The Dissertation Committee for Ethan Russell Elenberg
certifies that this is the approved version of the following dissertation:

Graph Analytics and Subset Selection Problems in Machine Learning

Committee:

Sriram Vishwanath, Cosupervisor

Alexandros G. Dimakis, Cosupervisor

Sanjay Shakkottai

Constantine Caramanis

Sahand Negahban

Graph Analytics and Subset Selection Problems in Machine Learning

by

Ethan Russell Elenberg

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2018

To my parents.

Acknowledgments

I have had the great pleasure of working with two amazing co-advisors, Sriram Vishwanath and Alex Dimakis. They helped me navigate the highs and lows of research for the past 6 years. Both of them have encouraged me throughout many professional and personal struggles. Their guidance has shaped the way I approach challenging problems. Sriram's strong intuition, breadth of technical knowledge, and ability to assemble a great team have helped me tremendously. I thank him for allowing me to pursue ideas across a wide range of interests, and for our many engaging discussions on theory, system design, research strategies, and entrepreneurship. It is inspiring to witness Alex develop an initial idea into a minimal working example and finally into polished research contribution. I admire his drive, skill, passion, and search for genuine understanding of a topic. He has been instrumental in helping me identify impactful subject areas, dissect new research problems, and focus on concrete goals. During our countless daytime meetings and late night calls, he always welcomed discussions across any level of abstraction.

I am also grateful for many collaborations with faculty from other universities: Sahand Negahban (the second half of this dissertation grew out of a series of meetings during his visit to UT Austin in Spring of 2016), Amin Karbasi, and Moran Feldman. Their insights have improved my eye for interesting theoretical

challenges and principled experimental design. I have learned from many excellent professors during my time at The University of Texas, including Gustavo de Veciana, Rachel Ward, Jeff Andrews, Sanjay Shakkottai, Al Bovik, Constantine Caramanis, and Sujay Sanghavi. I thank my committee members in particular for their helpful discussions and suggestions. I would also like to thank everyone I worked with during summer internships at 3 wonderful companies: Apple, MIT Lincoln Laboratory, and Twitter. In particular, Wassim El-Hassan, Dan Rabideau, Jim Melody, Jarred Barber, Venu Satuluri, Krishna Kamath, and Yao Wu all helped shape my research path.

During my time at UT Austin, I was very fortunate to collaborate with many excellent students: Karthik, Rajiv, Michael, Jonathan, and Anurag. Karthik was an early mentor and friend who demonstrated the value of diving into rich theoretical concepts. Rajiv's complementary perspective was invaluable during many brainstorming and writing sessions. I value their technical expertise, company, and friendship. I would like to thank my numerous current and former group members, in particular: Kumar, Dimitris, Ioannis, Ankit, Megas, Avhishek, Deepjyoti, Murat, Erik, Rajat, Shanshan, Rashish, Muryong, Yitao, Hardik, Chris Snyder, Ajil, Matt, Eirini, Justin, and Dave. All of them have made our group meetings fun, insightful, and conducive to candid discussions. I am also thankful for the friendship and camaraderie of several students throughout the Wireless Networking & Communications Group and other areas of the ECE department: Andrew Thornburg, Andrew Kerns, Chris Brennan, Xin, Jian, Todd, Lucas, Abishek, Derya, Sara, Soumya, and Jessica. It has been a pleasure discussing research ideas, homework

assignments, software packages, and random thoughts about food, politics, music, and technology. The ECE Department staff: Karen Little, Lauren Bringle, Apipol Piman, Jaymie Udan, Melanie Gulick, and Melody Singleton have always provided timely and expert administrative help.

I had many remarkable undergraduate professors at The Cooper Union, including Fred Fontaine, Sam Keene, Tim Hoerning, Toby Cumberbatch, Om Agrawal, and Robert Uglesich. Their contagious passion, rigor, and encouragement, combined with Peter Cooper's gift of free education, heavily influenced my decision to pursue a Ph.D. I am also thankful to have kept in touch with my undergraduate classmates, project partners, and extended Cooper family: Calvin, Kevin, Joe, Sam, John, Rafi, Dale, Igor, Gabe, Katie, and Abhay. I was fortunate to connect (and reconnect) with several alumni after our time at Cooper: Adam, Eugene, Andrew Massimino, Krishna, Julia, Andrew Leader, and Barry. I am glad to have met and played with dozens of musicians across New Jersey, New York, and Austin throughout the years. It was often difficult to schedule practices, but I valued all of our time together in basements, recording studios, venues, parks, and street corners.

Finally, I would like to thank my parents George and Laurie for their love, advice, and unwavering support. They instilled in me a driving work ethic, a passion for learning new things, and a desire to understand and improve on my mistakes. Their incredible sacrifices have enabled me to prioritize academics for most of my life, and they have always been available to offer comfort and congratulations.

Graph Analytics and Subset Selection Problems in Machine Learning

Ethan Russell Elenberg, Ph.D.
The University of Texas at Austin, 2018

Supervisors: Sriram Vishwanath
Alexandros G. Dimakis

In this dissertation we examine two topics relevant to modern machine learning research: 1) Subgraph counting and 2) High-dimensional subset selection. The former can be used to construct features for performing graph analytics. The latter has applications in sparse modeling such as feature selection, sparse regression, and interpretable machine learning. Since these problems become intractable for large datasets, we design efficient approximation algorithms for both tasks with data-dependent performance guarantees.

In the first part of the dissertation, we study the problem of approximating all three and four node induced subgraphs in a large graph. These counts are called the 3 and 4-profile, respectively, and describe a graph's connectivity properties. This problem generalizes graphlet counting and has found applications ranging from bioinformatics to spam detection. Our algorithms use the novel concept of graph profile sparsifiers: sparse graphs that can be used to approximate the full

profile counts for a given large graph. We obtain novel concentration results showing that graphs can be substantially sparsified and still retain good approximation quality for the global graph profile. We also study the problem of counting local and ego profiles centered at each vertex of the graph. These quantities embed every vertex into a low-dimensional space that characterizes the local geometry of its neighborhood. We introduce the concept of edge pivots and show that all local 3 and 4-profiles can be computed as vertex programs using compressed two-hop information. Our algorithms are local, distributed message-passing schemes and compute all graph profiles in parallel. We empirically evaluate these algorithms with a distributed GraphLab implementation, and show improvements over previous state-of-the-art in experiments scaling up to 640 cores on Amazon EC2.

In the second part we shift to the problem of subset selection: for example, selecting a few features from a large feature set. Motivated by the need for interpretable, nonlinear regression models for high-dimensional data, we draw a novel connection between this and submodular maximization. We extend an earlier concept of weak submodularity from the setting of sparse linear regression to a broad class of objective functions, including generalized linear model likelihoods. We then show that three greedy algorithms (Oblivious, Forward Stepwise, and Orthogonal Matching Pursuit) perform within a constant factor from the best possible subset. Our methods do not require any statistical modeling assumptions and allow direct control over the number of obtained features. This contrasts with other work that uses regularization parameters to control sparsity only implicitly. Our proof technique connecting convex analysis and submodular set function the-

ory may be of independent interest for other statistical learning applications that have combinatorial structure.

In the third part, we consider the problem of explaining the predictions of a given black-box classifier. For example, why does a deep neural network assign an image to a particular class? We cast interpretability of black-box classifiers as a subset selection problem and propose to solve it with an efficient streaming algorithm. We provide a constant factor approximation guarantee for this algorithm in the case of a random stream order and a weakly submodular objective function. This is the first such theoretical guarantee for this general class of functions, and we also show that no such algorithm exists for a worst case stream order. Our algorithm obtains similar explanations of Inception V3 predictions 10 times faster than the state-of-the-art LIME framework.

