-(1-1/d)^d)$ approximation algorithm via linear programming [1] or the folklore $(1-1/e)$ approximation greedy algorithm.

Algorithm for near-regular hypergraphs. In this subsection, we show that it is possible to reduce the space used to $\tilde{O}(\epsilon^{-3}dk)$ in the case of hypergraphs that are regular or nearly regular. Define $\kappa \le 1$ to be the ratio between the smallest degree and the largest degree; for a regular hypergraph $\kappa = 1$. We show that a $(\kappa - \epsilon)$ approximation is possible using $\tilde{O}(\epsilon^{-3}dk)$ space for rank d hypergraphs. This also implies a $(1-\epsilon)$ approximation for regular hypergraphs.

Theorem 47. *There exists a single-pass, $\tilde{O}(\epsilon^{-3}dk)$-space algorithm that finds a $(\kappa - \epsilon)$ approximation of* MaxVertexCoverage *of hypergraphs of rank d with high probability.*

Proof. Let t_1 and t_2 be the minimum and maximum degree of a node in G. Suppose we uniformly sample a set S of k nodes. Let $L_S(y) = \max(0, |y \cap S| - 1)$. Then the coverage of S satisfies

$$\mathcal{C}_G(S) = \sum_{y \in E_G} I[S \cap y \neq \emptyset] = \sum_{y \in E_G} (|S \cap y| - L_S(y)) \geq kt_1 - \sum_{y \in E_G} L_S(y) \ .$$

where the last inequality follows since every node in S covers at least t_1 hyperedges.

Let $\xi_y(j)$ denote the event that j nodes in the hyperedge y are in S and let $|y|$ denote the number of nodes in y. We have

$$\mathrm{E}\left[L_S(y)\right] = \sum_{j=1}^{|y|}(j-1)\Pr\left[\xi_y(j)\right] = \left(\sum_{j=0}^{|y|} j\Pr\left[\xi_y(j)\right]\right) - 1 + \Pr\left[\xi_y(0)\right] \ .$$

The sum $\sum_{j=0}^{|y|} j\Pr\left[\xi_y(j)\right]$ is the expected value of the hypergeometric distribution and therefore it evaluates to $|y|k/N$. Furthermore,

$$\begin{aligned}\Pr\left[\xi_y(0)\right] &= \prod_{i=0}^{k-1}\left(1-\frac{|y|}{N-i}\right) \le \left(1-\frac{|y|}{N}\right)^k \\ &\le \exp\left(-\frac{k|y|}{N}\right) \le 1-\frac{k|y|}{N}+\frac{1}{2}\left(\frac{k|y|}{N}\right)^2 .\end{aligned}$$

The last inequality follows from taking the first three terms of the Taylor's expansion. Hence,

$$\mathrm{E}\left[L_S(y)\right] \le \frac{k|y|}{N} - 1 + 1 - \frac{k|y|}{N} + \frac{1}{2}\left(\frac{k|y|}{N}\right)^2 = \frac{1}{2}\left(\frac{k|y|}{N}\right)^2 .$$

Hence, if $N \ge 4kd/\epsilon$, then

$$\begin{aligned}\sum_{y\in E_G}\mathrm{E}\left[L_S(y)\right] &\le \frac{1}{2}\sum_{y\in E_G}\left(\frac{k|y|}{N}\right)^2 \le \frac{1}{2}d\left(\frac{k}{N}\right)^2\sum_{y\in E_G}|y| \le \frac{1}{2}d\left(\frac{k}{N}\right)^2 Nt_2 = \frac{1}{2}d\frac{k^2}{N}t_2 \\ &\le \frac{1}{8}\epsilon k t_2 \ .\end{aligned}$$

By an application of Markov's inequality,

$$\Pr\left[\sum_{y\in E_G} L_S(y) \ge \epsilon k t_2\right] \le 1/8 \ .$$

Thus, if we sample $O(\log N)$ sets of k nodes in parallel, with high probability, there is a sample set S of k nodes satisfying $\sum_{y\in E_G} L_S(y) \le \epsilon k t_2$ which implies that

$\mathcal{C}_G(S) \geq kt_1 - \epsilon kt_2 \geq (\kappa - \epsilon)kt_2 \geq (\kappa - \epsilon)\,\mathrm{OPT}$. If $N \leq 4kd/\epsilon$, we simply construct the sparsifier of G' as described above to achieve a $1-\epsilon$ approximation. □

5.4 Lower Bounds

In this section, we prove space lower bounds for data stream algorithms that approximate MaxSetCoverage or MaxVertexCoverage. In particular, these imply that improving over an $(1-1/e)$ approximation of MaxSetCoverage with constant passes and constant k requires $\Omega(m)$ space. Recall that, still assuming k is constant, we designed a constant-pass algorithm that returned a $(1 - 1/e - \epsilon)$ approximation using $\tilde{O}(\epsilon^{-2}k)$ space. For constant k, we also show that improving over a κ approximation (where κ is the ratio between the lowest degree and the highest degree) for MaxVertexCoverage requires $\Omega(N\kappa^3)$ space. Our algorithm returned a $\kappa - \epsilon$ approximation using $\tilde{O}(\epsilon^{-3}k)$ space.

Approach. We prove both bounds by a reduction from r-player set-disjointness in communication complexity. In this problem, there are r players where the ith player has a set $S_i \subseteq [u]$. It is promised that exactly one of the following two cases happens.

- Case 1 (NO instance): All the sets are pairwise disjoint.
- Case 2 (YES instance): There is a unique element $e \in [u]$ such that $e \in S_i$ for all $i \in [r]$.

The goal of the communication problem is the rth player answers whether the input is a YES instance or a NO instance correctly with probability at least 0.9. We shall denote this problem by $\mathsf{DISJ}_r(u)$.

The communication complexity of the above problem in p-round, one-way model (where each round consists of player 1 sending a message to player 2, then player 2 sending a message to player 3 and so on) is $\Omega(u/r)$ [38] even if the players may use public randomness. This implies that in any randomized communication protocol, the maximum message sent by a player contains $\Omega(u/(pr^2))$ bits. Without loss of generality, we will assume that $|S_1 \cup S_2 \cup \ldots \cup S_r| \geq u/4$ via a padding argument.

Theorem 48. *Assuming $n = \Omega(\epsilon^{-2} k \log m)$, any constant-pass algorithm that finds a $(1+\epsilon)(1-(1-1/k)^k)$ approximation of* MaxSetCoverage *with probability at least 0.99 requires $\Omega(m/k^2)$ space even when all the sets have the same size.*

Proof. Our proof is a reduction from $\mathsf{DISJ}_k(m)$. Consider a sufficiently large n where k divides n. For each $i \in [m]$, let $\mathcal{P}_i$ be a random partition of $[n]$ into k sets $V_1^i, \ldots, V_k^i$ of equal size. Each partition is chosen independently and the players agree on these partitions using public randomness before receiving the input.

For each player j, if $i \in S_j$, then she puts V_j^i in the stream. According to the aforementioned assumption, the stream consists of $\Theta(m)$ sets.

If the input is a NO instance, then for each $i \in [m]$, there is at most one set V_j^i in the stream. Hence, the stream consists of independent random sets of size n/k. Therefore, for each $e \in [n]$ and any k sets $V_{j_1}^{i_1}, \ldots, V_{j_k}^{i_k}$ in the stream, $\Pr\left[e \in V_{j_1}^{i_1} \cup \ldots \cup V_{j_k}^{i_k}\right] = 1-(1-1/k)^k$. By an application of Chernoff bound for negatively correlated boolean random variables [123],

$$\begin{aligned}
&\Pr\left[\left||V_{j_1}^{i_1} \cup \ldots \cup V_{j_k}^{i_k}| - \left(1-\left(1-\frac{1}{k}\right)^k\right)n\right| > \epsilon\left(1-\left(1-\frac{1}{k}\right)^k\right)n\right] \\
&\leq 3\exp\left(-\epsilon^2\left(1-\left(1-\frac{1}{k}\right)^k\right)\frac{n}{3}\right) \\
&\leq 3\exp\left(-\epsilon^2(1-1/e)n/3\right) \leq \frac{1}{m^{10+k}} \ .
\end{aligned}$$

The last inequality holds when n is a sufficiently large multiple of $k\epsilon^{-2}\log m$. Therefore, the maximum coverage in this case is at most $(1+\epsilon)(1-(1-1/k)^k)n$ with probability at least $1-1/m^{10}$ by taking the union bound over all $\binom{m}{k} \leq m^k$ possible k sets.

If the input is a YES instance, then clearly, the maximum coverage is n. This is because there exists $i \in [m]$ such that $i \in S_1 \cap \ldots \cap S_k$ and therefore $V_1^i, \ldots, V_k^i$ are in the stream.

Therefore, any constant pass and $O(s)$-space algorithm that finds a $(1+2\epsilon)(1-(1-1/k)^k)$ approximation of the maximum coverage with probability at least 0.99 implies a protocol to solve the k-party disjointness problem using $O(s)$ bits of communication. Thus, $s = \Omega(m/k^2)$ as required. $\square$

Consider the sets $S_1, \ldots, S_r \subseteq [u]$ that satisfy the unique intersection promise as in $\mathsf{DISJ}_r(u)$. Let X be the r by u matrix in which the row X_i is the characteristic vector of S_i. Suppose there are $r' = \Omega(r^2)$ players. Chakrabarti et al. [36] showed that if each entry of X is given to a unique player and the order in which the entries are given to the players is random, then the players need to use $\Omega(u/r)$ bits of communication to tell whether the sets is a YES instance or a NO instance with probability at least 0.9. Thus, in any randomized protocol, the maximum message sent by a player contains $\Omega(u/r^3)$ bits. Hence, using the same reduction and assuming constant k, we show that the same lower bound holds even for random order stream.

Theorem 49. *Assuming $n = \Omega(\epsilon^{-2} k \log m)$, any constant-pass algorithm that finds a $(1+\epsilon)(1-(1-1/k)^k)$ approximation of* MaxSetCoverage *with probability at least 0.99 requires $\Omega(m/k^3)$ space even when all the sets have the same size and arrive in random order.*

Next, we prove a lower bound for the k-vertex coverage problem for graphs where the ratio between the minimum degree and the maximum degree is at least κ. We show that for constant k, beating κ approximation for constant κ requires $\Omega(N)$ space.

Since κ can be smaller than any constant, this also establishes that $\Omega(N)$ space is required for any constant approximation of MaxVertexCoverage.

Theorem 50. *For $\epsilon > 0$, any constant-pass algorithm that finds a $(\kappa+\epsilon)$ approximation of* $\mathsf{MaxVertexCoverage}$ *with probability at least 0.99 requires $\Omega(N\kappa^3/k)$ space.*

Proof. Initially, assume $k = 1$. We consider the multi-party set disjointness problem $\mathsf{DISJ}_t(N')$ where $t = 1/\kappa$ and $N' = N/t$. Here, there are t players and the input sets are subsets of $[N']$. We consider a bipartite graph where the set of possible nodes are $L \cup R$ where $L = \{u_i\}_{i \in [N']}$ and $R = \{v_{i,j}\}_{i \in [N'], j \in [t]}$. Note that this graph has $(t+1)N' = \Theta(N)$ nodes. However we only consider a node to exist if the stream contains an edge incident to that node.

The j-th player defines a set of edges on this graph based on their set S_j as follows. If $i \in S_j$, she puts the edge between u_i and $v_{i,j}$. If $S_1, \ldots, S_t$ is a YES instance, then there must be a node u_i that has degree t. If the input is a NO instance, then every node in the graph has degree at most 1. Hence the ratio of minimum degree to maximum degree is at least $1/t = \kappa$ as required.

Thus, for $k = 1$, a $1/t$ approximation with probability at least 0.99 on a graph of N nodes implies a protocol to solve $\mathsf{DISJ}_t(N')$. Therefore, the algorithm requires $\Omega(N\kappa^3)$ space. For general k, we make k copies of the above construction to deduce the lower bound $\Omega(N\kappa^3/k)$. $\square$

CHAPTER 6

FINDING CAPACITATED MAXCUT IN THE ADJACENCY LIST MODEL

6.1 Introduction and Related Work

Capacitated MaxCut. We consider the streaming capacitated max $(t+1)$-cut problem where t parts are bounded. This problem and its variations have been studied in various work [2, 65, 68, 142]. We consider a slightly more special case of the problem, denoted by $\mathsf{MaxCut}(k)$ in which the goal is to find t disjoint sets of nodes $S_1, S_2, \ldots, S_t$ such that $|S_i| \leq k$ and the number of edges across the parts is maximized. The optimal solution is defined as follows.

$$\mathrm{OPT} := \underset{\substack{\text{disjoint } S_1,\ldots,S_t \subset V \\ |S_i| \leq k}}{\arg\max} \left| \{(u,v) \in E : |\{u,v\} \cap S_i| = 1\} \right| .$$

We consider the streaming setting in which the stream is a concatenation of the adjacency lists. Our goal is to design algorithms, with constant approximations, that use space depending only on k. We note that the algorithm only needs to output the t bounded parts and the unbounded part is implicit. Therefore, we will refer to the solution as the bounded parts that the algorithm outputs.

Motivation. For this problem, Ageev and Sviridenko [2] presented a 1/2 approximation using linear programming. Gaur et al. [65] subsequently showed that local search obtains a $1 - 1/t$ approximation. Their algorithms also work for the more general constraint $|S_i| \leq k_i$. This problem is motivated by some applications such as:

- Placement of commercials where each of the bounded parts represents a commercial break with a time constraint. The nodes in the graphs represent a large pool of commercials and an edge between two commercials indicates competing products. This formulation tries to avoid competing products to be placed in the same break [65, 68]. See [68] for a detailed experimental study.

- Finding small sets of items for recommendation systems where each item is a node and an edge between two nodes represents that the corresponding items are often not bought together [65].

- Design of product modules where each node represents a component of a product and an edge connects a pair of nodes if the corresponding pair of components do not interact with each other. We want to maximize the number of non-interacting components are placed in different modules. The capacity restriction arises because of the need to construct balanced modules [65].

- Clustering of data in which edges specify dissimilarities and we know that one or more parts have bounded sizes. One example application is clustering in protein interaction networks [47]. This also applies to imbalanced classification and semi-supervised learning problems.

Other related problems such as MaxCut and MaxBisection have been extensively studied the RAM model [63, 71, 86, 135] using semidefinite programming, spectral and combinatorial methods. In streaming models and distributed models, MaxCut has been studied in [35, 90, 97]. In the MaxCut problem, the goal is to partition the vertices into two parts that maximize the cut across. In the MaxBisection problem, the constraint is that the two parts have the same size.

We also note that for $t = 1$, $\mathsf{MaxCut}(k)$ is a special case of non-monotone submodular maximization which is an important topic in algorithm design, both in the RAM and streaming models.

Graph stream model. We study this problem in the streaming setting. For undirected graphs, we consider the *adjacency list model* in which the stream is a concatenation of the adjacency lists. Many other graph problems have been studied in this model. Some examples include triangle counting, maximum matching, maximum coverage and set cover [13, 14, 29, 77, 100, 113, 114]. Furthermore, streaming graph algorithms is a major research topic to handle large graphs. See [111] for a recent survey.

The capacitated $\mathsf{MaxCut}(k)$ problem can also be viewed as a parameterized streaming problem. For example, under the assumption that one side of the MaxCut has at most k nodes, our algorithm allows us to approximate the MaxCut using space depending only on k. Similar parameterized streaming algorithms have been studied in the context of maximum matching and minimum vertex cover [45, 46].

We note that in the arbitrary order model, $\mathsf{MaxCut}(1)$ is the same as identifying the node with highest degree which requires $\Omega(n)$ space for a constant approximation (see [116]). In this model, streaming spectral sparsifiers [74, 91] obtain an upper bound $\Theta(\epsilon^{-2}n)$ for a $1-\epsilon$ approximation assuming unbounded running time.

As mentioned earlier, $\mathsf{MaxCut}(k)$ is a special case of non-monotone submodular maximization under a cardinality constraint. In the RAM setting, an incomplete list of works related to submodular maximization includes [18, 30, 42, 59, 66, 75]. In the streaming setting, submodular maximization has also received a lot of recent attention [17, 37, 41, 61, 116].

Our results. We make several contributions. Our main results are as follows.

1. For the case $t = 1$, we present a single-pass, $\tilde{O}(k^2)$-space algorithm that finds a $0.4 - o(1)$ approximation. We also present a two-pass, $\tilde{O}(k^3/\epsilon)$-space algorithm that finds a $0.5 - \epsilon$ approximation.

2. For the case $t > 1$, we present a single-pass $\tilde{O}((tk)^2)$-space algorithm that finds a $6/11 - o(1)$ approximation.