Table of Contents

List of Figures

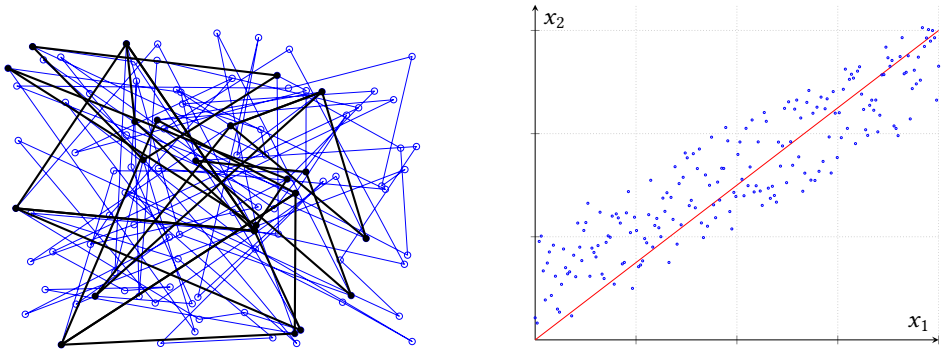
Chapter 1

Introduction

We consider problems in two areas of machine learning: 1) Algorithms for counting and approximating graph profiles and 2) Optimizing weakly submodular set functions subject to cardinality constraints. Both topic areas present new challenges as we seek to analyze increasingly large datasets, since naïve solutions would simply enumerate every subset of data points up to a given size. In both cases we use specific assumptions, either conditions on the graph structure or dependency conditions among the set elements, to design natural approximation algorithms with data-dependent guarantees.

Figure 1.1 shows motivating examples for these two topics. Many large, structured datasets are represented naturally in the form of a graph for improved information storage and retrieval. For example, given the graph in Figure 1.1(a) one straightforward task is to count the number of triangles, shown in black. One can also ask a more detailed question: for each vertex v, how many triangles include v? To generalize Figure 1.1(a), we consider a problem called graph k-profiles which involves counting all induced subgraphs up to size k.

Figure 1.1(b) illustrates dimensionality reduction via a simple linear regression example: in this case the set of 2-dimensional blue points (x_1, x_2) is well-



(a) (b)

Figure 1.1: Simplified motivating examples for the two types of problems addressed in this dissertation. (a) Count the number of triangles in given graph, shown in black. Additionally, count the number of triangles incident on a single edge or vertex. If this is a social network graph, then this information can be used to help identify or classify different users. (b) Select a plane on which to project a high dimensional dataset, such that the data has some structure and we can infer a relationship between some of the variables. For example, find the best two variables to predict some third variable.

approximated by the 1-dimensional red line. If our goal is to predict a third variable x_3 from observed data, then this linear relationship suggests that one feature can be eliminated. To generalize Figure 1.1(b), we consider sparsity constrained (potentially nonlinear) optimization. For example, build a model using only 100 out of 1 million features. In truly large scale problem settings, the entire dataset cannot be processed all at once. Here, the general combinatorial challenge is to select a good subset of N items using only $o(N)$ memory.

These problems are complements of each other: if we think about graph profiles as designing graph features, the sparse optimization can be thought of as selecting features to form a meaningful model. Both of these problems quickly be-

come intractable on modern datasets, so our focus will be on designing efficient approximation algorithms with provable, data-dependent performance guarantees.

The following sections provide an outline of the main results in the dissertation.

1.1 Part 1: Efficient Graph Profile Counting

The first part of this dissertation focuses on a quantity called the k-profile, which is based on counting induced subgraphs of size k in a large graph. We define three different kinds of profiles: the *global*, *local*, and *ego* k-profile in Chapter 2. Subgraph counting is an important primitive for graph analytics tasks across many domains, such as spam detection [1], bioinformatics [2], and clustering [3]. Additionally, they are deeply connected to the emerging theory of graphons and graph limits [4–6]. In supervised learning problems, counts of certain subgraphs have been used to classify nodes [7], edges [8], subgraphs [9], and graphs [10]. In all of these applications, new features are extracted from graph-structured data. For the case of local k-profiles, for example, every vertex is embedded into a moderately sized feature space.

In many large-scale settings, data is distributed across multiple locations. Thus it is desirable to design algorithms that run on distributed graph engines like GraphLab PowerGraph [11], GraphX [12], or Pregel [13]. These frameworks use message-passing to extract complex information beyond simple edge queries.

Question 1. *Can we design efficient, distributed algorithms to compute local, global,*

and ego profiles on very large graphs?

We propose algorithms for counting ego 3-profiles and local k-profiles for $k = \{3, 4\}$. These algorithms use message-passing to count all local profiles concurrently, which avoids recalculating intermediate computations. They also fit within the Gather-Apply-Scatter framework common to most modern, distributed graph engines. Experiments show that with modest cluster sizes, our proposed algorithms run much faster than naïve, serial implementations.

For many applications, such as interactive, exploratory analysis, a data scientist might be satisfied with approximate answers if they enable faster running times and less memory consumption. One simple way to reduce computation is to construct a subsampled version of the graph with uniform edge sampling, *i.e.* keep every edge independently with some fixed probability p. From this, we show how to derive unbiased estimators for each entry in the k-profile. Then the problem becomes how to obtain good performance guarantees for this sparsifier.

Question 2. *How well does a global k-profile estimator based on uniform edge subsampling perform on real-world graphs?*

We characterize the performance of the above *k-profile sparsifiers* as a function of the number of subgraphs dependent on a single edge. This allows us to randomly discard most edges of the graph and still have k-profile estimates that are provably within a bounded error with high probability. Our analysis uses two types of concentration bounds: a result on deviations of multivariate polynomials,

and an information theoretic inequality for families of dependent random functions.

1.2 Part 2: Regression Guarantees via Weak Submodularity

In the next topic area, we turn our attention to a complementary problem. Sparsity is a fundamental design principle for nearly all machine learning models. Similar to Occam's razor, the guiding methodology is that simple models are easy to implement and generalize better to new data not included in the initial training set. Many example applications have very high-dimensional data: time series, multimedia, one-hot encoding of categorical features, etc. Thus, many problems such as sparse regression, matrix approximation, and dictionary selection share a common framework: given p items, select a set S which maximizes some function $f(\mathrm{S})$ subject to the constraint $|\mathrm{S}| \leq k \ll p$.

In general we cannot avoid the subset selection problem's combinatorial nature and can only solve it exactly by exhaustive search. Moreover, greedy algorithms are not optimal unless the set function f is modular.[1] In a sparse regression application, this corresponds to the unrealistic assumption of independent features. However, if f is *submodular*, a classical result shows that the greedy forward selection algorithm of Nemhauser [15] returns a set within a factor of $1 - 1/e$ from the optimum.

Question 3. *Can classical (and recent) results from combinatorial optimization be*

[1] Very recently, a more refined analysis [14] characterizes problem instances of monotone submodular maximization for which greedy algorithms indeed return the true optimum.

used in a more general setting by relaxing the definition of submodular set functions?

This question was answered partially by Das and Kempe [16] who defined a notion of *weak* submodularity.[2] They showed that in the case of linear regression, sparse eigenvalue conditions on the covariance matrix imply similar constant factor guarantees on the approximation ratio. However, the question remains largely open for many recent advances in algorithms and applications from submodular function optimization literature [17–19]. In the second part of this dissertation, we make progress toward the answer by extending the analytical framework of [16] to more general sparsity-constrained regression problems.