Motivated by the case $t = 1$, we design new algorithms for non-monotone submodular maximization under a cardinality constraint. We show that by using more passes or more space, we can obtain a better approximation than the algorithm given by Chekuri et al. [41].

6.2 Algorithms for Capacitated MaxCut

Preliminaries. Let $N(S) := \{v \in V \setminus S : (u, v) \in E \text{ for some } u \in S\}$ denote the set of neighbors of S. We use $E(S) = \{(u, v) \in E : u \in S \text{ and } v \in S\}$ to denote the set of internal edges of S and $E^*(S) = \{(u, v) \in E : u \in S \text{ or } v \in S\}$ to denote the set of edges that have at least one end point in S. For convenience, we also use the standard notation $\overline{S} := V \setminus S$. For two disjoint sets of nodes A and B, the cut between A and B is defined as

$$\text{Cut}(A, B) := \{(u, v) \in E : u \in A \text{ and } v \in B\} .$$

We define $\delta(u, A) := \{(u, v) \in E : v \in A\} = \text{Cut}(\{u\}, A \setminus \{u\})$ and for convenience, let $f(S) := \left|\text{Cut}(S, \overline{S})\right|$. We note that f is submodular and we use the following standard notion of marginal gain

$$f(v \mid S) = f(S \cup \{v\}) - f(S) .$$

Algorithms for the case $t = 1$. We first develop algorithms for $\mathsf{MaxCut}(k)$ for the case $t = 1$. We focus solely on the space, pass complexity, and the approximation factor. We use $\mathcal{O}$ to denote the bounded part of the optimal solution, i.e., $\mathcal{O}$ is the set of at most k nodes that maximizes $f(\mathcal{O})$.

Single-pass algorithm. We observe that a random bipartition of the set of k nodes with highest degrees is a 0.25 approximation in expectation. This observation could easily be translated to a single-pass and $\tilde{O}(k^2)$-space algorithm.

We now present a better single-pass algorithm with a $0.4 - o(1)$ approximation. The algorithm stores the $2k$ nodes with highest degrees S and finds the best solution (of at most k nodes) T in S. To find the best solution in S, it is sufficient to store the internal edges $E(S)$ and for each node $v \in S$, we store the degree of v. The algorithm can be easily implemented in the adjacency list model with appropriate bookkeeping.

The space to maintain $\deg(v)$ for all nodes $v \in S$ is $\tilde{O}(k)$. Furthermore, maintaining $E(S)$ requires $\tilde{O}(k^2)$ space and therefore the algorithm uses $\tilde{O}(k^2)$ space overall.

Theorem 51. *For $t = 1$, there exists a single-pass and $\tilde{O}(k^2)$-space algorithm that finds a $0.4 - o(1)$ approximation of $\mathsf{MaxCut}(k)$ in the adjacency list model.*

Proof. Consider the described algorithm. Let the $2k$ highest degrees be $d_1 \geq d_2 \geq \ldots \geq d_{2k}$ and let

$$d^* := \frac{d_1 + \ldots + d_k}{k} \quad \text{and} \quad d' := \frac{d_{k+1} + \ldots + d_{2k}}{k} \ .$$

First, we observe that for each node $o \in \mathcal{O} \setminus T$, we have that $\deg(o) \leq d_{2k} \leq d'$. Hence,

$$f(\mathcal{O}) \leq f(\mathcal{O} \cap S) + \sum_{o \in \mathcal{O} \setminus T} \deg(o) \leq f(T) + kd' \ .$$

Thus, if $kd' < 0.6 f(\mathcal{O})$, then it is immediate that $f(T) \geq 0.4 f(\mathcal{O})$.

Now we consider the case $kd' \geq 0.6 f(\mathcal{O})$. We note that there are at least $k(d^* + d')/2$ edges in $E^*(S)$. If we randomly partition S into two sets of k nodes S_1 and S_2, we can see that

$$
\begin{aligned}
\mathrm{E}\left[f(S_1)\right] &= \sum_{v \in S, u \in N(v)} \mathrm{E}\left[I[v \in S_1 \text{ and } u \notin S_1]\right] \\
&\geq \sum_{v \in S} \frac{\deg(v)(k-1)}{2(2k-1)} \\
&= \left(\frac{1}{4} - \frac{1}{4(2k-1)}\right)(kd^* + kd') \ .
\end{aligned}
$$

Therefore, combining with the fact that $kd^* \geq f(\mathcal{O})$, we deduce that $f(T) \geq (1/4 - \Theta(1/k)) \cdot (kd^* + kd') \geq (1/4 - \Theta(1/k)) \cdot 1.6 \cdot f(\mathcal{O}) \geq (0.4 - \Theta(1/k))f(\mathcal{O})$. □

Remark. The above algorithm above gives a deterministic $0.4 - o(1)$ approximation whereas the generic streaming algorithm for submodular maximization given by Chekuri et al. in [41] obtains a $1/3 - \epsilon$ approximation only in expectation.

Two-pass algorithm. We now present a two-pass, $\tilde{O}(k^3/\epsilon)$-space algorithm with a 0.5 approximation. We rely on the following framework that allows us to focus on the case $f(\mathcal{O}) \leq O(k^2)$. Suppose γ is some constant. We observe that if $f(\mathcal{O}) > k^2/\gamma$, in one pass and $\tilde{O}(k)$ space, we can output the k nodes with highest degrees as the solution T to obtain a $1 - \gamma$ approximation. We refer to this as the naive algorithm. To see that this is a $1 - \gamma$ approximation given $f(\mathcal{O}) > k^2/\gamma$, we observe

$$
f(T) \geq \sum_{v \in T} \deg(v) - \sum_{u,v \in T} I[(u,v) \in E] \geq f(\mathcal{O}) - k^2 \geq (1-\gamma) f(\mathcal{O}) \ .
$$

Given $f(\mathcal{O}) \leq k^2/\gamma$, suppose we can design a χ-space core algorithm that is a $1 - \gamma$ approximation. If this assumption fails to hold, i.e., $f(\mathcal{O}) > k^2/\gamma$, the core algorithm may use more than χ space. In this case, we simply terminate the core algorithm and return the solution given by the naive algorithm. Otherwise, we return the better solution given by the two algorithms.

Since we need to store the solution, $\chi = \tilde{\Omega}(k)$. By combining the naive algorithm and the core algorithm as described, we get a χ-space algorithm that is a $1 - \gamma$ approximation.

Following the aforementioned framework, we only need to consider the case $f(\mathcal{O}) \leq 2k^2$. Assuming we have a guess z such that $f(\mathcal{O}) \leq z \leq (1+\epsilon)f(\mathcal{O})$, the core algorithm proceeds as follows.

1. First pass: When process the adjacency list of a node v, if $f(v \mid S) \geq z/(2k)$ and $|S| < k$, then $S \leftarrow S \cup \{u\}$. Maintain $N(S)$ in parallel.
2. If $|S| = k$, return S as the final solution T. Otherwise, take a second pass to find all edges in $E^*(S \cup N(S))$.
3. Using $E^*(S \cup N(S))$, return the best solution $A \subseteq S \cup N(S)$ where $|A| \leq k$ as the final solution T.

Note that the space to maintain $N(S)$ is $\tilde{O}(k^2)$ since we assume $\text{OPT} \leq 2k^2$.

Theorem 52. *For $t = 1$, there exists a two-pass, $\tilde{O}(k^3/\epsilon)$-space algorithm that finds a $1/2 - \epsilon$ approximation of* $\mathsf{MaxCut}(k)$ *in the adjacency list model.*

Proof. Consider the described algorithm. We first show that the space to maintain $E^*(S \cup N(S))$ is $\tilde{O}(k^3/\epsilon)$ if the second pass is needed, i.e., if the algorithm picks fewer than k nodes in the first pass. Consider S at the end of the first pass. If $|S| < k$, by submodularity, all unpicked nodes $v \notin S$ satisfy

$$f(v \mid S) = \left|\delta(v, \overline{S})\right| - |\delta(S, v)| \leq \frac{z}{2k} < \frac{f(\mathcal{O})}{k} \leq 2k$$

and so $\left|\delta(v, \overline{S})\right| < 2k + |\delta(S, v)|$.

The last inequality follows from the assumption $f(\mathcal{O}) \leq 2k^2$. Furthermore, we already argued that $f(\mathcal{O}) \leq 2k^2$ implies $|N(S)| \leq 2k^2$. Thus,

$$\sum_{v \in N(S)} \left|\delta(v, \overline{S})\right| \leq 2k^2 \cdot 2k + \sum_{v \in N(S)} |\delta(S, v)| \ .$$

This allows us to conclude that $|E^*(N(S))| \leq 4k^3 + 2k^2 \leq O(k^3)$. Finally, we have $|E(S)| \leq k^2$ and therefore

$$|E^*(S \cup N(S))| = |E(S)| + |E^*(N(S))| \leq O(k^3) \ .$$

Running this algorithm on $O(\epsilon^{-1} \log(k/\epsilon))$ different guesses entails $\tilde{O}(k^3/\epsilon)$ memory in total.

Suppose the algorithm picks k nodes in the first pass, then clearly $f(T) \geq z/2 \geq 1/2 \cdot f(\mathcal{O})$. Now, suppose that in the first pass, the algorithm picks fewer than k nodes. We observe that for all $o \notin S \cup N(S)$, by construction, o is not connected to any node in S. Then, we know that the $\deg(o) \leq z/(2k)$ since otherwise, the algorithm would pick o in the first pass. Let $U = S \cup N(S)$, we have

$$f(\mathcal{O}) \leq f(\mathcal{O} \cap U) + \sum_{o \in \mathcal{O} \setminus U} \deg(o) \leq f(T) + \frac{z}{2}$$

which implies $f(T) \geq (1/2 - \epsilon) \cdot f(\mathcal{O})$. □

Algorithm for the case $t > 1$. We generalize the algorithm for $t = 1$. The algorithm identifies $2kt$ nodes S with highest degrees and then find the best solution in S. Particularly, the best solution in S is t disjoint parts, formed by nodes in S, of sizes at most k that maximize the multicut.

More formally, let g to denote the size of the multicut,

$$g(S_1, \ldots, S_t) := \left| \{(u,v) \in E : |\{u,v\} \cap S_i| = 1\} \right| .$$

We want to find t disjoint parts $T_1, \ldots, T_t$ from the nodes in S such that $|T_i| \leq k$ and $g(T_1, \ldots, T_t)$ is maximized as the solution T. We again note that to find T, it is sufficient to maintain $E(S)$ and $\left|\delta(v, \overline{S})\right|$ for each $v \in S$. This requires $\tilde{O}(t^2k^2)$ space.

Let $\mathcal{O}_1, \ldots, \mathcal{O}_t$ denote the bounded parts in the optimal solution and let $\mathcal{O} = \cup_{i=1}^{t} O_i$ and note that $|\mathcal{O}| \leq tk$.

Theorem 53. *As $t \to \infty$, there exists a single-pass and $\tilde{O}(t^2k^2)$-space algorithm that finds a $6/11 - o(1)$ approximation of* $\mathsf{MaxCut}(k)$ *in the adjacency list model.*

Proof. Consider the described algorithm. Again, let the highest degrees be $d_1 \geq d_2 \geq \ldots \geq d_{2tk}$. Define

$$d^* := \frac{d_1 + \ldots + d_{tk}}{tk} \quad \text{and} \quad d' := \frac{d_{tk+1} + \ldots + d_{2tk}}{tk} .$$

For all $o \in \mathcal{O} \setminus S$, we have that $\deg(o) \leq d'$. Let $\mathcal{O}'_i = \mathcal{O}_i \cap S$. We have,

$$\mathrm{OPT} \leq g(\mathcal{O}'_1, \ldots, \mathcal{O}'_t) + \sum_{o \in \mathcal{O} \setminus T} \deg(o) \leq g(T_1, \ldots, T_t) + tkd' .$$

The second inequality follows from the definition of T and $|\mathcal{O}| \leq tk$. Thus, if

$$tkd' \leq \frac{3/4 - 1/8 \cdot (1 - 1/t)}{5/4 + 1/8 \cdot (1 - 1/t)} \,\mathrm{OPT} := \gamma \,\mathrm{OPT} \ ,$$

then $g(T_1, \ldots, T_t) \geq (1 - \gamma)\,\mathrm{OPT}$. It is straightforward to check that

$$1 - \gamma = \frac{1/2 + 1/4 \cdot (1 - 1/t)}{5/4 + 1/8 \cdot (1 - 1/t)} .$$

On ther other hand, if $tkd' > \gamma$, we again use the probabilistic argument where we pick t disjoint parts of size k randomly as the solution $T'_1, \ldots, T'_t$. First, there are at least $tk(d^* + d')/2$ edges in $E^*(S)$. It is easy to see that each edge belongs to the solution multicut with probability at least $1/2 + 1/4 \cdot (1 - 1/t)$.

Therefore,

$$\begin{aligned}
\mathrm{E}\left[g(T'_1, \ldots, T'_t)\right] &\geq \frac{tkd^* + tkd'}{2}\left(\frac{1}{2} + \frac{1}{4} \cdot \left(1 - \frac{1}{t}\right)\right) \\
&\geq \frac{\mathrm{OPT} + tkd'}{2}\left(\frac{1}{2} + \frac{1}{4} \cdot \left(1 - \frac{1}{t}\right)\right) \\
&\geq \frac{1/2 + 1/4 \cdot (1 - 1/t)}{5/4 + 1/8 \cdot (1 - 1/t)} \mathrm{OPT} = (1 - \gamma)\,\mathrm{OPT} \ .
\end{aligned}$$

But $g(T) \geq g(T')$ and we therefore have a $1 - \gamma$ approximation. If we substitute $t = 1$, we get a 0.4 approximation as expected. If we let $t \to \infty$, then the approximation approaches 6/11. □

Remark. We note that the above algorithm obtains a better approximation than that of the algorithm given by Ageev and Sviridenko [2]. However, it uses exponential time in terms of t and k.

6.3 New Algorithms for Non-monotone Submodular Maximization

In this section, we introduce a tool for non-monotone submodular maximization. In this model, the stream consists of m items $\mathcal{N} = \{u_1, u_2, \ldots, u_m\}$ and an oracle is available to evaluate a submodular function $f : 2^{\mathcal{N}} \to [1, \Delta]$. A function $f : 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ is submodular if for all A and B that are subsets of $\mathcal{N}$ where $A \subseteq B$, if $u \notin B$, then $f(B \cup \{u\}) - f(B) \leq f(A \cup \{u\}) - f(A)$. The goal is to pick a set of at most k items S such that $f(S)$ is maximized. For simplicity, we assume that $|\mathcal{O}| = k$ by padding dummy items with zero marginal gain to the stream. Formally,

$$\mathcal{O} = \underset{S \subset \mathcal{N} : |S| \leq k}{\arg\max}\, f(S) \ .$$

We refer to the above problem as $\mathsf{SubmodularMax}(k)$. Next, we state some useful lemmas for our analysis.

Lemma 54. *If the function $f : 2^{\mathcal{N}} \to \mathbb{R}$ is submodular, then the function $g : 2^{\mathcal{N} \setminus V} \to \mathbb{R}$ where $g(S) = f(S \cup V)$ is submodular.*

Proof. Consider arbitrary sets $A \subseteq B \subset \mathcal{N} \setminus V$ and $u \notin B$. Since $V \cup A \subseteq V \cup B$ and f is submodular, we have $f(u \mid V \cup B) \le f(u \mid V \cup A)$ which is equivalent to $g(u \mid B) \le g(u \mid A)$. $\square$

The following claims argue that that if an items outside $\mathcal{O}$ appears in the solution T with probability at most p, then $\mathrm{E}\left[f(\mathcal{O} \cup T)\right] \ge (1-p) f(\mathcal{O})$.

Lemma 55 (Buchbinder et al. [30])**.** *Let $A \subseteq \mathcal{N}$ be a random subset of $\mathcal{N}$ such that an item is in A with probability at most p. If $f : \mathcal{N} \to \mathbb{R}$ is submodular, we have $\mathrm{E}\left[f(A)\right] \ge (1-p) f(\emptyset)$.*

We deduce the following corollary by appealing to Lemma 54 and Lemma 55.

Corollary 56. *Let $A \subseteq \mathcal{N} \setminus V$ be a random subset of $\mathcal{N} \setminus V$ such that an item is in A with probability at most p. If $f : 2^{\mathcal{N}} \to \mathbb{R}$ is submodular and $g : 2^{\mathcal{N} \setminus V} \to \mathbb{R}$ where $g(S) = f(S \cup V)$, then $\mathrm{E}\left[f(A \cup V)\right] = \mathrm{E}\left[g(A)\right] \ge (1-p) g(\emptyset) = (1-p) f(V)$.*

Random coloring scheme. We use a $(k+1)$-wise hash function $h : [m] \to [k/\epsilon]$ to randomly color the stream items among $[k/\epsilon]$ colors. We use the notation $h(\mathcal{O}) := \{c : h(o_i) = c \text{ for some } o_i\}$ to denote the set of colors of items in $\mathcal{O} = \{o_1, \ldots, o_k\}$. We could try to guess a set of colors C with the hope that $C = h(\mathcal{O})$ and run some algorithm of choice on the items with the guessed colors, i.e., $\{u : h(u) \in C\}$. With the correct guess $C = h(\mathcal{O})$, the algorithm considers all items in $\mathcal{O}$ while other items are considered with probability at most ϵ.

The important observation is that because h is $(k+1)$-wise independent, for all $u \notin \mathcal{O}$, we have $\Pr\left[h(u) \in D \mid h(\mathcal{O}) = D\right] \le \epsilon$.

Single-pass algorithm. We are now ready to describe a single-pass algorithm that returns an expected $1/2-\epsilon$ approximation of $\mathsf{SubmodularMax}(k)$. This algorithm is a combination of our "coloring scheme" and the work of Badadiniyuru et al. [17]. The algorithm proceeds as follows.

1. Guess z such that $f(\mathcal{O}) \le z \le (1+\epsilon)f(\mathcal{O})$ and guess $C = h(\mathcal{O})$.
2. For each stream item u: if $|S| < k$ and $h(u) \in C$ and $f(u \mid S) \ge z/(2k)$, then $S \leftarrow S \cup \{u\}$.
3. Return S as the final solution T.

Lemma 57. *If $|T| < k$, then $f(T) \ge f(T \cup \mathcal{O}) - (1+\epsilon)/2 \cdot f(\mathcal{O})$.*

Proof. By submodularity and the algorithm's logic, every unpicked item u satisfies $f(u \mid T) \le z/(2k)$. Hence, $f(T\cup\mathcal{O}) \le f(T) + \sum_{o \in \mathcal{O}\setminus T} f(o \mid T) \le f(T) + z/2$ which implies $f(T) \ge f(T \cup \mathcal{O}) - (1+\epsilon)/2 \cdot f(\mathcal{O})$. ☐

Theorem 58. *There exists a single-pass, $O(\epsilon^{-1}2^{k/\epsilon}\log\Delta)$-space algorithm that finds a $1/2-\epsilon$ approximation in expectation of* $\mathsf{SubmodularMax}(k)$.

Proof. We first implicitly condition on a fixed $h(\mathcal{O}) = D$. We consider the algorithm copy that corresponds to the correct guess $C = D$. We then condition on a fixed $X = \{x \in \mathcal{N} \setminus \mathcal{O} : h(x) \in D\}$. Let T^X be the solution of the algorithm given a fixed X.

If $|S| = k$, we are done as $f(T) \ge z/2 \ge 1/2 \cdot f(\mathcal{O})$. Otherwise, according to Lemma 57, we have $f(T^X) \ge f(T^X \cup \mathcal{O}) - (1+\epsilon)/2 \cdot f(\mathcal{O})$. We then unfix X to yield

$$\mathrm{E}\left[f(T)\right] \ge \mathrm{E}\left[f(T\cup\mathcal{O})\right] - \frac{1+\epsilon}{2} \cdot f(\mathcal{O}) \ .$$

We have argued that for all $v \notin \mathcal{O}$, we have $\Pr[v \in D \mid h(\mathcal{O}) = D] \leq \epsilon$. Then $v \in T$ with probability at most ϵ. According to Corollary 56,

$$\mathrm{E}[f(T \cup \mathcal{O})] = \mathrm{E}[f((T \setminus \mathcal{O}) \cup \mathcal{O})] \geq (1-\epsilon) f(\mathcal{O}) \ .$$

Note that the above expectations are still implicitly conditioned on a fixed D. Finally, we unfix D and conclude $\mathrm{E}[f(T)] \geq (1/2 - \epsilon) f(\mathcal{O})$. Because $h(\mathcal{O})$ is unknown, we could run the algorithm on $\binom{k/\epsilon}{k} = O(2^{k/\epsilon})$ different guesses and return best solution given among the copies. Guessing z entails a factor $\tilde{O}(\epsilon^{-1} \log \Delta)$ in space. ☐

$O(1/\epsilon)$-pass algorithm. We now exhibit a multiple pass algorithm that returns a $1 - 1/e - \epsilon$ approximation of $\mathsf{SubmodularMax}(k)$. For monotone functions, this result could be derived from previous work [116]. Here we consider the non-monotone case. We again make use of the coloring scheme. The algorithm adds an item to the solution if its marginal gain is above some threshold. This threshold decreases by a factor $1+\epsilon$ after each pass.

1. Guess z such that $f(\mathcal{O}) \leq z \leq 2f(\mathcal{O})$ and guess $C = h(\mathcal{O})$.
2. For $i = 1, 2, \ldots, r$ where $r = \lceil \log_{1+\epsilon}(10 \cdot e) \rceil$, make a pass over the stream:
   (a) For each item u, if $|S| < k$ and $h(u) \in C$ and
   $$f(u \mid S) \geq \frac{z}{k(1+\epsilon)^i}, \text{ then } S \leftarrow S \cup \{u\} \ .$$
3. Return S as the final solution T.

Theorem 59. *There exists a $O(\epsilon^{-1})$-pass, $O(\epsilon^{-1} \cdot 2^{k/\epsilon} \log \Delta)$-space algorithm that finds an expected $1 - 1/e - \epsilon$ approximation of* $\mathsf{SubmodularMax}(k)$.

Proof. We implicitly condition on a fixed $h(\mathcal{O}) = D$. We then consider the algorithm copy corresponding to the correct guess of $C = D$. Let S_i be the solution after i items were added. For convenience, let $y = (1-\epsilon)f(\mathcal{O})$. Appealing to Corollary 56 and the random coloring scheme, for all i, we have

$$\mathrm{E}\left[f(S_i \cup \mathcal{O})\right] = \mathrm{E}\left[f((S_i \setminus \mathcal{O}) \cup \mathcal{O})\right] \geq (1-\epsilon)f(\mathcal{O}) = y \ .$$

The last inequality again follows from the fact that any item $u \notin \mathcal{O}$ is in S_i with probability at most ϵ as argued. We condition on a fixed $X = \{x \in \mathcal{N} \setminus \mathcal{O} : h(x) \in D\}$. Suppose the algorithm picks k items. Consider the items that are picked in the jth pass. If $j = 1$, then according to the algorithm,

$$f(S_i^X) - f(S_{i-1}^X) \geq \frac{z}{k} \geq \frac{f(\mathcal{O} \ \mid \ S_{i-1}^X)}{k} = \frac{f(\mathcal{O} \cup S_{i-1}^X) - f(S_{i-1}^X)}{k} \ .$$

For $j > 1$, there must be an item $o \in \mathcal{O} \setminus S_{i-1}^X$ such that $f(o \mid S_{i-1}^X) \geq 1/k \cdot f(\mathcal{O} \mid S_{i-1}^X)$. Since, o was not picked in the $(j-1)$th pass, we must have

$$f(S_i^X) - f(S_{i-1}^X) \geq \frac{z}{k(1+\epsilon)^j} \geq \frac{f(o \ \mid \ S_{i-1}^X)}{1+\epsilon} \geq \frac{1}{1+\epsilon}\frac{f(\mathcal{O} \cup S_{i-1}^X) - f(S_{i-1}^X)}{k} \ .$$

Unfixing X yields

$$\mathrm{E}\left[f(S_i)\right] - \mathrm{E}\left[f(S_{i-1})\right] \geq \frac{\mathrm{E}\left[f(\mathcal{O} \cup S_{i-1})\right] - \mathrm{E}\left[f(S_{i-1}]\right)}{k(1+\epsilon)} \geq \frac{y - \mathrm{E}\left[f(S_{i-1}]\right)}{k(1+\epsilon)} \ .$$

Therefore,

$$\begin{aligned} y - \mathrm{E}\left[f(S_i)\right] &\leq y - \frac{1}{1+\epsilon}\frac{y - \mathrm{E}\left[f(S_{i-1})\right]}{k} - \mathrm{E}\left[f(S_{i-1})\right] \\ &= \left(1 - \frac{1}{k(1+\epsilon)}\right)(y - \mathrm{E}\left[f(S_{i-1})\right]) \\ &\leq \left(1 - \frac{1}{k(1+\epsilon)}\right)^i y \end{aligned}$$

where the last inequality follows from induction. Thus, by letting $i = k$, we have

$$\mathrm{E}\left[f(S_k)\right] \geq \left(1 - \left(1 - \frac{1}{(1+\epsilon)k}\right)^k\right) y \geq (1 - 1/e - 4\epsilon) f(\mathcal{O}) \ .$$

Finally, if the algorithm picks fewer than k items. Let the final solution be T. We have

$$f(T^X \cup \mathcal{O}) \leq f(T^X) + \sum_{o \in \mathcal{O} \setminus T} f(o \mid T^X) \ .$$

Unfixing X and appealing to Corollary 56, we deduce that

$$(1-\epsilon) f(\mathcal{O}) \leq \mathrm{E}\left[f(T)\right] + \frac{f(\mathcal{O})}{5e} \ .$$

Therefore,

$$\mathrm{E}\left[f(T)\right] \geq \left(1 - \frac{1}{5e} - 2\epsilon\right) f(\mathcal{O}) \ .$$

Finally, we unfix D to deduce the claim. □

CHAPTER 7

TESTING BAYESIAN NETWORKS IN DATA STREAMS

7.1 Introduction and Related Work

The problem of testing n-wise independence in data streams has attracted recent attention in streaming algorithms literature [27, 28, 78]. In that problem, the stream consists of m items that are n-tuples (i.e., each item has n coordinates) that empirically defines a joint distribution of n random variables $X_1, X_2, \ldots, X_n$ where each X_i has range $[k] := \{1, 2, \ldots k\}$. One can think of X_i as the value of the ith coordinate of a random item in the stream. Specifically, the stream defines the joint probability mass function (pmf):

$$\begin{aligned}\mathcal{P}(x_1, \ldots, x_n) &= \Pr\left[X_1 = x_1 \text{ and } X_2 = x_2 \text{ and } \ldots \text{ and } X_n = x_n\right] && (7.1)\\ &:= \frac{c(x_1, x_2, \ldots, x_n)}{m} \ , && (7.2)\end{aligned}$$

where $c(x_1, x_2, \ldots, x_n)$ is the number of tuples equal to $(x_1, x_2, \ldots, x_n)$. The marginal probability of a subset of variables $\{X_j : j \in S\}$ for some $S \subset [n]$ is defined as:

$$\Pr\left[X_j = x_j \ \ \forall j \in S\right] := \sum_{x_\ell \in [k] \text{ for all } \ell \notin S} \Pr\left[X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n\right] \ .$$

The goal of previous work was to determine whether this distribution is close to being a product distribution or equivalently, whether the corresponding random variables are close to being independent by estimating:

$$\left(\sum_{x_1,\ldots,x_n\in[k]} \left|\Pr\left[X_1=x_1,\ldots,X_n=x_n\right]-\Pr\left[X_1=x_1\right]\ \cdots\Pr(X_n=x_n)\right|^p\right)^{1/p}$$

However, it is natural to ask more general questions about the dependencies between the variables, e.g., can we identify an X_i such that the other random variables are independent conditioned on X_i or whether there is an ordering $X_{\sigma(1)}, X_{\sigma(2)}, X_{\sigma(3)}, \ldots$ such that $X_{\sigma(i)}$ is independent of $X_{\sigma(1)}, X_{\sigma(2)}, \ldots, X_{\sigma(i-2)}$ conditioned on $X_{\sigma(i-1)}$.

The standard way to represent such dependencies is via Bayesian networks. A Bayesian network is a directed acyclic graph G with a node X_i corresponding to each variable X_i along with a set of directed edges E that encode a factorization of the joint distribution. Specifically, if $\mathrm{Pa}(X_i)=\{X_j : (X_j \to X_i)\in E\}$ are the parents of X_i in G then the Bayesian network represents the assertion that for all $x_1, x_2, \ldots, x_n$, the joint distribution can be factorized as follows:

$$\Pr\left[X_1=x_1, X_2=x_2,\ldots,X_n=x_n\right]=\prod_{i=1}^{n}\Pr\left[X_i=x_i|X_j=x_j\ \forall\ X_j\in\mathrm{Pa}(X_i)\right]\ .$$

For example, $E=\emptyset$ corresponds to the assertion that the X_i are fully independent whereas the graph on nodes $\{X_1, X_2, X_3\}$ with directed edges $X_1 \to X_2, X_1 \to X_3$ corresponds to the assertion that X_2 and X_3 are independent conditioned on X_1.

In this chapter, we consider the problem of evaluating how well the observed data fits a Bayesian network. The data stream of tuples in $[k]^n$ and a Bayesian network G defines an empirical distribution $\mathcal{P}_G$ with the pmf:

$$\mathcal{P}_G(x_1,\ldots,x_n) := \prod_{i=1}^{n}\Pr\left[X_i=x_i\ \mid\ X_j=x_j\ \forall\ X_j\in\mathrm{Pa}(X_i)\right], \tag{7.3}$$

where

$$\Pr\left[X_i=x_i\ \mid\ X_j=x_j \text{ for all } j\in\mathrm{Pa}(X_i)\right]=\frac{\Pr\left[X_i=x_i \text{ and } X_j=x_j, \forall X_j\in\mathrm{Pa}(X_i)\right]}{\Pr\left[X_j=x_j,\ \forall X_j\in\mathrm{Pa}(X_i)\right]}\ . \tag{7.4}$$

is just the fraction of tuples whose ith coordinate is x_i amongst the set of tuples whose jth coordinate is x_j for all $X_j \in \mathrm{Pa}(X_i)$. We then define the error of G to be the ℓ_p norm, for $p \in \{1, 2\}$, of the difference between the joint distribution and the factorization $\mathcal{P}_G$:

$$\mathcal{E}_p(G) := \left(\sum_{x_1,\ldots,x_n \in [k]} |\mathcal{P}(x_1, \ldots, x_n) - \mathcal{P}_G(x_1, \ldots, x_n)|^p \right)^{\frac{1}{p}} := \|\mathcal{P} - \mathcal{P}_G\|_p.$$

Clearly, if the factorization implied by G is valid then $\mathcal{E}_p(G) = 0$. More generally, if $\mathcal{E}_p(G)$ is small then we consider the factorization to be close to valid. The use of ℓ_p distance to measure "closeness" was considered previously in the Bayesian network literature [80, 124]. However, the space required to compute these measures was considered a major drawback because it was assumed that it would be necessary to explicitly store the full joint distribution whose space complexity is $\Omega(k^n)$. Our results show that this is not the case. Note that when G is the empty graph, $\mathcal{E}_p(\emptyset)$ is the quantity measured in [27, 28, 78].

In many applications, data comes in a streaming fashion. When it comes to very large data volume, it is important to maintain a data structure that uses small memory and estimates different statistics about the data accurately at the same time. As the space requirement to measure the accuracy of Bayesian networks is as large as $O(k^n)$ and as the size of our data set m increases, our problem of evaluating Bayesian networks via data streams with small memory is of considerable importance.

Our results. Here, and henceforth we use k, n, d and m to denote the range of the variables, the number of the variables, the maximum in-degree of the network and the length of the stream respectively.

1. *Testing and Estimating ℓ_p Accuracy.* For any Bayesian network G, we prove a space lower bound $\Omega(nk^d)$ for the problem of testing whether the data is

consistent with G, i.e., $\mathcal{E}_p(G) = 0$.The lower bound is based on the Local Markov Property, a result from Bayesian Networks literature, and a reduction from communication complexity.

2. We also present a $\tilde{O}(\epsilon^{-2}nk^{d+1})$-space algorithm for estimating $\mathcal{E}_p(G)$ up to a $(1+\epsilon)$ factor. This is near-optimal up to polylogarithmic factors.

Notation. $A \perp B \mid C$ denotes the assertion that random variables A, B are independent conditioned on C, i.e.,

$$\Pr\left[A = a, B = b \;\mid\; C = c\right] = \Pr\left[A = a \;\mid\; C = c\right] \Pr\left[B = b \;\mid\; C = c\right]$$

for all a, b, c in the range of A, B, C. $\mathrm{Pa}(X_i)$ denotes the set of variables that are parents of X_i and $\mathrm{ND}(X_i)$ denotes the set of variables that are non-descendants of X_i, other than $\mathrm{Pa}(X_i)$.