Question 4. *Explore the connection between combinatorial structure and standard assumptions in continuous optimization. In particular, can (weak) submodularity be used to provide guarantees for continuous problems such as sparse regression under Restricted Strong Convexity conditions?*

We answer this question positively by showing that any continuous objective function satisfying a version of Restricted Strong Convexity (along with a smoothness assumption) corresponds to a set function that satisfies weak submodularity. Using this connection, we obtain performance guarantees for three types of greedy algorithms (Oblivious, Forward Stepwise Selection, and Orthogonal Matching Pursuit) without making any other modeling assumptions. Specifically, we need not assume that a true sparse solution exists but can directly control

[2] This is one of many recent relaxations of submodularity. The reader is referred to Appendix C for a detailed discussion of the relationships among different parameterizations of non-submodular functions.

the sparsity of the output. Alternatives such as ℓ_1 regularization either require very strong assumptions or only control the sparsity level implicitly. First we show via weak submodularity that when the function satisfies RSC, the approximation ratio of each algorithm is lower bounded by a constant. Then we provide a general procedure to convert any approximation ratio guarantee into a bound on recovering the optimal sparse parameter. In other words (after adding some other standard statistical assumptions), convergence to the maximum function value also implies convergence to the argmax.

1.3 Part 3: Streaming Weak Submodularity

In some applications, we wish to interpret the predictions of a black-box neural network so that a data scientist, engineer, or domain expert can assess its performance. Explanations for machine learning models are believed to be crucial for adoption in many industries like healthcare, autonomous vehicles, and defense. They also help provide a framework to enforce regulations on fairness across demographic groups and to assign liability when a failure occurs. For example, the European Union's General Data Protection Regulation (GDPR), which took effect in May 2018, contains language designed to inform consumers about how their data is used to make automated decisions.[3]

However, each model evaluation can be expensive, *e.g.* if it is stored on the cloud and accessed through an API. One disadvantage of the standard greedy

[3]Presently, the GDPR's requirement of a "right to explanation" (and its enforceability in production machine learning systems) remains controversial among legal scholars [20, 21].

algorithm [15] is that it requires repeated access to each data element. This is undesirable for additional reasons in tasks such as large-scale data summarization, where the entire dataset cannot fit in main memory. We address these issues by considering a streaming version of the problem. Streaming algorithms make a constant number of passes (often only one) over the data and use sublinear space. Therefore, we examine streaming algorithms for maximizing weakly submodular functions, with applications to interpretability of black-box neural network classifiers.

Question 5. *When set elements arrive in a random (or adversarial) streaming order, what is the approximability of weak submodular maximization as a function of γ? Are algorithmic guarantees consistent with the $\gamma = 1$ submodular case, or is $0 < \gamma < 1$ fundamentally different?*

First, we obtain an impossibility result which shows that, even for $\gamma = 0.5$, no randomized streaming algorithm which uses sublinear memory can have a constant factor approximation. This is quite different from the case of $\gamma = 1$ where a worst-case approximation factor of $1/2 - \epsilon$ is achievable [22]. Next, we design and analyze a greedy, deterministic streaming algorithm for maximizing γ-weakly submodular functions which has an expected approximation ratio of $(1 - \epsilon)\gamma \cdot \left[4 + \frac{\gamma}{2} - 2\sqrt{\gamma + 4}\right]$. Here the expectation is with respect to a random stream order. The algorithm uses $O(\epsilon^{-1} k \log k)$ memory and does not require knowledge of γ to run. This is achieved by extending key algorithmic components of Sieve-Streaming [22], combined with a novel analysis.

Question 6. *Can streaming weak submodularity be used to establish performance guarantees in new application areas, such as interpretability of black-box classifiers?*

Finally, we provide a partial answer to the above question. We conduct an experimental evaluation of our algorithm for nonlinear sparse regression and interpretability of black-box neural network classifiers, namely Inception V3 [23]. We define a new subset selection problem similar to that of LIME [24] and apply our framework to approximately maximize this function. Experimentally, we find that our interpretability method produces explanations of similar quality to LIME and runs approximately 10 times faster.

1.4 Organization

Chapter 2 deals with the first part of the dissertation: algorithms and approximation guarantees for several k-profile problems on large-scale graphs. Chapter 3 deals with the concept of weak submodularity and its application to subset selection problems, with sparse logistic regression as a special case. Chapter 4 analyzes a streaming variant of weak submodular maximization, with applications to both sparse regression and black-box interpretability. Each chapter concludes with relevant directions for future work.

1.5 Notation

Next we collect some notation that will be used throughout this dissertation. Sets are represented by sans script fonts *e.g.* A, B. Vectors are represented

using lower case bold letters *e.g.* $\mathbf{x}, \mathbf{y}$, and matrices are represented using upper case bold letters *e.g.* $\mathbf{X}, \mathbf{Y}$. The i-th column of $\mathbf{X}$ is denoted $\mathbf{X}_i$. Non-bold face letters are used for scalars (*e.g.* j, M, r), graphs (*e.g.* G, H_1, $F_3(v)$) vertices and edges of a graph (*e.g.* $v_i, v_j \subseteq V$, $e_{ij} = (v_i, v_j) = v_i v_j \subseteq E$), and function names (*e.g.* $f(\cdot)$). The neighborhood set of a vertex v is denoted $\Gamma(v)$, and the ego graph of v is denoted $N(v)$. The transpose of a vector or a matrix is represented by $\top$ *e.g.* $\mathbf{X}^\top$. For any vector $\mathbf{v}$, define $\|\mathbf{v}\|_{2,k} := \sqrt{\sum_{i=1}^{k} v_{(i)}^2}$, where $v_{(i)}$ represent the order statistics of $\mathbf{v}$ in decreasing order. Define $[p] := \{1, 2, \ldots, p\}$. For simplicity, we assume a set function defined on a ground set of size p has domain $2^{[p]}$. For singleton sets, we write $f(j) := f(\{j\})$.

1.6 Publications (Related to Dissertation)

[1] Ethan R. Elenberg, Karthikeyan Shanmugam, Michael Borokhovich, and Alexandros G. Dimakis. Beyond Triangles: A Distributed Framework for Estimating 3-profiles of Large Graphs. In *KDD*, pages 229–238, 2015.

[2] Ethan R. Elenberg, Karthikeyan Shanmugam, Michael Borokhovich, and Alexandros G. Dimakis. Distributed Estimation of Graph 4-profiles. In *WWW*, pages 483–493, 2016.

[3] Ethan R. Elenberg, Rajiv Khanna, Alexandros G. Dimakis, and Sahand Negahban. Restricted Strong Convexity Implies Weak Submodularity. In *NIPS Workshop on Learning in High Dimensions with Structure*, 2016.

[4] Ethan R. Elenberg, Rajiv Khanna, Alexandros G. Dimakis, and Sahand Negahban. Restricted Strong Convexity Implies Weak Submodularity. *The Annals of Statistics*, 2018 (to appear).

[5] Ethan R. Elenberg, Alexandros G. Dimakis, Moran Feldman, and Amin Karbasi. Streaming Weak Submodularity: Interpreting Neural Networks on the Fly. In *NIPS*, pages 4047–4057, 2017.

Note: The dissertation author is the primary author of all of these papers. *arXiv* versions of almost all of these papers are available. They can also be accessed at https://eelenberg.github.io/.

Chapter 2

Graph Profiles: Algorithms and Approximation Guarantees

2.1 Introduction

In this chapter,[1] we discuss several variations of the graph k-profile problem. We propose distributed algorithms for computing local k-profiles that fall within the Gather-Apply-Scatter framework. Then we describe a general scheme for approximate global k-profile counting, and analyze it using two types of concentration inequalities. Finally, we show the effectiveness of our distributed algorithms and sampling schemes through a series of experiments on multicore, shared memory platforms as well as distributed computing clusters.

Graph k-profiles are local statistics that count the number of small subgraphs in a big graph. They are a natural generalization of triangle counting and are increasingly popular for several problems in big graph analytics. Globally,

[1] The material in this chapter is based on the following conference papers: [25] E. R. Elenberg, K. Shanmugam, M. Borokhovich, and A. G. Dimakis. Beyond Triangles: A Distributed Framework for Estimating 3-profiles of Large Graphs. In *KDD*, pages 229–238, 2015. [26] E. R. Elenberg, K. Shanmugam, M. Borokhovich, and A. G. Dimakis. Distributed Estimation of Graph 4-profiles. In *WWW*, pages 483–493, 2016. The dissertation author's primary contributions are derivation of unbiased k-profile estimators, application of concentration inequalities, algorithm implementations, and analysis of experiments. The dissertation author also assisted with other contributions and is the primary contributor of these papers.

they form a concise graph description that has found several applications for the web [1, 27] as well as social and biological networks [2, 28]. Furthermore, as we explain, the *local* profile of a vertex is an embedding in a low-dimensional feature space that reveals local structural information. Mathematically, k-profiles are of significant recent interest since they are connected to the emerging theory of graph homomorphisms, graph limits, and graphons [4,5,28]. Estimating k-profiles of big graphs is a topic that has received attention from several communities recently (*e.g.* see [3, 25, 26, 28–32] and references therein).

2.1.1 Graph Profile Definitions

There are 4 possible graphs on 3 vertices, labeled $H_0, \ldots, H_3$ in Figure 2.1(a). The *global* 3-profile of a graph $G(V,E)$ is a vector having one coordinate for each distinct H_i that counts how many times that H_i appears as an induced subgraph of G. For example, the graph $G = K_4$ (the complete graph on 4 vertices) has the 3-profile $[0,0,0,4]$ since it contains 4 triangles and no other (induced) subgraphs. The graph C_5 (the cycle on 5 vertices, *i.e.* a pentagon) has the 3-profile $[0,5,5,0]$. Note that the sum of the k-profile is always $\binom{|V|}{k}$, the total number of subgraphs. In this chapter we are also interested in the significantly more challenging problem of estimating 4-profiles. Figure 2.1(b) shows the 11 possible graphs on 4 vertices,[2] labeled as F_i, $i = 0, \ldots, 10$. Given a big graph $G(V,E)$ and $k = \{3,4\}$, we are interested in estimating the *global* k-profile, *i.e.* count how many times each H_i

[2] Actually there are 17 local subgraphs when considering vertex automorphisms. This is discussed in Section 2.2.3 and [26] in detail, but for clarity we will ignore vertex automorphisms in this introductory section.

and F_i appears as an induced subgraph of G.

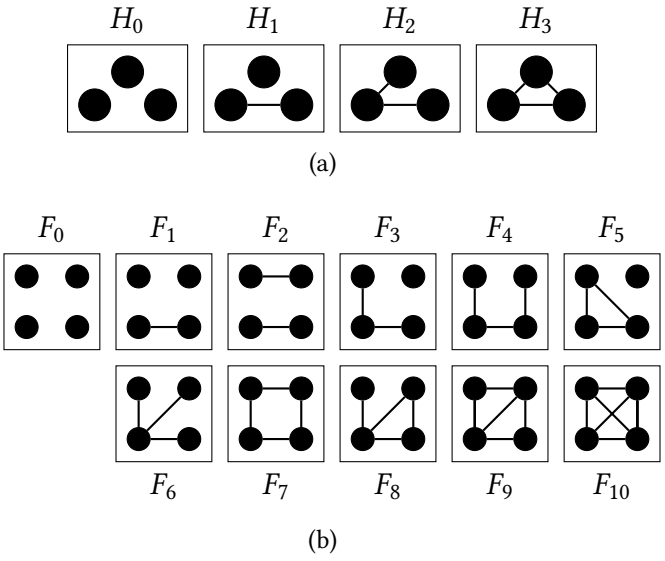


Figure 2.1: (a) The 4 possible non-isomorphic graphs on 3 vertices used to calculate the 3-profile of a graph G. The 3-profile counts how many times each H_i appears in G. (b) The 11 non-isomorphic graphs on 4 vertices used to calculate the 4-profile of a graph.

In addition to global graph statistics, we are interested in *local* k-profiles: given a specific vertex v_0, the local 3-profile (4-profile) of v_0 is a 4-dimensional (11-dimensional) vector, with each coordinate i counting how many induced H_i's (F_i's) contain v_0. In Figure 2.2 we show an example of the local 4-profile of a vertex. The local k-profile can be seen as an embedding of each vertex in a low-dimensional space that characterizes the local geometry of its neighborhood: vertices that connect different clusters will have different local 4-profiles compared to those that are only part of one dense cluster. A very naïve estimation of 4-profiles requires examining $\binom{n}{4}$ possible subgraphs. Furthermore, for estimating each local 4-profile

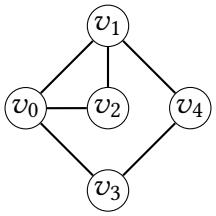


Figure 2.2: An example for local profiles. The global 3-profile is $[0,3,6,1]$. The global 4-profile is $[0,0,0,0,2,0,0,1,2,0,0]$. The ego 3-profile of v_0 is $[0,1,0,0]$. The local 4-profile of v_0 is $[0,0,0,0,1,0,0,1,2,0,0]$. The first 1 in the profile corresponds to the subgraph F_4. Notice that v_0 participates in only one F_4, jointly with vertices v_2, v_3, v_4.

independently, this computation has to be repeated n times, once for each vertex. Note that the local 4-profiles may be rescaled and added together to obtain the global 4-profile. Since some of the 4-profile subgraphs are disconnected (like F_0, F_1, F_5), local 4-profiles contain information beyond the local neighborhood of a vertex. Therefore, in a distributed setting, it seems that global communication is required.

Finally, we also consider the problem of calculating the *ego* 3-profile for each vertex, *e.g.* v_0. This is the 3-profile of the graph $N(v_0)$ *i.e.* the neighbors of v_0, also called the ego graph of v_0. The ego 3-profile of v_0 can be seen as a projection of the vertex into a coordinate system [28]. This is a very interesting idea of viewing a big graph as a collection of small dense graphs, in this case the ego graphs of the vertices. Note that calculating the ego 3-profiles for a set of vertices of a graph is different (in fact, significantly harder) than calculating local 3-profiles. Indeed, we show that the ego 3-profile can be computed by counting a subset of the local 4-profile. Figure 2.2 also shows an example of the ego 3-profile of a vertex.

2.1.2 Contributions

For the rest of this chapter we describe three contributions in this area, which are summarized as follows:

Algorithms: Our first contribution deals with efficiently designing *distributed* algorithms for estimating k-profiles on large graphs. We rely on the Gather-Apply-Scatter model used in GraphLab PowerGraph [11] but, more generally, our algorithm fits the architecture of most graph engines. This restrictive setting does not allow communication between nonadjacent vertices, a key component of previous centralized, shared-memory approaches. We introduce the concept of *edge pivoting* which allows us to collect 2-hop information without maintaining an explicit 2-hop neighborhood list at each vertex. This enables the computation of all the local 3-profiles in parallel. Each edge requires only information from its endpoints and each vertex only computes quantities using data from incident edges. For the problem of ego 3-profiles, we show how to calculate them by combining edge pivot equations and local 4-clique counts.