7.2 Algorithms for Estimating $\mathcal{E}_p(G)$

In this section, we present approximation algorithms for estimating $\mathcal{E}_p(G)$ for an arbitrary Bayesian network G. We first note that the factorized distribution $\mathcal{P}_G$ can be computed and stored exactly in $O(nk^{d+1}\log m)$ bits since, by Eq. (7.1) and Eq. (7.4), it suffices to compute

$$\frac{\sum_{a\in[k]^n \;:\; a_j = x_j \;\forall j \;s.t\; X_j \in \{X_i\}\cup \mathrm{Pa}(X_i)} c(a)}{\sum_{a\in[k]^n \;:\; a_j = x_j \;\forall j \;s.t\; X_j \in \mathrm{Pa}(X_i)} c(a)} \;.$$

for each $i \in [n]$ and each of at most k^{d+1} combinations of values for X_i and $\mathrm{Pa}(X_i)$. Given this observation, it is straightforward to approximate $\mathcal{E}_p(G)$ given any data stream "sketch" algorithm that returns a $(1+\epsilon)$ estimate for the ℓ_p norm of a vector v. Kane et al. [88] presented such as algorithm that uses space that is logarithmic in the dimension of the vector.

Specifically, we apply the algorithm on a vector v defined as follows. Consider v to be indexed as $[k] \times [k] \times \ldots \times [k]$. On the arrival of tuple $(x_1, \ldots, x_n)$, we increment the coordinate corresponding to $(x_1, \ldots, x_n)$ by $1/m$. At the end of the stream, v encodes the empirical joint distribution. For each $(x_1, \ldots, x_n)$, we now decrement the corresponding coordinate by $\mathcal{P}_G(x_1, \ldots, x_n)$. At this point, $v_{x_1,\ldots,x_n} = \mathcal{P}(x_1, \ldots, x_n) - \mathcal{P}_G(x_1, \ldots, x_n)$ and hence the ℓ_p norm of v is $\mathcal{E}_p(G)$. Hence, returning the estimate from the algorithm yields a $1+\epsilon$ approximation to $\mathcal{E}_p(G)$ as required.

Note that this simple approach also improves over existing work [28] on the case of measuring $\ell_p(G)$ when G has no edges (i.e., measuring how far the data is from independent) unless n is very small compared to k. The space used in previous work is doubly-exponential in n but logarithmic in k whereas our approach uses $\tilde{O}(nk)$ space and hence, our approach is more space-efficient unless $k > 2^{n^n}/n$.

Theorem 60. *There exists a single-pass algorithm that computes a* $(1+\epsilon)$ *approximation of* $\mathcal{E}_p(G)$ *with probability at least* $1-\delta$ *using* $\tilde{O}(\epsilon^{-2}k^{d+1}n\log\delta^{-1})$ *space.*

We also design a slightly more-efficient two-pass algorithm that returns an additive approximation for $p = 1$.

Theorem 61. *There exists a two-pass algorithm that computes* $\mathcal{E}_1(G) \pm \epsilon$ *with probability at least* $1-\delta$ *using* $\tilde{O}(\epsilon^{-2}k^{d}n\log\delta^{-1})$ *space.*

Proof. We first rewrite:

$$\begin{aligned}\mathcal{E}_1(G) &= \sum_{x:\mathcal{P}(x)<\mathcal{P}_G(x)} (\mathcal{P}_G(x) - \mathcal{P}(x)) + \sum_{x:\mathcal{P}(x)>\mathcal{P}_G(x)} (\mathcal{P}(x) - \mathcal{P}_G(x)) \\ &= \sum_{x\in[k]^n} \mathcal{P}_G(x) g_1(x) + \sum_{x\in[k]^n} \mathcal{P}(x) g_2(x)\end{aligned}$$

where

$$g_1(x) = \begin{cases} 1 - \frac{\mathcal{P}(x)}{\mathcal{P}_G(x)}, \text{ if } \frac{\mathcal{P}(x)}{\mathcal{P}_G(x)} < 1 \\ 0, \text{ otherwise} \end{cases}$$

$$g_2(x) = \begin{cases} 1 - \frac{\mathcal{P}_G(x)}{\mathcal{P}(x)}, \text{ if } \frac{\mathcal{P}_G(x)}{\mathcal{P}(x)} < 1 \\ 0, \text{ otherwise.} \end{cases}$$

In the first pass, we sample $O(\epsilon^{-2} \log \delta^{-1})$ samples from each of $\mathcal{P}$ and $\mathcal{P}_G$. A sample $\mathcal{S}'$ from $\mathcal{P}$ can be chosen by just picking a tuple from the stream uniformly at random using Reservoir sampling [137]. Using $\tilde{O}(nk^d)$ space we can construct a sample $\mathcal{S} = (s_1, s_2, \ldots, s_n)$ from the distribution $\mathcal{P}_G$ as follows. For each $i \in [n]$ and $y = [k]^{|\operatorname{Pa}(X_i)|}$, we pick a tuple uniformly among those where the value of the set of variables in $\operatorname{Pa}(X_i)$ equals y and set s_i^y be the ith value of this tuple. At the end of the stream, we build a sample $\mathcal{S} = (s_1, \ldots, s_n)$ from the $O(nk^d)$ stored values: first we set $s_i = s_i^{\emptyset}$ for all i where $\operatorname{Pa}(X_i) = \emptyset$, then set $s_i = s_i^y$ if the $\operatorname{Pa}(X_i)$ have already been set to y.

In the second pass, for each sample $\mathcal{S}$ drawn from $\mathcal{P}_G$ in the first pass, compute $g_1(\mathcal{S})$ exactly and for each sample $\mathcal{S}'$ drawn from $\mathcal{P}$, compute $g_2(\mathcal{S}')$ exactly. Note that $\operatorname{E}[g_1(\mathcal{S})] + \operatorname{E}[g_2(\mathcal{S}')] = \mathcal{E}_1(G)$. Since, $g_1(\mathcal{S})$ and $g_2(\mathcal{S}')$ are bounded between 0 and 1, an application of the Chernoff bound implies that repeating this process $O(\epsilon^{-2} \log \delta^{-1})$ in parallel yields an additive ϵ approximation with probability at least $1 - \delta$. □

7.3 Lower Bound for Estimating $\mathcal{E}_p(G)$

Decision problem. We now prove a space lower bound for testing $\mathcal{E}_p(G) = 0$.

Definition 62. *A Bayesian network G with vertices $X_1, .., X_n$ satisfies the* Local Markov Property *if $X_i \perp \operatorname{ND}(X_i) \mid \operatorname{Pa}(X_i)$ for all $i \in [n]$.*

We rely on the following theorem. Its proof can be found in many Bayesian networks literature such as [82].

Theorem 63 (Jordan et al. [82]). *(Local Markov Property) Any given Bayesian network G satisfies $\mathcal{E}_p(G) = 0$ if and only if it satisfies the Local Markov Property.*

Next, we show that the approximation algorithm above is near optimal. It has been shown that independence testing via ℓ_p distance can be done in $O(\operatorname{polylog} k)$ space. The open question we are trying to answer is whether it is still possible to test more general dependencies in $O(\operatorname{polylog} k)$ space. Unfortunately, the answer is, in general, no. We first prove that for testing whether two variables are perfectly independent given the third variable, any constant-pass streaming algorithm requires $\Omega(k)$ space.

The proofs of our lower bounds use the standard technique of reducing from a communication complexity problem. In particular, we consider the disjointness problem where Alice and Bob each has a string $x \in \{1,2\}^k$ and $y \in \{1,2\}^k$ respectively and want evaluate $\mathsf{DISJ}(x,y)$ where

$$\mathsf{DISJ}(x,y) = \begin{cases} 0 & \text{if there exists } i \text{ such that } x_i = y_i = 1 \\ 1 & \text{otherwise} \end{cases}$$

A classic result [87] shows that any (randomized) protocol with constant number of rounds for this problem requires $\Omega(k)$ bits to be communicated. The following remark is useful in our reduction.

Lemma 64. *Given a stream of two binary (A,B)-tuple $(a,2),(2,b)$. Then, A,B are independent if and only if a,b are not both equal 1.*

Proof. If $a = b = 1$, then $\Pr[A = 1 \text{ and } B = 2] = 0.5 \neq \Pr[A = 1]\Pr[B = 2] = 0.5 \times 0.5 = 0.25$. Otherwise, one can easily check that $\Pr[A = x \text{ and } B = y] = \Pr[A = x]\Pr[B = y]$ for all $x,y \in \{1,2\}$. □

Proposition 65. *There exists a network G such that any constant-pass algorithm that decides if $\mathcal{E}_p(G) = 0$ with probability at least 2/3 requires $\Omega(k^d)$ space.*

Proof. Consider the Bayesian network G with vertices $X_1, \ldots, X_d, Y, Z$ where each X_i is a parent of $X_1, X_2, \ldots, X_{i-1}, Y$, and Z. Let $X = (X_1, \ldots, X_d)$ and $x = (x_1, \ldots, x_d)$. Then,

$$\mathcal{E}_1(G) = \sum_{\substack{y,z \in [2] \\ x \in [k]^d}} \Bigg| \Pr\left[Y = y, Z = z \;\mid\; X = x\right] \Pr\left[X = x\right] - \tag{7.5}$$

$$\Pr\left[Y = y \;\mid\; X = x\right] \Pr\left[Z = z \;\mid\; X = x\right] \prod_{i=1}^{d} \Pr\left[X_i = x_i \;\mid\; X_{i+1} = x_{i+1}, \ldots, X_d = x_{d+1}\right] \Bigg| \tag{7.6}$$

$$= \Bigg| \Pr\left[Y = y, Z = z \;\mid\; X = x\right] - \Pr\left[Y = y \;\mid\; X = x\right] \Pr\left[Z = z \;\mid\; X = x\right] \Bigg| \Pr\left[X = x\right] \;. \tag{7.7}$$

We make the reduction from DISJ where Alice and Bob, with bit strings a and b of length k^d, generate the streams S_A and S_B of $(Y, Z, X_1, X_2, \ldots, X_d)$-tuples respectively:

$$\begin{aligned} S_A &= \{(a_i, 2, x_1, \ldots, x_d) : (x_1, \ldots, x_d) \in [k]^d\} \\ S_B &= \{(2, b_i, x_1, \ldots, x_d) : (x_1, \ldots, x_d) \in [k]^d\} \;. \end{aligned}$$

By Equation 7.7, we have that $\mathcal{E}_p(G) = 0$ if and only if $Y \perp Z \mid \{X = c\}$ for all $c \in [k]^d$. By Lemma 64, this is satisfied if and only if $\mathsf{DISJ}(a, b) = 1$. Therefore, any constant-pass algorithm that decides if $\mathcal{E}_p(G) = 0$ requires $\Omega(k^d)$ space.

□

We now construct a more sophisticated reduction to incorporate n into the lower bound.

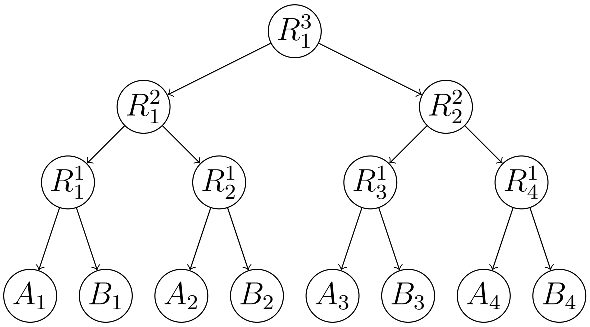

Figure 7.1. Construction for $n = 4$

Theorem 66. *There exists a Bayesian network G such that any constant-pass algorithm that determines if $\mathcal{E}_p(G) = 0$ with probability at least 2/3 requires $\Omega(nk^d)$ space.*

Proof. Without loss of generality, assume n is a power of 2. Let $x \in \{1,2\}^{nk}, y \in \{1,2\}^{nk}$ be an instance of DISJ where it be convenient to index x and y by $[n] \times [k]$. The Bayesian network we consider is balanced binary tree with leaves A_1, B_1, A_2, $B_2, \ldots, A_n$, B_n and internal nodes R_i^j where R_i^1 in the parent of A_i and B_i and R_i^j is the parent of R_{2i-1}^{j-1} and R_{2i}^{j-1} for $j > 1$. The root node is $R_1^{\log n+1}$. See Figure 7.1. The variables R_i^j will take $3k$ different values and it will be convenient to index these values as $[3] \times [k]$. The leaf variables take either the value 1 or 2.

Alice generates a stream that defines samples from the joint distribution based on x. Each sample generated satisfies the following criteria and all distinct samples that obey this criteria are generated:

1. $R_1^{\log n+1} \in \{(1,z),(2,z) : z \in [k]\}$.

2. If $R_i^j = (1,z)$ for $j > 1$:

   - The left child $R_{2i-1}^{j-1} \in \{(1,z),(2,z)\}$ and the right child $R_{2i}^{j-1} = (3,z)$.

3. If $R_i^j = (2,z)$ for $j > 1$:

- The left child $R_{2i-1}^{j-1} = (3, z)$ and the right child $R_{2i}^{j-1} \in \{(1, z), (2, z)\}$.

4. If $R_i^j = (3, z)$ for $j > 1$:
    - Both the values for the children R_{2i-1}^{j-1} and R_{2i}^{j-1} are $(3, z)$.
5. If $R_i^1 \in \{(1, z), (2, z)\}$:
    - The values for the children are $A_i = x_{i,z}$, $B_i = 2$
6. If $R_i^1 = (3, z)$:
    - The values for the children are $A_i = 2$, $B_i = 2$

Bob then generates a series samples in a similar manner except that Rule 5 becomes: If $R_i^1 \in \{(1, z), (2, z)\}$, then $A_i = 2$, $B_i = y_{i,z}$.

Note that each sample generated by either Alice or Bob specifies a path from the root to a pair A_i, B_i as following: Starting from the root, if the current node's value is equal to $(1, z)$, then go to its left child; on the other hand, if its value is equal to $(2, z)$, then go to the right child. Once we commit to a direction, every descendant on the other direction is set to $(3, z)$ for the R nodes and 2 for the A and B nodes.

First assume that $\mathsf{DISJ}(x, y) = 0$. Then $x_{i,z} = y_{i,z} = 1$ for some $z \in [k], i \in [n]$. By Lemma 64 we infer that A_i and B_i are not independent conditioned on either $R_i^1 = (1, z)$ or $R_i^1 = (2, z)$ and hence, $\mathcal{E}_p(G) \neq 0$.

Conversely, assume that $\mathsf{DISJ}(x, y) = 1$. The Local Markov Property says that if every vertex is independent of its non-descendants given its parents then $\mathcal{E}_p(G) = 0$.

- First we show that it is true for any R_i^j variable. Conditioned on the event that the parent of R_i^j takes the value $(3, z)$, R_i^j is constant and hence independent of its non-descendants. Conditioned on the event that the parent of R_i^j takes the value $(1, z)$ or $(2, z)$, the values of the non-descendants of R_i^j are fixed and hence independent of R_i^j.

- Next, we show that it is true for any A_i variable. The argument for B_i is identical. Conditioned on $R_i^1 = (3, z)$, then A_i is constant and hence independent of all non-descendants. If $R_i^1 = (1, z)$ or $R_i^1 = (2, z)$, the values of all non-descendants, except possibly B_i, are constant. But by Lemma 64, B_i is independent of A_i conditioned on R_i^1 since $\mathsf{DISJ}(x, y) = 1$.

Hence, $\mathsf{DISJ}(x, y) = 1$ if and only if $\mathcal{E}_p(G) = 0$ and therefore testing if $\mathcal{E}_p(G) = 0$ requires $\Omega(nk)$ space.

To extend the lower bound to $\Omega(nk^d)$ consider an instance of DISJ of length nk^d. Let the variables in G be children of all $d-1$ new variables $D_1, \ldots, D_{d-1}$ where there is a directed edge between $D_i \to D_j$ for $i > j$. Let the new network be G'. Similar to the proof of Theorem 65, to solve DISJ on the wth pair of bit strings of length nk where $w \in [k^{d-1}]$, Alice and Bob generate samples with variables in G as described above and set $(D_1, \ldots, D_{d-1}) = w$. Hence, any streaming algorithm that decides if $\mathcal{E}_p(G') = 0$ requires $\Omega(nk^d)$ space. □

CHAPTER 8

FINDING SUBCUBE HEAVY HITTERS IN DATA STREAMS

8.1 Introduction and Related Work

We study the problem of finding heavy hitters in high dimensional data streams. Most companies see transactions with items sold, time, store location, price, etc. that arrive over time. Modern online companies see streams of user web activities that typically have components of user information including ID (e.g., cookies), hardware (e.g., device), software (e.g., browser, OS), and contents such as web properties, apps. Activity streams also include events (e.g., impressions, views, clicks, purchases) and event attributes (e.g., product id, price, geolocation, time). Even classical IP traffic streams have many dimensions including source and destination IP addresses, port numbers and other features of an IP connection such as application type. Furthermore, in applications such as Natural Language Processing, streams of documents can be thought of as streams of a large number of bigrams or multi-grams over word combinations [73]. As these examples show, analytics data streams with 100's and 1000's of dimensions arise in many applications. Motivated by this, we study the problem of finding *heavy hitters* on data streams focusing on d, the number of dimensions, as a parameter. Given d one sees in practice, d^2 in space usage is prohibitive, for solving the heavy hitters problem on such streams.