We also show that, surprisingly, very limited global information is sufficient to calculate the local 4-profile of a vertex and that it can be reused to calculate all the local 4-profiles in parallel. Specifically, we introduce a distributed algorithm to estimate all the local 4-profiles and the global 4-profile of a big graph. Our algorithm operates by having each vertex first perform local message-passing to its neighbors and then solve a novel system of equations for the local 4-profile. Focusing on a vertex v_0, the first easy step is to calculate its local 3-profile. It can be

shown that the local 3-profile combined with the full two-hop connectivity information is sufficient to estimate the local 4-profile for each vertex v_0. This is not immediately obvious, since naïvely counting the 3-path (an automorphism of F_4) would require 3-hop connectivity information. However, we show that less information needs to be communicated. Specifically, we prove that the triangle list combined with what we call the *two-hop histogram* is sufficient: for each vertex v_i that is 2-hops from v_0, we only need the *number of distinct paths* connecting it to v_0, not the full two hop neighborhood. If the two-hop neighborhood is a tree, this amounts to no compression. However, for real graphs the two-hop histogram saves a factor of 3x to 5x in communication (see the experiments in Section 2.4). This enables an even more significant running time speedup of 5–10 times on several distributed experiments using 12–20 compute nodes.

Graph Profile Sparsification: Our second innovation is a provable edge subsampling scheme: we establish sharp concentration results for estimating the entire global k-profile of a graph. This allows us to randomly discard most edges of the graph and still have k-profile estimates that are provably within a bounded error with high probability. One idea that originated from triangle counting [33, 34] is to first perform random subsampling of edges to create a sparse graph called a *triangle sparsifier*. Then count the number of triangles in the sparse graph and rescale appropriately to estimate the number in the original graph. We present two proof techniques, based on [35] and [36], that show the randomly sparsified graph has a profile sufficiently concentrated around its expectation.

System Implementation: Finally, we implement our algorithms in GraphLab PowerGraph [11] and perform several experiments scaling up to 640 cores on Amazon EC2. We present results on both overall runtimes and network communication on multicore and distributed systems. We find that our algorithm can estimate the 3-profile of a graph in approximately the same time as triangle counting. Specifically, we compare against the PowerGraph triangle counting routine and find that it takes us only 1%-10% more time to compute the full 3-profile. For the significantly harder problem of ego 3-profiles, we were able to compute (in parallel) the 3-profiles of up to 100,000 ego graphs in the timescale of several minutes. We compare our parallel ego 3-profile algorithm to a simple sequential algorithm that operates on each ego graph sequentially and show tremendous scalability benefits, as expected. The benefits of two-hop histogram compression and sparsification allow us to compute the global and local 4-profiles of very large graphs. For example, for a graph with 5 million vertices and 40 million edges we estimated the global 4-profile in less than 10 seconds. For computing all local 4-profiles on this graph, the previous state-of-the-art [30] required 1200 seconds while our distributed algorithm required less than 100 seconds.

2.1.3 Related Work

In this section, we describe several related topics and discuss differences in relation to our work.

Graph Subsampling and Concentration Inequalities: Random edge subsampling is a natural way to quickly obtain estimates for graph parameters. For the case of triangle counting such graphs are called triangle sparsifiers [34]. Related ideas were explored in the Doulion algorithm [33, 34, 37] with increasingly strong concentration bounds. The recent work by Ahmed *et al.* [38] develops subgraph estimators for clustering coefficient, triangle count, and wedge count in a streaming subsampled graph. Other recent work [39–42] uses random sampling to estimate parts of the 3 and 4-profile. These methods do not account for a distributed computation model and require more complex sampling rules. As discussed, our theoretical results build on [34] to define the first 3-profile sparsifiers, sparse graphs that are *a fortiori* triangle sparsifiers.

Concentration inequalities for the number of triangles in a random graph have been studied extensively. The standard method of martingale bounded differences (McDiarmid's inequality) is known to yield weak concentrations around the mean for this problem. The breakthrough work of Kim and Vu [35] provides superior asymptotic bounds by analyzing the concentration of multivariate polynomials. This was later improved and generalized in [43], and applied to subsampled triangle counting in [34]. The results in Section 2.3.2 follow this proof technique, while the analysis in Section 2.3.3 uses a different, information theoretic technique called read-k functions [36] that produces sharper concentration results for practical problem sizes.[3]

[3]Even though concentrations using Kim-Vu become tighter asymptotically, this happens for graphs with well over 10^{13} edges (see also Figure 2.6).

Triangle Counting in Graph Engines: Graph engines (*e.g.* Pregel, GraphLab, Galois, GraphX, see [44] for a comparison) are frameworks for expressing distributed computation on graphs in the language of vertex programs. Triangle counting algorithms [27, 45] form one of the standard graph analytics tasks for such frameworks [11,44]. In [46], the authors *list* triangles efficiently by partitioning the graph into components and processing each component in parallel. Typically it is much harder to perform graph analytics over the MapReduce framework, but some recent work [47,48] has used clever partitioning and provided theoretical guarantees for triangle counting.

Frequent Subgraph Discovery: The general problem of finding frequent subgraphs, also known as motifs or subgraph isomorphisms, is to find the number of occurrences of a small query graph within a larger graph. Typically frequent subgraph discovery algorithms offer pruning rules to eliminate false positives early in the search [49–51]. This is most applicable when subgraphs have labeled vertices or directed edges. For these problems, the number of unique isomorphisms grows much larger than in our application.

In [28], subgraphs were queried on the ego graphs of users. While enumerating all 3-sets and sampling 4-sets of neighbors can be done in parallel, forming the ego subgraphs requires checking for edges between neighbors. This suggests that a graph engine implementation would be highly preferable over an Apache Hive system. Our algorithms simultaneously compute the ego subgraphs and their profiles, reducing the amount of communication between nodes. Algorithm 3 is

suitable for both NUMA multicore and distributed architectures, but our implementation focus in this chapter is on GraphLab.

Graphlets and Applications: First described in [2], graphlets generalize the concept of vertex degree to include the connected subgraphs a particular vertex participates in with its neighbors. Unique graphlets are defined at a vertex based on its degree in the subgraph. Graphlet frequency distributions (GFDs) have proven useful in the fields of computational neuroscience [52] and bioinformatics. Specifically, GFD analysis of protein interaction networks helps to design improved generative models [53], accurate similarity measures [2], and better features for classification [10]. The term *graphlets* typically refers to only connected subgraphs, but in some examples such as [54], the authors analyze neuronal networks using *all* global 4-subgraphs.

Subgraph Counting: Fast matrix multiplication has been used for certain types of subgraph counting. Alon *et al.* proposed a cycle counting algorithm which uses the trace of a matrix power on high degree vertices [55]. Related approximation schemes [37] and randomized algorithms [29] depend on centralized architectures and computing matrix powers of very large matrices.

Previous systems of equations relating clique counts to other 4-subgraphs appear in [29, 30, 56–59]. However, these are applied in a centralized setting and depend on information collected from nonadjacent vertices. In this work, we use additional equations to solve the same system by sharing only local information

over adjacent vertices. We evaluate our algorithm against ORCA [30], a centralized 4-graphlet counting algorithm, as well as its GPU implementation [60]. Notice that while ORCA calculates only connected 4-subgraphs, our algorithm calculates all the connected and the disconnected 4-subgraphs for each vertex.

Concurrent with the writing and subsequent publication of these results in [26], a parallel algorithm for 4-subgraph counting was introduced in [59]. Our algorithm differs by working within GraphLab PowerGraph's Gather-Apply-Scatter framework instead of the native, multithreaded C++ implementation of [59]. In terms of empirical performance, both our work and [59] show similar running time improvements of one order of magnitude over ORCA. A more detailed comparison would depend on the hardware and datasets used. More importantly, our work focuses on a distributed (as opposed to multicore parallel) framework, and for our setting minimizing communication is critical. Our theoretical results are significantly different from [59] and may be useful in improving that system also. Specifically, [59] explicitly counts the number of 4-cycles (F_7 in Figure 2.1(b)) whereas we show that it is possible to use only *two-hop histograms* instead. This results in less communication overhead, but this benefit is perhaps not as significant for shared-memory multicore platforms. Our second theoretical result, the novel sparsification concentration bounds, can be used for any subgraph estimation algorithm and quantifies a provable tradeoff between speed and accuracy.

2.2 Distributed Algorithms

In this section we introduce novel, distributed algorithms for estimating the local k-profiles of massive graphs for $k = \{3, 4\}$, as well as ego 3-profiles. These local vectors can be used to obtain the global profiles (by rescaled addition as we discuss in the sequel). Our algorithms can be applied independently of the edge sampling described in Section 2.3. We rely on the Gather-Apply-Scatter (GAS) model used in GraphLab PowerGraph (but, more generally, our algorithm fits the architecture of most graph engines). A distributed algorithm in this framework has 3 main phases: Gather, Apply, and Scatter. Every vertex and edge has stored data which is acted upon. During the Gather phase, a vertex can access all its adjacent edges and neighbors and *gather* data they possess, *e.g.* neighbor ID, using a custom reduce operation $\oplus$ (*e.g.* addition, concatenation). The accumulated information is available for a vertex at the next phase, Apply, in which it can change its own data. In the final Scatter phase, every edge sees the data of its (incident) vertices and operates on it to modify the edge data. All nodes start each phase simultaneously, and if needed, the whole GAS cycle is repeated until the algorithm's completion.

We introduce the concept of *edge pivoting* which allows us to collect 2-hop information without maintaining an explicit 2-hop neighborhood list at each vertex. This enables the computation of all the local k-profiles in parallel. Each edge requires only information from its endpoints and each vertex only computes quantities using data from incident edges. For the problem of ego 3-profiles, we show how to calculate them by combining edge pivot equations and local clique counts.

The key to our approach is to identify subgraphs at a vertex based on the degree with which it participates in the subgraph. From the perspective of a given vertex v, there are actually six distinct 3 node subgraphs up to isomorphism as given in Figure 2.3(a). Let $n_{0,v}, n^e_{1,v}, n^d_{1,v}, n^e_{2,v}, n^d_{2,v}$, and $n_{3,v}$ denote the corresponding local subgraph counts at v. We will first outline an approach that calculates these counts and then add across different vertex perspectives to calculate the final 4 scalars ($n_{2,v} = n^e_{2,v} + n^c_{2,v}$ and $n_{1,v} = n^e_{1,v} + n^d_{1,v}$). It is easy to see that the global counts can be obtained from these local counts by summing across vertices:

$$n_i = \frac{1}{3}\left(\sum_{v \in V} n_{i,v}\right), \ \forall i. \tag{2.1}$$

2.2.1 Distributed Local 3-profile

We will now give our approach for calculating the local 3-profile counts of $G(V, E)$ using only local information combined with $|V|$ and $|E|$.

Scatter: We assume that every edge (v, a) has access to the neighborhood sets of both v and a, *i.e.* $\Gamma(v)$ and $\Gamma(a)$. Therefore, intersection sizes are first calculated at every edge, *i.e.* $|\Gamma(v) \cap \Gamma(a)|$. Each edge computes the following scalars and stores them:

$$\begin{aligned}
n^e_{1,va} &= |V| - (|\Gamma(v)| + |\Gamma(a)| - |\Gamma(v) \cap \Gamma(a)|), \\
n^c_{2,va} &= |\Gamma(v) \backslash \{\Gamma(a) \cup a\}| = |\Gamma(v)| - |\Gamma(v) \cap \Gamma(a)| - 1, \\
n^e_{2,va} &= |\Gamma(a)| - |\Gamma(v) \cap \Gamma(a)| - 1 = n^c_{2,av}, \\
n_{3,va} &= |\Gamma(v) \cap \Gamma(a)|.
\end{aligned} \tag{2.2}$$

The computational effort at every edge is at most $O(d_{\max})$, where $d_{\max}$ is the maximum degree of the graph, for the neighborhood intersection size.

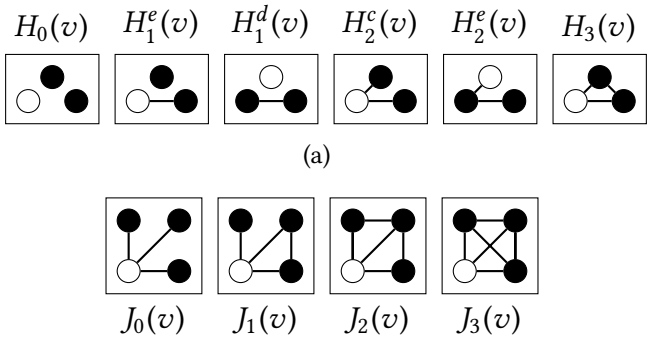


Figure 2.3: Naming conventions for local subgraphs throughout this chapter: unique (a) 3-subgraphs, (b) ego 3-subgraphs, and (c) 4-subgraphs from the perspective of the white vertex v. Note that $J_0(v)$ is the same as $F_6^{'}(v)$. F_8 is the only subgraph with a third vertex automorphism $F_8^{''}$ because no other subgraph contains vertices with 3 different degrees.

Gather: In the next round, vertex v "gathers" the above scalars as follows:

$$
\begin{aligned}
n_{2,v}^e &= \sum_{a\in\Gamma(v)} n_{2,va}^e, \;\; n_{1,v}^e = \sum_{a\in\Gamma(v)} n_{1,va}. \\
n_{2,v}^c &\overset{a}{=} \frac{1}{2}\sum_{a\in\Gamma(v)} n_{2,va}^c, \;\; n_{3,v} \overset{b}{=} \frac{1}{2}\sum_{a\in\Gamma(v)} n_{3,va}. \\
n_{1,v}^d &\overset{c}{=} |E| - |\Gamma(v)| - n_{3,v} - n_{2,v}^e. \\
n_{0,v} &= \binom{|V|-1}{2} - n_{1,v} - n_{2,v} - n_{3,v}.
\end{aligned}
\tag{2.3}
$$

Here, relations (a) and (b) are because triangles and wedges from center are double counted. (c) comes from noticing that each triangle and wedge from endpoint excludes an extra edge from forming $H_1^d(v)$. In this Gather stage, the communication complexity is $O(M)$ where it is assumed that $\Gamma(v)$ is stored over M different machines. The corresponding distributed algorithm is described in Algorithm 1.