Formally, let us start with a one-dimensional stream of items $x_1, \ldots x_m$ where each $x_i \in [n] := \{1, 2, \ldots, n\}$. We can look at the *count* $c(v) = |\{i : x_i = v\}|$ or the *frequency ratio* $f(v) = c(v)/m$. A *heavy hitter* value v is one with $c(v) \geq \gamma m$ or equivalently $f(v) \geq \gamma$, for some constant γ. The standard *data stream model* is that we maintain data structures of size $\text{polylog}(m, n)$ and determine if v is a heavy hitter with probability of success at least 3/4, that is, if $f(v) \geq \gamma$ output YES and output NO if $f(v) < \gamma/4$ for all v.[1] We note that if $\gamma/4 \leq f(v) < \gamma$, then either answer is acceptable.

Detecting heavy hitters on data streams is a fundamental problem that arises in guises such as finding elephant flows and network attacks in networking, finding hot trends in databases, finding frequent patterns in data mining, finding largest coefficients in signal analysis, and so on. Therefore, the heavy hitters problem has been studied extensively in theory, databases, networking and signal processing literature. See [49] for an early survey and [139] for a recent survey.

Subcube heavy hitter problems. Our focus is on modern data streams such as in analytics cases, with d dimensions, for large d. The data stream consists of d-dimensional items $x_1, \ldots, x_m$. In particular,

$$x_i = (x_{i,1}, \ldots, x_{i,d}) \text{ and each } x_{i,j} \in [n] \ .$$

A *k-dimensional subcube* T is a subset of k distinct coordinates $\{T_1, \cdots, T_k\} \subseteq [d]$. We refer to the joint values of the coordinates T of x_i as $x_{i,T}$.

The number of items whose coordinates T have joint values v is denoted by $c_T(v)$, i.e., $c_T(v) = |\{i : x_{i,T} = v\}|$. Finally, we use X_T to denote the random variable of the joint values of the coordinates T of a random item. We have the following relationship

$$f_T(v) := \Pr[X_T = v] = \frac{c_T(v)}{m} \ .$$

---

[1] The gap constant 4 can be narrowed arbitrarily and the success probability can be amplified to $1 - \delta$ as needed, and we omit these factors in the discussions.

For a single coordinate i, we slightly abuse the notation by using f_i and $f_{\{i\}}$ interchangeably. For example, $f_{T_i}(v)$ is the same as $f_{\{T_i\}}(v)$. Similarly, X_i is the same as $X_{\{i\}}$.

We are now ready to define our problems. They take k, γ as parameters and the stream as the input and build data structures to answer:

- *Subcube Heavy Hitter*: Query(T, v), where $|T| = k$, and $v \in [n]^k$, returns an estimate if $f_T(v) \geq \gamma$. Specifically, output YES if $f_T(v) \geq \gamma$ and NO if $f_T(v) < \gamma/4$. If $\gamma/4 \leq f_T(v) < \gamma$, then either output is acceptable. The required success probability *for all* k-dimensional subcubes T and $v \in [n]^k$ is at least 3/4.
- *All Subcube Heavy Hitters*: AllQuery(T) outputs all joint values v that return YES to Query(T, v). This is conditioned on the algorithm used for Query(T, v).

It is important to emphasize that the stream is presented (in a single pass or constant passes) to the algorithm before the algorithm receives any query.

Subcube heavy hitters are relevant wherever one dimensional heavy hitters have found applications: combination of source and destination IP addresses forms the subcube heavy hitters that detect network attacks; combination of stores, sales quarters and nature of products forms the subcube heavy hitters that might be the pattern of interest in the data, etc. Given the omnipresence of multiple dimensions in digital analytics, arguably, subcube heavy hitters limn the significant data properties far more than the single dimensional view.

Related work. The problem we address is directly related to frequent itemset mining studied in the data mining community. In frequent itemset mining, each dimension is binary ($n = 2$), and we consider Query(T, v) where $v = \mathbf{U}_k := (1, \ldots, 1)$. It is known that counting all maximal subcubes T that have a frequent itemset, i.e., $f_T(\mathbf{U}_k) \geq \gamma$, is $\#P$-complete [140]. Furthermore, finding even a single T of maximal size such that $f_T(\mathbf{U}_k) \geq \gamma$ is NP-hard [76,108]. Recently, Liberty et al. showed that any constant-pass streaming algorithm answering Query($T, \mathbf{U}_k$) requires $\Omega(kd/\gamma \cdot \log(d/k))$ bits of memory in general [108]. In the worst case, this is $\Omega(d^2/\gamma)$ for large k, ignoring the polylogarithmic factors. For this specific problem, sampling algorithms will nearly meet their lower bound for space. Our problem is more general, with arbitrary n and v.

Our contributions Clearly, the case $k = 1$ can be solved by building one of the many known single dimensional data structures for the heavy hitters problem on each of the d dimension; the $k = d$ case can be thought of as a giant single dimensional problem by linearizing the space of all values in $[n]^k$; for any other k, there are $\binom{d}{k}$ distinct choices for subcube T, and these could be treated as separate one-dimensional problems by linearizing each of the subcubes. In general, this entails $\binom{d}{k}$ and $\log(n^d)$ cost in space or time bounds over the one-dimensional case, which we seek to avoid. Also, our problem can be reduced to the binary case by unary encoding each dimension by n bits, and solving frequent itemset mining: the query then has kn dimensions. The resulting bound will have an additional n factor which is large.

First, we observe that the reservoir sampling approach [137] solves subcube heavy hitters problems more efficiently compared to the approaches mentioned above. Our analysis shows that the space we use is within polylogarithmic factors of the lower bound shown in [108] for binary dimensions and query vector $\mathbf{U}_k$, which is a special case of our problem. Therefore, the subcube heavy hitters problem can be solved using $\tilde{O}(kd/\gamma)$ space. However, this is $\Omega(d^2)$ in worst case.

Our main contribution is to avoid this quadratic bottleneck for finding subcube heavy hitters. We adopt the notion that there is an underlying probabilistic model behind the data, and in the spirit of the Naive Bayes model, we assume that the dimensions are nearly (not exactly) mutually independent given an observable latent dimension. This could be considered as a low rank factorization of the dimensions. In particular, one could formalize this assumption by bounding the total variational distance between the data's joint distribution and that derived from the Naive Bayes formula. This assumption is common in statistical data analysis and highly prevalent in machine learning. Following this modeling, we make two main contributions:

- We present a two-pass, $\tilde{O}(d/\gamma)$-space streaming algorithm for answering Query(T, v). This improves upon the kd factor in the space complexity from sampling, without assumptions, to just d with the Naive Bayes assumption, which would make this algorithm practical for large k. Our algorithm uses sketching in each dimension in one pass to detect heavy hitters, and then needs a second pass to precisely estimate their frequencies.

- We present a fast algorithm for answering AllQuery(T) in $\tilde{O}((k/\gamma)^2)$ time. The naive procedure would take exponential time $\Omega((1/\gamma)^k)$ by considering the Cartesian product of the heavy hitters in each dimension. Our approach, on the other hand, uses the structure of the Naive Bayes assumption to iteratively construct the subcube heavy hitters one dimension at a time.

Our work develops the direction of model-based data stream analysis. Model-based data analysis has been effective in other areas. For example, in compressed sensing, realistic signal models that include dependencies between values and locations of the signal coefficients improve upon unconstrained cases [55]. In statistics, using tree constrained models of multidimensional data sometimes improves point and density estimation. In high dimensional distribution testing, model based approach has also been studied to overcome the curse of dimensionality [54].

In the data stream model, [27, 28, 78] studied the problem of testing independence. McGregor and Vu [115] studied the problem of evaluating Bayesian Networks. In another work, Kveton et al. [104] assumed a tree graphical model and designed a one-pass algorithm that estimates the joint frequency; their work however only solved the $k = d$ case for the joint frequency estimation problem. Our model is a bit different and more importantly, we solve the subcube heavy hitters problem (addressing all the $\binom{d}{k}$ subcubes) which prior work does not solve. In following such a direction, we have extended the fundamental heavy hitters problem to higher dimensional data. Given that many implementations already exist for the sketches we use for one-dimensional heavy hitters as a blackbox, our algorithms are therefore easily implementable.

Background on the Naive Bayes model and its use in our context. The Naive Bayes Model [127] is a Bayesian network over d features $X_1, \ldots, X_d$ and a class variable Y. This model represents a joint probability distribution of the form

$$\begin{aligned} &\Pr\left[X_1 = x_1, \ldots, X_d = x_d, Y = y\right] \\ =&\Pr\left[Y = y\right] \prod_{j=1}^{d} \Pr\left[X_j = x_j \mid Y = y\right] \ , \end{aligned}$$

which means that the values of the features are conditionally independent given the value of the class variable. The simplicity of the Naive Bayes model makes it a popular choice in text processing and information retrieval [107, 110], with state-of-the-art performance in spam filtering [10], text classification [107], and others.

8.2 The Sampling Algorithm

In this section, we show that sampling solves the problem efficiently compared to running one-dimensional heavy hitters algorithms for each of $\binom{d}{k}$ k-dimensional subcubes independently. It also matches the lower bound in [108] up to polylogarithmic factors.

Algorithm details. The algorithm samples $m' = \tilde{O}(\gamma^{-1} kd)$ random items $z_1, \ldots, z_{m'}$ from the stream using Reservoir sampling [137]. Let $S = \{z_1, \ldots, z_{m'}\}$ be the sample set. Given Query(T, v), we output YES if and only if the sample frequency of v, denoted by $\hat{f}_T(v)$, is at least $\gamma/2$. Specifically,

$$\hat{f}_T(v) := \frac{|\{x_i : x_i \in S \text{ and } x_{i,T} = v\}|}{m'} \ .$$

For all subcubes T and joint values v of T, the expected sample frequency $\hat{f}_T(v)$ is $f_T(v)$. Intuitively, if v is a frequent joint values, then its sample frequency $\hat{f}_T(v) \approx f_T(v)$; otherwise, $\hat{f}_T(v)$ stays small.

Let us fix a k-dimensional subcube T and suppose that for all $v \in [n]^k$, we have

$$\hat{f}_T(v) = f_T(v) \pm \frac{\max\{\gamma, f_T(v)\}}{4} \ . \tag{8.1}$$

It is then straightforward to see that if $f_T(v) < \gamma/4$, then $\hat{f}_T(v) < \gamma/4 + \gamma/4 = \gamma/2$. Otherwise, if $f_T(v) \geq \gamma$, then $\hat{f}_T(v) \geq 3f_T(v)/4 \geq 3\gamma/4 > \gamma/2$. Hence, we output YES for all v where $\hat{f}_T(v) \geq \gamma/2$, and output NO otherwise.

Lemma 67. *(Chernoff bound) Let $X_1, \cdots, X_n$ be independent or negatively correlated binary random variables. Let $X = \sum_{i=1}^n X_i$ and $\mu = \mathrm{E}[X]$. Then,*

$$\Pr[|X - \mu| \geq \epsilon\mu] \leq \exp(-\min\{\epsilon^2, \epsilon\}\mu/3) \ .$$

Recall that $S = \{z_1, z_2, \ldots, z_{m'}\}$ is the sample set returned by the algorithm. For a fixed $v \in [n]^k$, we use Z_i as the indicator variable for the event $z_{i,T} = v$. Since we sample without replacement, the random variables Z_i are negatively correlated. The following lemma shows that Eq. 8.1 holds for all v and k-dimensional subcubes T via Chernoff bound.

Lemma 68. *For all k-dimensional subcubes T and joint values $v \in [n]^k$, with probability at least 0.9,*

$$\hat{f}_T(v) = f_T(v) \pm \frac{\max\{\gamma, f_T(v)\}}{4} \ .$$

Proof. Let $m' = c\gamma^{-1}\log(d^k \cdot n^k)$ for some sufficiently large constant c. We first consider a fixed $v \in [n]^k$ and define the random variables Z_i as above, i.e., $Z_i = 1$ if $z_{i,T} = v$. Suppose $f_T(v) \geq \gamma$. Appealing to Lemma 67, we have

$$\begin{aligned}
&\Pr\left[\left|\left(\sum_{i=1}^{m'} \frac{Z_i}{m'}\right) - f_T(v)\right| \geq \frac{f_T(v)}{4}\right] \\
=& \Pr\left[\left|\hat{f}_T(v) - f_T(v)\right| \geq \frac{f_T(v)}{4}\right] \\
\leq& \exp\left(-\frac{f_T(v)m'}{3 \times 16}\right) \leq \frac{1}{10d^k n^k} \ .
\end{aligned}$$

On the other hand, if $f_T(v) < \gamma/4$, then

$$\begin{aligned}
\Pr\left[\left|\hat{f}_T(v) - f_T(v)\right| \geq \frac{\gamma}{4}\right] &\leq \exp\left(-\left(\frac{\gamma}{4f_T(v)}\right) f_T(v)\frac{m'}{3}\right) \\
&\leq \frac{1}{10d^k n^k} \ .
\end{aligned}$$

Therefore, by taking the union bound over all $\binom{d}{k} \cdot n^k \leq d^k \cdot n^k$ possible combinations of k-dimensional subcubes and the corresponding joint values $v \in [n]^k$, we deduce the claim. $\square$

We therefore could answer all Query(T, v) correctly with probability at least 0.9 for all joint values $v \in [n]^k$ and k-dimensional subcubes T. Because storing each sample z_i requires $\tilde{O}(d)$ bits of space, the algorithm uses $\tilde{O}(dk\gamma^{-1})$ space. We note that answering Query(T, v) requires computing $\hat{f}_T(v)$ which takes $O(|S|)$ time. We can answer AllQuery(T) by computing $\hat{f}_T(v)$ for all joint values v of coordinates T that appear in the sample set which will take $O(|S|^2)$ time. We summarize the result as follows.

Theorem 69. *There exists a 1-pass algorithm that uses $\tilde{O}(dk\gamma^{-1})$ space and solves k-dimensional subcube heavy hitters. Furthermore,* Query(T, v) *and* AllQuery(T) *take $\tilde{O}(dk\gamma^{-1})$ and $\tilde{O}\left((dk\gamma^{-1})^2\right)$ time respectively.*

8.3 Algorithms under The Near-Independence Assumption

The near-independence assumption. Suppose the random variables representing the dimensions $X_1, X_2, \ldots, X_d$ are *near* independent. We show that there is a 2-pass algorithm that uses less space and has faster query time. At a high level, we make the assumption that the joint probability is approximately factorized

$$f_{\{1,\ldots,d\}}(v) \approx f_1(v_1) f_2(v_2) \cdots f_d(v_d) \ .$$

More formally, we assume that the total variation distance is bounded by a small quantity α. Furthermore, we assume that α is reasonable with respect to γ that controls the heavy hitters. For example, $\alpha \le \gamma/10$ will suffice.

The formal *near-independence* assumption is as follows: There exists $\alpha \le \gamma/10$ such that for all subcubes T,

$$\max_{v \in [n]^{|T|}} \left| f_T(v) - \prod_{i=1}^{|T|} f_{T_i}(v_i) \right| < \alpha \ .$$

We observe that:

- If $f_T(v) \ge \gamma$, then

$$\prod_{i=1}^{|T|} f_{T_i}(v_i) \ge f_T(v) - \gamma/10 > \gamma/2 \ .$$

- If $f_T(v) < \gamma/4$, then

$$\prod_{i=1}^{|T|} f_{T_i}(v_i) \le f_T(v) + \gamma/10 < \gamma/2 \ .$$

Thus, it suffices to output YES to Query(T, v) if and only if the marginals product $\prod_{i=1}^{|T|} f_{T_i}(v_i) \ge \gamma/2$. For convenience, let

$$\lambda := \gamma/2 \ .$$

Algorithm details. We note that simply computing $f_i(x)$ for all coordinates $i \in [d]$ and $x \in [n]$ will need $\Omega(dn)$ space. To over come this, we make following simple but useful observation. We observe that if v is a heavy hitter in the subcube T and if T' is a subcube of T, then $v_{T'}$ is a heavy hitter in the subcube T'.

Lemma 70. *For all subcubes T,*

$$\prod_{i=1}^{|T|} f_{T_i}(v_i) \geq \lambda \implies \prod_{i \in \mathcal{V}} f_{T_i}(v_i) \geq \lambda$$

for all $\mathcal{V} \subseteq [|T|]$ (i.e., $\{T_i : i \in \mathcal{V}\}$ is a subcube of T).

The proof is trivial since all $f_{T_i}(v_i) \leq 1$. Therefore, we have the following corollary.