Algorithm 1 3-PROF

Input: Graph $G(V, E)$ with $|V|$ vertices, $|E|$ edges
Gather: For each vertex v union over edges of the 'other' vertex in the edge, $\cup_{a\in\Gamma(v)} a = \Gamma(v)$.
Apply: Store the gather as vertex data `v.nb`, size automatically stored.
Scatter: For each edge e_{va}, compute and store scalars in (2.2).
Gather: For each vertex v, sum edge scalar data of neighbors
$\mathtt{g} \leftarrow \sum_{(v,a)\in\Gamma(v)} \mathtt{e.data}$.
Apply: For each vertex v, calculate and store the quantities described in (2.3).
return `[v: v.n0 v.n1 v.n2 v.n3]`

2.2.2 Distributed Ego 3-profile

In this section, we give an approach to compute ego 3-profiles for a set of vertices $\mathsf{V} \subseteq V$ in G. For each vertex v, the algorithm returns a 3-profile corresponding to that vertex's ego $N(v)$, a subgraph induced by the neighborhood set $\Gamma(v)$, including edges between neighbors and excluding v itself. Formally, our goal is to compute $\{\mathbf{n}_3(N(v))\}_{v\in\mathsf{V}}$. Clearly, this can be accomplished in two steps repeated serially on all $v \in \mathsf{V}$: first obtain the ego subgraph $N(v)$ and then pass it as input to Algorithm 1, summing over the ego vertices $\Gamma(v)$ to get a global count. The serial implementation is provided in Algorithm 2. We note that this was essentially done in [28], where ego subgraphs were extracted from a common graph

separately from 3-profile computations.

Instead, Algorithm 3 provides a parallel implementation which solves the problem by finding cliques in parallel for all $v \in V$. The main idea behind this approach is to realize that calculating the 3-profile on the induced subgraph $N(v)$ is exactly equivalent to computing specific 4-node subgraph frequencies among v and 3 of its neighbors, enumerated as $J_i(v)$, $0 \leq i \leq 3$ in Figure 2.3(b). Now, the aim is to calculate $J_i(v)$'s, effectively part of a local 4-profile.

Scatter: We assume that every edge (v, a) has already computed the scalars from (2.2). Additionally, every edge (v, a) also computes the list $\mathcal{N}_{va} = \Gamma(v) \cap \Gamma(a)$ instead of only its size. The computational complexity is still $O(d_{\max})$.

Gather: First, the vertex "gathers" the following scalars, forming three *edge pivot equations* in unknown variables $J_i(v)$:

$$\begin{aligned} \sum_{a \in \Gamma(v)} \binom{n^c_{2,va}}{2} &= 3J_0(v) + J_1(v) \\ \sum_{a \in \Gamma(v)} \binom{n_{3,va}}{2} &= J_2(v) + 3J_3(v) \\ \sum_{a \in \Gamma(v)} n^c_{2,va} n_{3,va} &= 2J_1(v) + 2J_2(v) \end{aligned} \tag{2.4}$$

By choosing two subgraphs that the edge (v, a) participates in, and then summing over neighbors a, these equations gather implicit connectivity information 2 hops away from v. However, note that there are only three equations in four variables and we must count one of them directly, namely the number of 4-cliques $J_3(v)$. Therefore, at the same gather step, the vertex also creates the list

$\mathcal{CN}_v = \bigcup_{a\in\Gamma(v),p\in\mathcal{N}_{va}}(a,p)$. Essentially, this is the list of edges in the subgraph induced by $\Gamma(v)$. This requires worst case communication proportional to the number of edges in $N(v)$, independent of the number of machines M.

Scatter: Now, at the next scatter stage, each edge (v,a) accesses the pair of lists $\mathcal{CN}_v, \mathcal{CN}_a$. Each edge (v,a) computes the number of 4-cliques it is a part of, defined as follows:

$$n_{4,va} = \sum_{i,j\in\Gamma(v)\cap\Gamma(a)} \mathbf{1}\left((i,j)\in\mathcal{CN}_v\right). \tag{2.5}$$

This incurs computation time of $|\mathcal{CN}_v|$.

Gather: In the final gather stage, every vertex v accumulates these scalars to get $J_3(v) = \frac{1}{3}\sum_{a\in\Gamma(v)} n_{4,va}$ requiring $O(M)$ communication time. As in the previous section, the scaling accounts for extra counting. Finally, the vertex solves the equations (2.4) using $J_3(v)$.

Algorithm 2 EGO-SER

Input: Graph $G(V,E)$ with $|V|$ vertices, $|E|$ edges, set of ego vertices V
for $v \in \mathsf{V}$ **do**
  Signal v and its neighbors.
  Include an edge if both its endpoints are signaled.
  Run Algorithm 1 on the graph induced by the neighbors and edges between them.
end for
return `[v: vego.n0 vego.n1 vego.n2 vego.n3]`

2.2.3 Distributed Local 4-profile

In this section, we describe 4-PROF, our algorithm for computing the exact 4-profiles in a distributed manner. To the best of our knowledge, this is the first

Algorithm 3 EGO-PAR

Input: Graph $G(V,E)$ with $|V|$ vertices, $|E|$ edges, set of ego vertices V
Gather: For each vertex v union over edges of the 'other' vertex in the edge, $\cup_{e_{va}} a = \Gamma(v)$.
Apply: Store the gather as vertex data `v.nb`, size automatically stored.
Scatter: For each edge e_{va}, compute and store as edge data:
  Scalars in (2.2).
  The list $\mathcal{N}_{va}$.
Gather: For each vertex v, sum edge data of neighbors:
  Accumulate LHS of (2.4).
  $\mathrm{g}.\mathcal{CN} \leftarrow \mathrm{g}.\mathcal{CN} \cup \mathcal{N}_{va}$.
Apply: Obtain $\mathcal{CN}_v$ and equations in (2.4) using the scalars and $\mathrm{g}.\mathcal{CN}$.
Scatter: Scatter $\mathcal{CN}_v, \mathcal{CN}_a$ to all edges (v,a).
  Compute $n_{4,va}$ as in (2.5).
Gather: Sum edge data $n_{4,va}$ of neighbors at v.
Apply: Compute $J_3(v)$.
return `[v: vego.n0 vego.n1 vego.n2 vego.n3]`

distributed algorithm for calculating 4-profiles. The key insight is to cast existing and novel equations into the GraphLab PowerGraph framework [11] to get implicit connectivity information about vertices outside the 1-hop neighborhood. Specifically, we construct the local 4-profile from local 3-profile, local 4-clique count, and additional histogram information which describes the number of paths to all 2-hop neighbors.

Theorem 2.2.1. *There is a distributed algorithm that computes the exact local* 4*-profile of a graph given each vertex has stored its local* 3*-profile, triangle list, and* 2*-hop histogram.*

Note that similar to (2.1), the local 4-profiles at each vertex can be added and appropriately rescaled (using the symmetries of each subgraph, also called

automorphism orbits [2]) to obtain the global 4-profile.

4-PROF solves a slightly larger problem of keeping track of counts of 17 unique subgraphs up to vertex automorphism (see Figure 2.3(c)). We will describe a full rank system of equations which is sufficient to calculate the local 4-profile at every $v \in V$. The remainder of this subsection explains each component of 4-PROF. These separate routines are combined efficiently in Algorithm 4 to calculate the local 4-profile in a small number of GAS cycles.