Corollary 71. *For all subcubes T,*

$$\prod_{i=1}^{|T|} f_{T_i}(v_i) \geq \lambda \implies f_{T_i}(v_i) \geq \lambda \text{ for all } i \in [|T|] \ .$$

We therefore only need to compute $f_i(x)$ if x is a heavy hitter in coordinate i. To this end, for each coordinate $i \in [d]$, by using (for example) Misra-Gries algorithm [118] or Count-Min sketch [53], we can find a set H_i such that if $f_i(x) \geq \lambda/2$, then $x \in H_i$ and if $f_i(x) < \lambda/4$, then $x \notin H_i$. In the second pass, for each $x \in H_i$, we compute $f_i(x)$ exactly to obtain

$$S_i := \{x \in [n] : f_i(x) \geq \lambda\} \ .$$

We output YES to Query(T, v) if and only if all $v_i \in S_{T_i}$ and $\prod_{i=1}^{|T|} f_{T_i}(v_i) \geq \lambda$. Note that if $v \in S_i$, then $f_i(v)$ is available to the algorithm since it is computed exactly in the second pass. The detailed algorithm is as follows.

1. First pass: For each coordinate $i \in [d]$, use Misra-Gries algorithm to find H_i.

2. Second pass: For each coordinate $i \in [d]$, compute $f_i(x)$ exactly for each $x \in H_i$ to obtain S_i.

3. Output YES to Query(T, v) if and only if $v_i \in S_{T_i}$ for all $i \in [|T|]$ and

$$\prod_{i=1}^{|T|} f_{T_i}(v_i) \geq \lambda \ .$$

The next theorem establishes that the above algorithm solves subcube heavy hitters under the near-independence assumption.

Theorem 72. *There exists a 2-pass algorithm that uses $\tilde{O}(d\gamma^{-1})$ space and solves subcube heavy hitters under the near-independence assumption. The time to answer* Query(T, v) *and* AllQuery(T) *are $\tilde{O}(k)$ and $\tilde{O}(k\gamma^{-1})$ respectively where k is the dimensionality of T.*

Proof. The first pass uses $\tilde{O}(d\lambda^{-1})$ space since Misra-Gries algorithm uses $\tilde{O}(\lambda^{-1})$ space for each coordinate $i \in [d]$. Since the size of each H_i is $O(\lambda^{-1})$, the second pass also uses $\tilde{O}(d\lambda^{-1})$ space. Recall that $\lambda = \gamma/2$. We then conclude that the algorithm uses $\tilde{O}(d\gamma^{-1})$ space.

For an arbitrary Query(T, v), the algorithm's correctness follows immediately from Corollary 71 and the observation that if $v_i \in S_{T_i}$, then $f_{T_i}(v_i)$ is available since it was computed exactly in the second pass. Specifically, if

$$\prod_{i=1}^{|T|} f_{T_i}(v_i) \geq \lambda \ , \tag{8.2}$$

then $v_i \in S_{T_i}$ for all $i \in [|T|]$ and we could verify the inequality and output YES. On the other hand, suppose Eq. 8.2 does not hold. Then, if $v_i \notin S_{T_i}$ for some i, we correctly output NO. But if all $v_i \in S_{T_i}$, then we are able to verify that the inequality does not hold (and correctly output NO).

The parameter k only affects the query time. We now analyze the time to answer Query(T, v) and AllQuery(T) for a k-dimensional subcube T.

We can easily see that Query(T, v) takes $\tilde{O}(k)$ time as we need to check if all $v_i \in S_{T_i}$ (e.g., using binary searches) and compute $\prod_{i=1}^{k} f_{T_i}(v_i)$.

Next, we exhibit a fast algorithm to answer AllQuery(T). We note that naively checking all combinations $(v_1, \cdots, v_k)$ in $S_{T_1} \times S_{T_2} \times \cdots \times S_{T_k}$ takes exponential $\Omega(\gamma^{-k})$ time in the worst case.

Our approach figures out the heavy hitters gradually and takes advantage of the near-independence assumption. In particular, define

$$W_j := \{v \in [n]^j : f_{T_1}(v_1) \cdots f_{T_j}(v_j) \geq \lambda\} \ .$$

Recall that the goal is to find W_k. Note that $W_1 = S_1$ is obtained directly by the algorithm. We now show that it is possible to construct W_{j+1} from W_j in $\tilde{O}(\lambda^{-1})$ time which in turn means that we can find W_k in $\tilde{O}(k\lambda^{-1})$ time. We use the notation $T_{[j]} := \{T_1, \ldots, T_j\}$ and $v_{[j]} := (v_1, v_2, \ldots, v_j)$.

We note that $|W_j| \leq 5/(4\lambda)$. This holds since if $y \in W_j$, then $\prod_{i=1}^{j} f_{T_i}(y_i) \geq \lambda$. Appealing to the near-independence assumption, we have

$$f_{T_{[j]}}(y) \geq \prod_{i=1}^{j} f_{T_i}(y_i) - \alpha \geq \lambda - \alpha \geq 4/5 \cdot \lambda \ .$$

For each $y \in W_j$, we collect all $x \in S_{j+1}$ such that

$$\left(\prod_{i=1}^{j} f_{T_i}(y_i) \right) f_{T_{j+1}}(x) \geq \lambda$$

and put $(y_1, \cdots, y_j, x)$ into W_{j+1}. Since $|W_j| \leq 5/4 \cdot \lambda^{-1}$ and $|S_{j+1}| \leq \lambda^{-1}$, this step obviously takes $O(\lambda^{-2})$ time. However, by observing that there could be at most $\lambda^{-1} \prod_{i=1}^{j} f_{T_i}(y_i)$ such x for each $y \in W_j$, the upper bound for the number of combinations of x and y is

$$\begin{aligned}\sum_{y \in W_j} \frac{1}{\lambda} \prod_{i=1}^{j} f_{T_i}(y_i) &\leq \sum_{y \in W_j} \frac{1}{\lambda}(f_{T_{[j]}}(y) + \alpha) \\ &= \sum_{y \in W_j} \frac{\alpha}{\lambda} + \sum_{y \in W_j} \frac{f_{T_{[j]}}(y)}{\lambda} \\ &\leq |W_j| + \frac{1}{\lambda} \leq \frac{3}{\lambda} \ .\end{aligned}$$

The last inequality follows from the assumption that $\alpha \leq \lambda/5$ and $\sum_{y \in W_j} f_{T_{[j]}}(y) \leq 1$. Thus, the algorithm can find W_{j+1} given W_j in $\tilde{O}(\lambda^{-1})$ time. Hence, we obtain W_k in $\tilde{O}(k\lambda^{-1}) = \tilde{O}(k\gamma^{-1})$ time. The correctness of this procedure follows directly from Lemma 70 and induction since $v = (v_1, \ldots, v_{j+1}) \in W_{j+1}$ implies that $v_{[j]} \in W_j$ and $v_{j+1} \in S_{j+1}$. Thus, by checking all combinations of $y \in W_j$ and $x \in S_{j+1}$, we can construct W_{j+1} correctly. □

8.4 Algorithms under The Naive Bayes Assumption

The Naive Bayes assumption. In this section, we focus on the data streams inspired by the Naive Bayes model which is strictly more general than the near-independence assumption. In particular, we assume that the coordinates are near-independent given an extra $(d+1)$th observable *class coordinate* that has a value in $\{1, \ldots, \ell\}$. The $(d+1)$th coordinate is also often referred to as the *latent coordinate.*

As in typical in Naive Bayes analysis, we assume ℓ is a constant but perform the calculations in terms of ℓ so its role in the complexity of the problem is apparent.

Informally, this model asserts that the random variables representing coordinates $X_1, \ldots, X_d$ are near independent conditioning on a the random variable X_{d+1} that represents the class coordinate.

We introduce the following notation

$$\begin{aligned} f_{T \mid d+1}(v \mid z) &:= \frac{\left|\{x_i : x_{i,T} = v \wedge x_{i,\{d+1\}} = z\}\right|}{\left|\{x_i : x_{i,\{d+1\}} = z\}\right|} \\ &= \Pr\left[X_T = v \mid X_{d+1} = z\right] . \end{aligned}$$

In other words, $f_{T \mid d+1}(v \mid z)$ is the frequency of the joint values v in the T coordinates among the stream items where the class coordinate $d+1$ has value z.

The formal *Naive Bayes* assumption is as follows: There exists $\alpha \leq \gamma/10$ such that for all subcubes T,

$$\max_{v \in [n]^{|T|}} \left| f_T(v) - \sum_{z \in [\ell]} f_{d+1}(z) \prod_{i=1}^{|T|} f_{T_i \mid d+1}(v_i \mid z) \right| < \alpha .$$

Algorithm details. As argued in the previous section, it suffices to output YES to Query(T, v) if and only if

$$\sum_{z \in [\ell]} f_{d+1}(z) \prod_{i=1}^{|T|} f_{T_i \mid d+1}(v_i \mid z) \geq \gamma/2 = \lambda .$$

However, naively computing all $f_{i \mid d+1}(v \mid z)$ uses $\Omega(\ell d n)$ space. We circumvent this problem by generalizing Lemma 70 as follows. If a joint values v is a heavy hitter in a subcube T in the Naive Bayes formula and T' is a subcube of T, then $v_{T'}$ is a heavy hitter in the subcube T'.

Lemma 73. *For all subcubes T,*

$$q(v) := \sum_{z\in[\ell]} f_{d+1}(z) \prod_{i=1}^{|T|} f_{T_i \mid d+1}(v_i \mid z) \geq \lambda$$
$$\implies \sum_{z\in[\ell]} f_{d+1}(z) \prod_{i\in\mathcal{V}} f_{T_i \mid d+1}(v_i \mid z) \geq \lambda$$

for all $\mathcal{V} \subseteq [|T|]$ (i.e., $\{T_i : i \in \mathcal{V}\}$ is a subcube of T).

Proof. For a fixed z, observe that

$$\sum_{y_j\in[n]} f_{T_j \mid d+1}(y_j \mid z) = 1 \ .$$

Suppose $q(v) \geq \lambda$ and consider an arbitrary $\mathcal{V} \subseteq [|T|]$. We have

$$\begin{aligned}
&\sum_{z\in[\ell]} f_{d+1}(z) \prod_{i\in\mathcal{V}} f_{T_i \mid d+1}(v_i \mid z) \\
=&\sum_{z\in[\ell]} f_{d+1}(z) \prod_{i\in\mathcal{V}} f_{T_i \mid d+1}(v_i \mid z) \prod_{j\notin\mathcal{V}} \left(\sum_{y_j\in[n]} f_{T_j \mid d+1}(y_j \mid z) \right) \\
\geq&\sum_{z\in[\ell]} f_{d+1}(z) \prod_{i\in\mathcal{V}} f_{T_i \mid d+1}(v_i \mid z) \prod_{j\notin\mathcal{V}} f_{T_j \mid d+1}(v_j \mid z) \\
=&\sum_{z\in[\ell]} f_{d+1}(z) \prod_{i=1}^{|T|} f_{T_i \mid d+1}(v_i \mid z) = q(v) \geq \lambda \ .
\end{aligned}$$

An alternative proof is by noticing that $q(v)$ is a valid probability density function of $|T|$ variables. The claim follows by marginalizing over the the variables that are not in $\mathcal{V}$. □

Setting $\mathcal{V} = \{i\}$ for each $i \in [|T|]$ and appealing to the fact that

$$\sum_{z\in[\ell]} f_{d+1}(z) f_{T_i \mid d+1}(v_i \mid z) = \sum_{z\in[\ell]} f_{\{T_i,d+1\}}((v_i, z)) = f_{T_i}(v_i) \ ,$$

we deduce the following corollary.

Corollary 74. *For all subcubes* T*,*

$$\sum_{z\in[\ell]} f_{d+1}(z)\prod_{i=1}^{|T|} f_{T_i \mid d+1}(v_i \mid z) \geq \lambda \implies f_{T_i}(v_i) \geq \lambda$$

for all $i \in [|T|]$*.*

Therefore, we only need to compute $f_{i \mid d+1}(x \mid z)$ for all coordinates $i \in [d]$, values $z \in [\ell]$ if x is a heavy hitter of coordinate i. Similar to the previous section, for each dimension $i \in [d]$, we find H_i in the first pass and use H_i to find S_i in the second pass. Appealing to Corrollary 74, we deduce that if

$$q(v) := \sum_{z\in[\ell]} f_{d+1}(z)\prod_{i=1}^{|T|} f_{T_i \mid d+1}(v_i \mid z) \geq \lambda$$

then for all $i = 1, 2, \ldots, |T|$, we have $f_{T_i}(v_i) \geq \lambda$ which in turn implies that $v_i \in S_{T_i}$. Therefore, we output YES to Query(T, v) if and only if all $v_i \in S_{T_i}$ and $q(v) \geq \lambda$.

To this end, we only need to compute $f_{i \mid d+1}(x \mid z)$ and $f_{d+1}(z)$ for all $x \in H_i$, $z \in [\ell]$, and $i \in [d]$. The detailed algorithm is as follows.

1. First pass:
    (a) For each value $z \in [\ell]$, compute $f_{d+1}(z)$ exactly.
    (b) For each coordinate $i \in [d]$, use Misra-Gries algorithm to find H_i.
2. Second pass:
    (a) For each coordinate $i \in [d]$ and each value $x \in H_i$, compute $f_i(x)$ exactly to obtain S_i.
    (b) For each value $z \in [\ell]$, coordinate $i \in [d]$, and $x \in H_i$, compute $f_{i \mid d+1}(x \mid z)$ exactly.
3. Output YES to $\text{Query}(T, v)$ if and only if $v_i \in S_{T_i}$ for all $i \in [|T|]$ and

$$\sum_{z \in [\ell]} f_{d+1}(z) \prod_{i=1}^{|T|} f_{T_i \mid d+1}(v_i \mid z) \geq \lambda \ .$$

The next theorem establishes that the above algorithm solves subcube heavy hitters under the Naive Bayes assumption.

Theorem 75. *There exists a 2-pass algorithm that uses* $\tilde{O}(\ell d \gamma^{-1})$ *space and solves subcube heavy hitters under the Naive Bayes assumption. The time to answer* $\text{Query}(T, v)$ *and* $\text{AllQuery}(T)$ *are* $\tilde{O}(\ell k)$ *and* $O(\ell(k/\gamma)^2)$ *respectively where* k *is the dimensionality of* T*.*

Proof. The space to obtain H_i and S_i over the two passes is $\tilde{O}(d\lambda^{-1})$. Additionally, computing $f_{i \mid d+1}(x \mid z)$ for all $i \in [d]$, $z \in [\ell]$, and $x \in H_i$ requires $\tilde{O}(\ell d \lambda^{-1})$ bits of space. The overall space we need is therefore $\tilde{O}(\ell d \lambda^{-1}) = \tilde{O}(\ell d \gamma^{-1})$.

The correctness of answering an arbitrary $\text{Query}(T, v)$ follows directly from Corollary 74. Specifically, if

$$\sum_{z\in[\ell]} f_{d+1}(z)\prod_{i=1}^{|T|} f_{T_i \mid d+1}(v_i \mid z) \geq \lambda \ , \tag{8.3}$$

then, $v_i \in S_{T_i} \subseteq H_{T_i}$ for all $i \in [|T|]$ as argued. Hence, $f_{T_i \mid d+1}(v_i \mid z)$ is computed exactly in the second pass for all $z \in [\ell]$. As a result, we could verify the inequality and output YES. On the other hand, if Eq. 8.3 does not hold. Then, if some $v_i \notin S_{T_i}$, we will correctly output NO. Otherwise if all $v_i \in S_{T_i}$, then we can compute the left hand side and verify that Eq. 8.3 does not hold (and correctly output NO).

Obviously, Query(T, v) takes $\tilde{O}(\ell k)$ time for a k-dimensional subcube T. We now exhibit a fast algorithm to answer AllQuery(T) for a k-dimensional subcube T. Define

$$W_j := \{v \in [n]^j : \sum_{z\in[\ell]} f_{d+1}(z)\prod_{i=1}^{j} f_{T_i \mid d+1}(v_i \mid z) \geq \lambda\} \ .$$

Recall that the goal is to find W_k. We note that $W_1 = S_1$ is obtained directly by the algorithm. Next, we show how to obtain W_{j+1} in $\tilde{O}(\lambda^{-2})$ time from W_j. Note that $|W_j| \leq 5/(4\lambda)$ because if $y \in W_j$, then

$$\sum_{z\in[\ell]} f_{T_1 \mid d+1}(y_1 \mid z)\cdots f_{T_j \mid d+1}(y_j \mid z) f_{d+1}(z) \geq \lambda$$

and hence $f_{T_{[j]}}(y) \geq \lambda - \alpha = 4/5 \cdot \lambda^{-1}$ according to the Naive Bayes assumption. This implies that $|W_j| \leq 5/(4\lambda)$.

For each $(v_1, \cdots, v_j)$ in W_j, we collect all $v_{j+1} \in S_{j+1}$ such that

$$\sum_{z\in[\ell]} f_{T_1 \mid d+1}(v_1 \mid z)\cdots f_{T_{j+1} \mid d+1}(v_{j+1} \mid z) f_{d+1}(z) \geq \lambda$$

and put $(v_1, \cdots, v_{j+1})$ to W_{j+1}. Since $|W_j| \leq 5/(4\lambda)$ and $|S_{j+1}| \leq 1/\lambda$, this step obviously takes $\tilde{O}(\ell k \lambda^{-2})$ time. Since we need to do this for $j = 2, 3, \ldots, k$, we attain

W_k in $\tilde{O}(\ell(k/\gamma)^2)$ time. The correctness of this procedure follows directly from Lemma 73 and induction since $(v_1, \ldots, v_{j+1}) \in W_{j+1}$ implies that $(v_1, \ldots, v_j)$ is in W_j and v_{j+1} is in S_{j+1}. Since we check all possible combinations of $(v_1, \ldots, v_j) \in W_j$ and $v_{j+1} \in S_{j+1}$, we guarantee to construct W_{j+1} correctly. □

It is possible to improve upon the running time of AllQuery(T) with a divide-and-conquer approach. Without loss of generality, we assume that $|T| = k$ be a power of 2. Let us consider the following divide-and-conquer algorithm:

1. Input AllQuery(T).
2. Let $T_L = \{T_1, \ldots, T_{k/2}\}$ and $T_R = \{T_{k/2+1}, \ldots, T_k\}$.
3. Solve for AllQuery(T_L) and AllQuery(T_R).
4. For all v_L returned by AllQuery(T_L) and v_R returned by AllQuery(T_R), let $v = (v_1, \ldots, v_k)$ be the joint values formed by combining v_L and v_R. Specifically, $v_L = (v_1, \ldots, v_{k/2})$ and $v_R = (v_{k/2+1}, \ldots, v_k)$.
5. Add v to the solution if the AllQuery(T, v) returns YES.

We notice that the base case $k = 1$ can be done by simply outputting S_i corresponding to the single coordinate in T. The correctness of this algorithm follows directly from Lemma 73.

Since the time complexity for a Query(T, v) is $O(\ell k)$. The running time of each recursive level is at most $O(\ell k \cdot 1/\gamma^2)$ by the same argument above, i.e., there could be at most $1/\gamma$ different joint values v_L and $1/\gamma$ different joint values v_R. Thus, we have the following.

Theorem 76. *The time to answer* Query(T, v) *can be improved to* $O(\ell k \cdot (1/\gamma)^2 \log k)$ *where* k *is the dimensionality of* T*.*

CHAPTER 9

FUTURE WORK

In this chapter, we propose some related research problems for future work.

9.1 Graph Theory and Combinatorial Optimization

One open question is whether we could improve the query time for the densest subgraph problem. Bhattacharya et al. provided a dynamic algorithm with a 1/4 approximation [25]. Therefore, an interesting result would be a better approximation while still maintaining polylog(n) query time.

Another important research topic is to investigate the effect of random order stream in the context of combinatorial optimization. Very recently, for the problem of maximizing a monotone submodular function under a cardinality constraint, Norouzi-Fard et al. showed that we could beat a 1/2 approximation by a small constant in random order streams [121]. This result could be applied directly to the maximum k-set coverage problem we considered in this thesis. One can ask whether the coverage function admits a more significant improvement in terms of approximation. Additionally, it is also natural to attempt to extend their result to non-monotone submodular functions.

For streaming set problems, we would also like to consider other related models. A set system can be represented by a bipartite graph G with the bipartition (L, R) where the nodes in L correspond the elements in the universe and the nodes in R correspond to the sets. An edge (i, j) indicates that the ith element belongs to the jth set. Bateni et al. considered the edge-arrival model where the stream consists of edges (i, j) in this bipartite graph [22]; however, some results quickly become infeasible in this model. For example, we and they independently showed that any constant approximation in this model for the maximum coverage problem requires $\Omega(m)$ space. On the other hand, in the streaming set model where edges are grouped by end points in R, we showed that it is possible to obtain a constant approximation in $\tilde{O}(k)$ space. Another natural model would be the edge arrival model where the edges are grouped by end points in L.

9.2 High Dimensional Data Streams

We propose some open problems in high dimensional data streams. The first problem is to design an ℓ_p sampling algorithm that allows us to sample a tuple from an arbitrary subset of coordinates (i.e., subcube). The naive approach is to maintain 2^d samplers corresponding to 2^d subcubes. Note that it is trivial to beat this bound for the case $p = 1$ by simply sampling an item in the stream at uniformly random. However, for other values of p, nothing is currently known.

A closely related problem is to estimate the number of distinct tuples in each subcube. Given a d-dimensional stream, we want to estimate (up to a constant factor) the number of distinct tuples in each of 2^d subcubes. Is it possible to beat the naive upper bound $\Omega(2^d)$ that maintains an F_0 sketch for each subcube?

BIBLIOGRAPHY

[1] Ageev, Alexander A., and Sviridenko, Maxim. Approximation algorithms for maximum coverage and max cut with given sizes of parts. In *IPCO* (1999), vol. 1610 of *Lecture Notes in Computer Science*, Springer, pp. 17–30.

[2] Ageev, Alexander A., and Sviridenko, Maxim. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *J. Comb. Optim. 8*, 3 (2004), 307–328.

[3] Ahn, Kook Jin, Cormode, Graham, Guha, Sudipto, McGregor, Andrew, and Wirth, Anthony. Correlation clustering in data streams. In *ICML* (2015), vol. 37 of *JMLR Workshop and Conference Proceedings*, JMLR.org, pp. 2237–2246.

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

[5] Ahn, Kook Jin, Guha, Sudipto, and McGregor, Andrew. Graph sketches: sparsification, spanners, and subgraphs. In *PODS* (2012), ACM, pp. 5–14.

[6] Ahn, Kook Jin, Guha, Sudipto, and McGregor, Andrew. Spectral sparsification in dynamic graph streams. In *APPROX-RANDOM* (2013), vol. 8096 of *Lecture Notes in Computer Science*, Springer, pp. 1–10.

[7] Alon, Noga, Yuster, Raphael, and Zwick, Uri. Finding and counting given length cycles. *Algorithmica 17*, 3 (1997), 209–223.

[8] Anagnostopoulos, Aris, Becchetti, Luca, Bordino, Ilaria, Leonardi, Stefano, Mele, Ida, and Sankowski, Piotr. Stochastic query covering for fast approximate document retrieval. *ACM Trans. Inf. Syst. 33*, 3 (2015), 11:1–11:35.

[9] Andoni, Alexandr, Krauthgamer, Robert, and Onak, Krzysztof. Streaming algorithms via precision sampling. In *FOCS* (2011), IEEE Computer Society, pp. 363–372.

[10] Androutsopoulos, Ion, Paliouras, Georgios, Karkaletsis, Vangelis, Sakkis, Georgios, Spyropoulos, Constantine D., and Stamatopoulos, Panagiotis. Learning to filter spam e-mail: A comparison of a naive bayesian and a memory-based approach. *CoRR cs.CL/0009009* (2000).

[11] Arge, Lars, Goodrich, Michael T., and Sitchinava, Nodari. Parallel external memory graph algorithms. In *IPDPS* (2010), IEEE, pp. 1–11.

[12] Arifuzzaman, Shaikh, Khan, Maleq, and Marathe, Madhav V. PATRIC: a parallel algorithm for counting triangles in massive networks. In *CIKM* (2013), ACM, pp. 529–538.

[13] Assadi, Sepehr. Tight space-approximation tradeoff for the multi-pass streaming set cover problem. In *PODS* (2017), ACM, pp. 321–335.

[14] Assadi, Sepehr, Khanna, Sanjeev, and Li, Yang. Tight bounds for single-pass streaming complexity of the set cover problem. In *STOC* (2016), ACM, pp. 698–711.

[15] Assadi, Sepehr, Khanna, Sanjeev, Li, Yang, and Yaroslavtsev, Grigory. Maximum matchings in dynamic graph streams and the simultaneous communication model. In *SODA* (2016), SIAM, pp. 1345–1364.

[16] Ausiello, Giorgio, Boria, Nicolas, Giannakos, Aristotelis, Lucarelli, Giorgio, and Paschos, Vangelis Th. Online maximum k-coverage. *Discrete Applied Mathematics 160*, 13-14 (2012), 1901–1913.

[17] Badanidiyuru, Ashwinkumar, Mirzasoleiman, Baharan, Karbasi, Amin, and Krause, Andreas. Streaming submodular maximization: massive data summarization on the fly. In *KDD* (2014), ACM, pp. 671–680.

[18] Badanidiyuru, Ashwinkumar, and Vondrák, Jan. Fast algorithms for maximizing submodular functions. In *SODA* (2014), SIAM, pp. 1497–1514.

[19] Bahmani, Bahman, Goel, Ashish, and Munagala, Kamesh. Efficient primal-dual graph algorithms for mapreduce. In *WAW* (2014), vol. 8882 of *Lecture Notes in Computer Science*, Springer, pp. 59–78.

[20] Bahmani, Bahman, Kumar, Ravi, and Vassilvitskii, Sergei. Densest subgraph in streaming and mapreduce. *PVLDB 5*, 5 (2012), 454–465.

[21] Bar-Yossef, Ziv, Kumar, Ravi, and Sivakumar, D. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *SODA* (2002), ACM/SIAM, pp. 623–632.

[22] Bateni, MohammadHossein, Esfandiari, Hossein, and Mirrokni, Vahab S. Almost optimal streaming algorithms for coverage problems. *CoRR abs/1610.08096* (2016).

[23] Bera, Suman K., and Chakrabarti, Amit. Towards tighter space bounds for counting triangles and other substructures in graph streams. In *STACS* (2017), vol. 66 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 11:1–11:14.

[24] Berry, Jonathan W., Hendrickson, Bruce, Kahan, Simon, and Konecny, Petr. Software and algorithms for graph queries on multithreaded architectures. In *IPDPS* (2007), IEEE, pp. 1–14.

[25] Bhattacharya, Sayan, Henzinger, Monika, Nanongkai, Danupon, and Tsourakakis, Charalampos E. Space- and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In *STOC* (2015), ACM, pp. 173–182.

[26] Bonnet, Édouard, Escoffier, Bruno, Paschos, Vangelis Th., and Stamoulis, Georgios. A 0.821-ratio purely combinatorial algorithm for maximum k-vertex cover in bipartite graphs. In *LATIN* (2016), vol. 9644 of *Lecture Notes in Computer Science*, Springer, pp. 235–248.

[27] Braverman, Vladimir, Chung, Kai-Min, Liu, Zhenming, Mitzenmacher, Michael, and Ostrovsky, Rafail. AMS without 4-wise independence on product domains. In *STACS* (2010), vol. 5 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 119–130.

[28] Braverman, Vladimir, and Ostrovsky, Rafail. Measuring independence of datasets. In *STOC* (2010), ACM, pp. 271–280.

[29] Braverman, Vladimir, Ostrovsky, Rafail, and Vilenchik, Dan. How hard is counting triangles in the streaming model? In *ICALP (1)* (2013), vol. 7965 of *Lecture Notes in Computer Science*, Springer, pp. 244–254.

[30] Buchbinder, Niv, Feldman, Moran, Naor, Joseph, and Schwartz, Roy. Submodular maximization with cardinality constraints. In *SODA* (2014), SIAM, pp. 1433–1452.

[31] Bulteau, Laurent, Froese, Vincent, Kutzkov, Konstantin, and Pagh, Rasmus. Triangle counting in dynamic graph streams. *CoRR abs/1404.4696* (2014).

[32] Bulteau, Laurent, Froese, Vincent, Kutzkov, Konstantin, and Pagh, Rasmus. Triangle counting in dynamic graph streams. *Algorithmica 76*, 1 (2016), 259–278.

[33] Buriol, Luciana S., Frahling, Gereon, Leonardi, Stefano, Marchetti-Spaccamela, Alberto, and Sohler, Christian. Counting triangles in data streams. In *PODS* (2006), ACM, pp. 253–262.

[34] Caskurlu, Bugra, Mkrtchyan, Vahan, Parekh, Ojas, and Subramani, K. On partial vertex cover and budgeted maximum coverage problems in bipartite graphs. In *IFIP TCS* (2014), vol. 8705 of *Lecture Notes in Computer Science*, Springer, pp. 13–26.

[35] Censor-Hillel, Keren, Levy, Rina, and Shachnai, Hadas. Fast distributed approximation for max-cut. In *ALGOSENSORS* (2017), vol. 10718 of *Lecture Notes in Computer Science*, Springer, pp. 41–56.

[36] Chakrabarti, Amit, Cormode, Graham, and McGregor, Andrew. Robust lower bounds for communication and stream computation. *Theory of Computing 12*, 1 (2016), 1–35.

[37] Chakrabarti, Amit, and Kale, Sagar. Submodular maximization meets streaming: matchings, matroids, and more. *Math. Program. 154*, 1-2 (2015), 225–247.

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

[39] Chakrabarti, Amit, and Wirth, Anthony. Incidence geometries and the pass complexity of semi-streaming set cover. In *SODA* (2016), SIAM, pp. 1365–1373.

[40] Charikar, Moses. Greedy approximation algorithms for finding dense components in a graph. In *Approximation Algorithms for Combinatorial Optimization, Third International Workshop, APPROX 2000, Saarbrücken, Germany, September 5-8, 2000, Proceedings* (2000), pp. 84–95.

[41] Chekuri, Chandra, Gupta, Shalmoli, and Quanrud, Kent. Streaming algorithms for submodular function maximization. In *ICALP (1)* (2015), vol. 9134 of *Lecture Notes in Computer Science*, Springer, pp. 318–330.

[42] Chekuri, Chandra, Jayram, T. S., and Vondrák, Jan. On multiplicative weight updates for concave and submodular function maximization. In *ITCS* (2015), ACM, pp. 201–210.

[43] Chekuri, Chandra, and Kumar, Amit. Maximum coverage problem with group budget constraints and applications. In *APPROX-RANDOM* (2004), vol. 3122 of *Lecture Notes in Computer Science*, Springer, pp. 72–83.

[44] Chiba, Norishige, and Nishizeki, Takao. Arboricity and subgraph listing algorithms. *SIAM J. Comput. 14*, 1 (1985), 210–223.

[45] Chitnis, Rajesh, Cormode, Graham, Esfandiari, Hossein, Hajiaghayi, MohammadTaghi, McGregor, Andrew, Monemizadeh, Morteza, and Vorotnikova, Sofya. Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In *SODA* (2016), SIAM, pp. 1326–1344.

[46] Chitnis, Rajesh Hemant, Cormode, Graham, Hajiaghayi, Mohammad Taghi, and Monemizadeh, Morteza. Parameterized streaming: Maximal matching and vertex cover. In *SODA* (2015), SIAM, pp. 1234–1251.

[47] Choudhury, Salimur Rashid, et al. *Approximation algorithms for a graph-cut problem with applications to a clustering problem in bioinformatics.* PhD thesis, Lethbridge, Alta.: University of Lethbridge, Deptartment of Mathematics and Computer Science, 2008, 2008.

[48] Clarkson, Kenneth L., and Woodruff, David P. Numerical linear algebra in the streaming model. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009* (2009), pp. 205–214.

[49] Cormode, Graham. Finding frequent items in data streams. `http://dmac.rutgers.edu/Workshops/WGUnifyingTheory/Slides/cormode.pdf`, 2008. DIMACS Workshop.

[50] Cormode, Graham, Datar, Mayur, Indyk, Piotr, and Muthukrishnan, S. Comparing data streams using hamming norms (how to zero in). *IEEE Trans. Knowl. Data Eng. 15*, 3 (2003), 529–540.

[51] Cormode, Graham, and Jowhari, Hossein. A second look at counting triangles in graph streams. *Theor. Comput. Sci. 552* (2014), 44–51.

[52] Cormode, Graham, Karloff, Howard J., and Wirth, Anthony. Set cover algorithms for very large datasets. In *CIKM* (2010), ACM, pp. 479–488.

[53] Cormode, Graham, and Muthukrishnan, S. An improved data stream summary: The count-min sketch and its applications. In *LATIN* (2004), vol. 2976 of *Lecture Notes in Computer Science*, Springer, pp. 29–38.

[54] Daskalakis, Constantinos, Dikkala, Nishanth, and Kamath, Gautam. Testing ising models. *CoRR abs/1612.03147* (2016).

[55] Duarte, Marco F., Cevher, Volkan, and Baraniuk, Richard G. Model-based compressive sensing for signal ensembles. In *Communication, Control, and Computing, 2009. Allerton 2009. 47th Annual Allerton Conference on* (2009), IEEE, pp. 244–250.

[56] Eckmann, Jean-Pierre, and Moses, Elisha. Curvature of co-links uncovers hidden thematic layers in the world wide web. *Proceedings of the National Academy of Sciences 99*, 9 (2002), 5825–5829.

[57] Eden, Talya, Levi, Amit, Ron, Dana, and Seshadhri, C. Approximately counting triangles in sublinear time. *SIAM J. Comput. 46*, 5 (2017), 1603–1646.

[58] Emek, Yuval, and Rosén, Adi. Semi-streaming set cover - (extended abstract). In *ICALP (1)* (2014), vol. 8572 of *Lecture Notes in Computer Science*, Springer, pp. 453–464.

[59] Ene, Alina, and Nguyen, Huy L. Constrained submodular maximization: Beyond 1/e. In *FOCS* (2016), IEEE Computer Society, pp. 248–257.

[60] Epasto, Alessandro, Lattanzi, Silvio, and Sozio, Mauro. Efficient densest subgraph computation in evolving graphs. In *WWW* (2015).

[61] Epasto, Alessandro, Lattanzi, Silvio, Vassilvitskii, Sergei, and Zadimoghaddam, Morteza. Submodular optimization over sliding windows. In *WWW* (2017), ACM, pp. 421–430.

[62] Feige, Uriel. A threshold of ln n for approximating set cover. *J. ACM 45*, 4 (1998), 634–652.

[63] Frieze, Alan M., and Jerrum, Mark. Improved approximation algorithms for MAX k-cut and MAX BISECTION. *Algorithmica 18*, 1 (1997), 67–81.

[64] Gallo, Giorgio, Grigoriadis, Michael D., and Tarjan, Robert Endre. A fast parametric maximum flow algorithm and applications. *SIAM J. Comput. 18*, 1 (1989), 30–55.

[65] Gaur, Daya Ram, Krishnamurti, Ramesh, and Kohli, Rajeev. Erratum to: The capacitated max k-cut problem. *Math. Program. 126*, 1 (2011), 191.

[66] Gharan, Shayan Oveis, and Vondrák, Jan. Submodular maximization by simulated annealing. In *SODA* (2011), SIAM, pp. 1098–1116.

[67] Ghashami, Mina, Liberty, Edo, Phillips, Jeff M., and Woodruff, David P. Frequent directions: Simple and deterministic matrix sketching. *SIAM J. Comput. 45*, 5 (2016), 1762–1792.

[68] Giallombardo, Giovanni, Jiang, Houyuan, and Miglionico, Giovanna. New formulations for the conflict resolution problem in the scheduling of television commercials. *Operations Research 64*, 4 (2016), 838–848.

[69] Gilbert, Anna C., and Indyk, Piotr. Sparse recovery using sparse matrices. *Proceedings of the IEEE 98*, 6 (2010), 937–947.

[70] Goel, Ashish, Kapralov, Michael, and Khanna, Sanjeev. On the communication and streaming complexity of maximum bipartite matching. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012* (2012), pp. 468–485.

[71] Goemans, Michel X., and Williamson, David P. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM 42*, 6 (1995), 1115–1145.

[72] Goldberg, A. V. Finding a maximum density subgraph. Tech. rep., Berkeley, CA, USA, 1984.

[73] Goyal, Amit, III, Hal Daumé, and Venkatasubramanian, Suresh. Streaming for large scale NLP: language modeling. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, May 31 - June 5, 2009, Boulder, Colorado, USA* (2009), The Association for Computational Linguistics, pp. 512–520.

[74] Guha, Sudipto, McGregor, Andrew, and Tench, David. Vertex and hyperedge connectivity in dynamic graph streams. In *PODS* (2015), ACM, pp. 241–247.

[75] Gupta, Anupam, Roth, Aaron, Schoenebeck, Grant, and Talwar, Kunal. Constrained non-monotone submodular maximization: Offline and secretary algorithms. In *WINE* (2010), vol. 6484 of *Lecture Notes in Computer Science*, Springer, pp. 246–257.

[76] Hamilton, Matthew, Chaytor, Rhonda, and Wareham, Todd. The parameterized complexity of enumerating frequent itemsets. In *IWPEC* (2006), vol. 4169 of *Lecture Notes in Computer Science*, Springer, pp. 227–238.

[77] Har-Peled, Sariel, Indyk, Piotr, Mahabadi, Sepideh, and Vakilian, Ali. Towards tight bounds for the streaming set cover problem. In *PODS* (2016), ACM, pp. 371–383.

[78] Indyk, Piotr, and McGregor, Andrew. Declaring independence via the sketching of sketches. In *SODA* (2008), SIAM, pp. 737–745.

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

[80] Jensen, Finn Verner, and Nielsen, Thomas Dyhre. *Bayesian networks and decision graphs*. Springer, 2007.

[81] Jha, Madhav, Seshadhri, C., and Pinar, Ali. A space-efficient streaming algorithm for estimating transitivity and triangle counts using the birthday paradox. *TKDD 9*, 3 (2015), 15:1–15:21.

[82] Jordan, Michael I, Ghahramani, Zoubin, Jaakkola, Tommi S, and Saul, Lawrence K. *An introduction to variational methods for graphical models*. Springer, 1998.

[83] Jørgensen, Allan Grønlund, and Pettie, Seth. Threesomes, degenerates, and love triangles. In *FOCS* (2014), IEEE Computer Society, pp. 621–630.

[84] Jowhari, Hossein, and Ghodsi, Mohammad. New streaming algorithms for counting triangles in graphs. In *COCOON* (2005), vol. 3595 of *Lecture Notes in Computer Science*, Springer, pp. 710–716.

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

[86] Kale, Satyen, and Seshadhri, C. Combinatorial approximation algorithms for maxcut using random walks. In *ICS* (2011), Tsinghua University Press, pp. 367–388.

[87] Kalyanasundaram, Bala, and Schnitger, Georg. The probabilistic communication complexity of set intersection. *SIAM J. Discrete Math. 5*, 4 (1992), 545–557.

[88] Kane, Daniel M., Nelson, Jelani, Porat, Ely, and Woodruff, David P. Fast moment estimation in data streams in optimal space. In *STOC* (2011), ACM, pp. 745–754.

[89] Kapralov, Michael. Better bounds for matchings in the streaming model. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013* (2013), pp. 1679–1697.

[90] Kapralov, Michael, Khanna, Sanjeev, and Sudan, Madhu. Streaming lower bounds for approximating MAX-CUT. In *SODA* (2015), SIAM, pp. 1263–1282.

[91] Kapralov, Michael, Lee, Yin Tat, Musco, Cameron, Musco, Christopher, and Sidford, Aaron. Single pass spectral sparsification in dynamic streams. In *FOCS* (2014), IEEE Computer Society, pp. 561–570.

[92] Kapralov, Michael, and Woodruff, David P. Spanners and sparsifiers in dynamic streams. In *PODC* (2014), ACM, pp. 272–281.

[93] Kempe, David, Kleinberg, Jon M., and Tardos, Éva. Maximizing the spread of influence through a social network. *Theory of Computing 11* (2015), 105–147.

[94] Khuller, Samir, Moss, Anna, and Naor, Joseph. The budgeted maximum coverage problem. *Inf. Process. Lett. 70*, 1 (1999), 39–45.

[95] Khuller, Samir, and Saha, Barna. On finding dense subgraphs. In *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I* (2009), pp. 597–608.

[96] Kogan, Dmitry, and Krauthgamer, Robert. Sketching cuts in graphs and hypergraphs. In *6th Innovations in Theoretical Computer Science* (2015).

[97] Kogan, Dmitry, and Krauthgamer, Robert. Sketching cuts in graphs and hypergraphs. In *ITCS* (2015), ACM, pp. 367–376.

[98] Kolountzakis, Mihail N., Miller, Gary L., Peng, Richard, and Tsourakakis, Charalampos E. Efficient triangle counting in large graphs via degree-based vertex partitioning. *Internet Mathematics 8*, 1-2 (2012), 161–185.

[99] Konrad, Christian. Maximum matching in turnstile streams. In *ESA* (2015), vol. 9294 of *Lecture Notes in Computer Science*, Springer, pp. 840–852.

[100] Konrad, Christian, Magniez, Frédéric, and Mathieu, Claire. Maximum matching in semi-streaming with few passes. In *APPROX-RANDOM* (2012), vol. 7408 of *Lecture Notes in Computer Science*, Springer, pp. 231–242.

[101] Krause, Andreas, and Guestrin, Carlos. Near-optimal observation selection using submodular functions. In *AAAI* (2007), AAAI Press, pp. 1650–1654.

[102] Kumar, Ravi, Moseley, Benjamin, Vassilvitskii, Sergei, and Vattani, Andrea. Fast greedy algorithms in mapreduce and streaming. *TOPC 2*, 3 (2015), 14.

[103] Kutzkov, Konstantin, and Pagh, Rasmus. Triangle counting in dynamic graph streams. In *Algorithm Theory - SWAT 2014 - 14th Scandinavian Symposium and Workshops, Copenhagen, Denmark, July 2-4, 2014. Proceedings* (2014), pp. 306–318.

[104] Kveton, Branislav, Bui, Hung Hai, Ghavamzadeh, Mohammad, Theocharous, Georgios, Muthukrishnan, S., and Sun, Siqi. Graphical model sketch. In *ECML/PKDD (1)* (2016), vol. 9851 of *Lecture Notes in Computer Science*, Springer, pp. 81–97.

[105] Lee, VictorE., Ruan, Ning, Jin, Ruoming, and Aggarwal, Charu. A survey of algorithms for dense subgraph discovery. In *Managing and Mining Graph Data*, Charu C. Aggarwal and Haixun Wang, Eds., vol. 40 of *Advances in Database Systems*. Springer US, 2010, pp. 303–336.

[106] Leskovec, Jure, Backstrom, Lars, Kumar, Ravi, and Tomkins, Andrew. Microscopic evolution of social networks. In *KDD* (2008), ACM, pp. 462–470.

[107] Lewis, David D., Yang, Yiming, Rose, Tony G., and Li, Fan. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research 5* (2004), 361–397.

[108] Liberty, Edo, Mitzenmacher, Michael, Thaler, Justin, and Ullman, Jonathan. Space lower bounds for itemset frequency sketches. In *PODS* (2016), ACM, pp. 441–454.

[109] Manjunath, Madhusudan, Mehlhorn, Kurt, Panagiotou, Konstantinos, and Sun, He. Approximate counting of cycles in streams. In *ESA* (2011), vol. 6942 of *Lecture Notes in Computer Science*, Springer, pp. 677–688.

[110] Manning, Christopher D., Raghavan, Prabhakar, and Schütze, Hinrich. *Introduction to Information Retrieval.* Cambridge University Press, 2008.

[111] McGregor, Andrew. Graph stream algorithms: a survey. *SIGMOD Record 43*, 1 (2014), 9–20.

[112] McGregor, Andrew, Tench, David, Vorotnikova, Sofya, and Vu, Hoa T. Densest subgraph in dynamic graph streams. In *MFCS (2)* (2015), vol. 9235 of *Lecture Notes in Computer Science*, Springer, pp. 472–482.

[113] McGregor, Andrew, and Vorotnikova, Sofya. Planar matching in streams revisited. In *APPROX-RANDOM* (2016), vol. 60 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 17:1–17:12.

[114] McGregor, Andrew, Vorotnikova, Sofya, and Vu, Hoa T. Better algorithms for counting triangles in data streams. In *PODS* (2016), ACM, pp. 401–411.

[115] McGregor, Andrew, and Vu, Hoa T. Evaluating bayesian networks via data streams. In *COCOON* (2015), vol. 9198 of *Lecture Notes in Computer Science*, Springer, pp. 731–743.

[116] McGregor, Andrew, and Vu, Hoa T. Better streaming algorithms for the maximum coverage problem. In *ICDT* (2017), vol. 68 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 22:1–22:18.

[117] Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., and Alon, U. Network motifs: simple building blocks of complex networks. In *Science* (2002), pp. 824–827.

[118] Misra, Jayadev, and Gries, David. Finding repeated elements. *Sci. Comput. Program. 2*, 2 (1982), 143–152.

[119] Mitzenmacher, Michael, and Upfal, Eli. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, New York, NY, USA, 2005.

[120] Monemizadeh, Morteza, and Woodruff, David P. 1-pass relative-error l_p-sampling with applications. In *SODA* (2010), SIAM, pp. 1143–1160.

[121] Norouzi-Fard, Ashkan, Tarnawski, Jakub, Mitrovic, Slobodan, Zandieh, Amir, Mousavifar, Aidasadat, and Svensson, Ola. Beyond 1/2-approximation for submodular maximization on massive data streams. In *ICML* (2018), vol. 80 of *JMLR Workshop and Conference Proceedings*, JMLR.org, pp. 3826–3835.

[122] Pagh, Rasmus, and Tsourakakis, Charalampos E. Colorful triangle counting and a mapreduce implementation. *Inf. Process. Lett. 112*, 7 (2012), 277–281.

[123] Panconesi, Alessandro, and Srinivasan, Aravind. Randomized distributed edge coloring via an extension of the chernoff-hoeffding bounds. *SIAM J. Comput. 26*, 2 (1997), 350–368.

[124] Pappas, Alexandros, and Gillies, Duncan F. A new measure for the accuracy of a bayesian network. In *MICAI 2002: Advances in Artificial Intelligence.* Springer, 2002, pp. 411–419.

[125] Pavan, A., Tangwongsan, Kanat, Tirthapura, Srikanta, and Wu, Kun-Lung. Counting and sampling triangles from a graph stream. *PVLDB 6*, 14 (2013), 1870–1881.

[126] Radhakrishnan, Jaikumar, and Shannigrahi, Saswata. Streaming algorithms for 2-coloring uniform hypergraphs. In *Algorithms and Data Structures - 12th International Symposium, WADS 2011, New York, NY, USA, August 15-17, 2011. Proceedings* (2011), pp. 667–678.

[127] Russell, Stuart J., and Norvig, Peter. *Artificial Intelligence - A Modern Approach.* Pearson Education, 2010.

[128] Saha, Barna, and Getoor, Lise. On maximum coverage in the streaming model & application to multi-topic blog-watch. In *SDM* (2009), SIAM, pp. 697–708.

[129] Schmidt, Jeanette P., Siegel, Alan, and Srinivasan, Aravind. Chernoff-hoeffding bounds for applications with limited independence. *SIAM J. Discrete Math. 8*, 2 (1995), 223–250.

[130] Spielman, Daniel A., and Teng, Shang-Hua. Spectral sparsification of graphs. *SIAM J. Comput. 40*, 4 (2011), 981–1025.

[131] Srinivasan, Aravind. Distributions on level-sets with applications to approximation algorithms. In *FOCS* (2001), IEEE Computer Society, pp. 588–597.

[132] Sun, He. Counting hypergraphs in data streams. *CoRR abs/1304.7456* (2013).

[133] Suri, Siddharth, and Vassilvitskii, Sergei. Counting triangles and the curse of the last reducer. In *WWW* (2011), ACM, pp. 607–614.

[134] Tangwongsan, Kanat, Pavan, A., and Tirthapura, Srikanta. Parallel triangle counting in massive streaming graphs. In *CIKM* (2013), ACM, pp. 781–786.

[135] Trevisan, Luca. Max cut and the smallest eigenvalue. *SIAM J. Comput. 41*, 6 (2012), 1769–1786.

[136] Tsourakakis, Charalampos E., Kolountzakis, Mihail N., and Miller, Gary L. Triangle sparsifiers. *J. Graph Algorithms Appl. 15*, 6 (2011), 703–726.

[137] Vitter, Jeffrey Scott. Random sampling with a reservoir. *ACM Trans. Math. Softw. 11*, 1 (1985), 37–57.

[138] Welles, Brooke Foucault, Devender, Anne Van, and Contractor, Noshir S. Is a friend a friend?: investigating the structure of friendship networks in virtual worlds. In *CHI Extended Abstracts* (2010), ACM, pp. 4027–4032.

[139] Woodruff, David P. New algorithms for heavy hitters in data streams (invited talk). In *ICDT* (2016), vol. 48 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 4:1–4:12.

[140] Yang, Guizhen. The complexity of mining maximal frequent itemsets and maximal frequent patterns. In *KDD* (2004), ACM, pp. 344–353.

[141] Yu, Huiwen, and Yuan, Dayu. Set coverage problems in a one-pass data stream. In *SDM* (2013), SIAM, pp. 758–766.

[142] Zhu, Wenxing, Lin, Geng, and Ali, M. Montaz. Max-k-cut by the discrete dynamic convexized method. *INFORMS Journal on Computing 25*, 1 (2013), 27–40.