Edge Pivot Equations: The majority of our equations relate the local 4-profile to neighboring local 3-profiles with *edge pivots* as described earlier. At each vertex v, each combinatorial equation relates a linear combination of the local 4-subgraph counts to the count of a pair of 3-subgraphs sharing an edge va. Some of these equations appear in a centralized setting in previous literature ([29, 30, 57, 59]). In our algorithm, the 3-subgraph pair count accumulates at v as all incident edges va *pivot* over it. The edges fixed by a specific 3-subgraph pair correspond to common edges among a subset of 4-subgraphs. Before that, in an initial GAS round, each vertex v must gather the ID of each vertex in its neighborhood, *i.e.* $a \in \Gamma(v)$, and the quantities in (2.2) must be stored at each edge va during the Scatter phase. **Gather:** The above quantities are summed at each vertex v as in (2.3) to calculate the local 3-profile at v. In addition, we gather the sum of functions of pairs of these

quantities forming 13 *edge pivot* equations:

$$\begin{aligned}
\sum_{a\in\Gamma(v)} \binom{n^e_{1,va}}{2} &= F_1(v) + F_2(v),\\
\sum_{a\in\Gamma(v)} \binom{n^c_{2,va}}{2} &= 3F_6^{'}(v) + F_8^{'}(v),\\
\sum_{a\in\Gamma(v)} \binom{n_{3,va}}{2} &= F_9^{'}(v) + 3F_{10}(v),\\
\sum_{a\in\Gamma(v)} n^e_{1,va} n^c_{2,va} &= 2F_3^{'}(v) + F_4^{'}(v),\\
\sum_{a\in\Gamma(v)} n^e_{1,va} n_{3,va} &= 2F_5(v) + F_8^{''}(v),\\
\sum_{a\in\Gamma(v)} n^c_{2,va} n^e_{2,va} &= F_4^{'}(v) + 2F_7(v),\\
\sum_{a\in\Gamma(v)} n^c_{2,va} n_{3,va} &= 2F_8^{'}(v) + 2F_9^{'}(v),\\
n^d_{1,v}|\Gamma(v)| &= F_2(v) + F_4(v) + F_8(v).
\end{aligned} \tag{2.6}$$

The primed notation differentiates between subgraphs of different automorphism orbits, as in Figure 2.3(c). By accumulating pairs of 3-profile structures as in (2.6), we receive aggregate connectivity information about vertices more than 1 hop away. Consider the sixth equation as an example. The product between $n_{2,va}$ and $n^e_{2,va}$ subgraphs along edge va forms 4-node graphs with the following structural constraints: three vertex pairs are connected, two vertex pairs are disjoint, and one pair may be either connected or disjoint. $F_4^{'}(v)$ and $F_7(v)$ satisfy these constraints and differ on the unconstrained edge. Thus, as shown in Figure 2.4, they both contribute to the sum of $n^c_{2,va} n^e_{2,va}$.

The following edge pivot equations are linearly independent when solving for the local 4-profile only. Note the last 2 equations require calculating the local

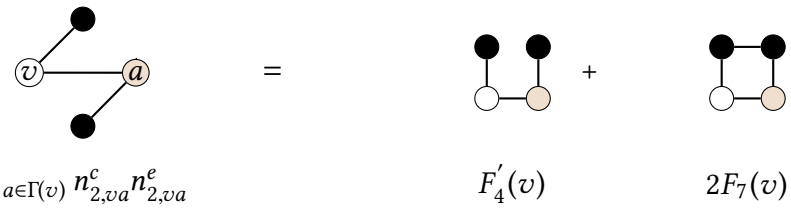


Figure 2.4: Edge pivot equation for vertex v counting triangles as edges va pivot about their common vertex v. The subgraphs $F_4'(v)$ and $F_7(v)$ differ by one edge.

3-profile:

$$\begin{aligned}
\sum_{a\in\Gamma(v)} \binom{n^e_{2,va}}{2} &= F_6(v) + F_8(v),\\
\sum_{a\in\Gamma(v)} n^e_{1,va} n^e_{2,va} &= F_3(v) + F_4(v),\\
\sum_{a\in\Gamma(v)} n^e_{2,va} n_{3,va} &= F_8''(v) + 2F_9(v),\\
\sum_{a\in\Gamma(v)} n_{3,a} - n_{3,va} &= F_8(v) + 2F_9(v) + 3F_{10}(v),\\
\sum_{a\in\Gamma(v)} n^e_{2,a} - n^c_{2,va} &= F_4(v) + 2F_7(v) + F_8''(v) + 2F_9'(v).
\end{aligned} \tag{2.7}$$

Apply: Store the left hand sides of all 13 equations at v.

Clique Counting: The aim of this subtask is to count 4-cliques that contain the vertex v. Similar to Section 2.2.2, we accumulate a list of triangles at each vertex v. Then, at the Scatter stage for every va, it is possible to check if neighbors common to v and a have an edge between them. This implies a 4-clique.

Scatter: In addition to the intersection size $|\Gamma(v) \cap \Gamma(a)|$ at each edge va as before, we now require the intersection list $\{b : b \in \Gamma(v), b \in \Gamma(a)\}$ as a starting point.

Gather, Apply: The intersection list is gathered at each vertex v. This produces all pairs of neighbors in $\Gamma(v)$ which are adjacent, *i.e.* all triangles containing v. It is stored as $\Delta(v)$ during the Apply stage at v.

Gather, Apply: Each edge va computes the number of 4-cliques by counting how many pairs in $\Delta(a)$ contain exactly two neighbors of v. We use a similar equation to calculate $F_8(v)$ concurrently:

$$\begin{aligned} \sum_{a \in \Gamma(v)} |(b,c) \in \Delta(a) : b \in \Gamma(v), c \in \Gamma(v)| &= 3F_{10}(v), \\ \sum_{a \in \Gamma(v)} |(b,c) \in \Delta(a) : b \notin \Gamma(v), c \notin \Gamma(v)| &= F_8(v). \end{aligned} \tag{2.8}$$

At the Apply stage, store the left hand sides as vertex data.

Histogram 2-hop Information: Instead of calculating the number of cycles $F_7(v)$ directly, we can simply construct another linearly independent equation and add it to our system. Let each vertex a have a vector of (vertex ID, count) pairs $(p, c_a[p])$ for each of its adjacent vertices p. Initially, $c_a[p] = 1$ and this *histogram* contains the same information as $\Gamma(a)$. For any $a \in \Gamma(v)$ and $p \notin \Gamma(v)$, $c_a[p] = 1 \Leftrightarrow vap$ forms a 2-path. Thus, v can collect these vectors to determine the total number of 2-paths from v to p. This lets us calculate a linear combination involving cycle subgraph counts with an equation that is linearly independent from the others in our system.

Gather: At each v, take a union of histograms from each neighbor a, resolving duplicate entries with the reduce operation $(p, c_{a_1}) \oplus (p, c_{a_2}) = (p, c_{a_1} + c_{a_2})$.

Algorithm 4 4-PROF

1: **Input:** Graph $G(V,E)$ with $|V|$ vertices, $|E|$ edges.
2: **Gather:** For each vertex v union over edges of the 'other' vertex in the edge, $\cup_{a\in\Gamma(v)} a = \Gamma(v)$.
3: **Apply:** Store the gather as vertex data `v.nb`, size automatically stored.
4: **Scatter:** For each edge e_{va}, compute and store scalars in (2.2).
5: **Gather:** For each edge e_{va}, sum edge scalar data of neighbors in (2.6) - (2.7) and combine two-hop histograms.
6: **Apply:** For each vertex v, sum over $p \notin \Gamma(v)$ in (2.9), store other data in array `v.u`. No Scatter.
7: **Gather:** For each vertex v collect pairs of connected neighbors in $\Delta(v)$.
8: **Apply:** Store connected neighbor (triangle) list as vertex data `v.conn`. No Scatter.
9: **Gather:** For each vertex v sum (2.8).
10: **Apply:** Append data to array `v.u`. Multiply `v.u` by a matrix to solve system of equations.
11: **return** `[v: v.N0 v.N1 v.N2 . . . v.N10]`

Apply: Given the gathered histogram vector, $\{\oplus_{a\in\Gamma(v)} c_a[p]\}_{p\notin\Gamma(v)}$, calculate the number of non-induced 4-cycles involving p and two neighbors:

$$\sum_{p\notin\Gamma(v)} \binom{\oplus_{a\in\Gamma(v)} c_a[p]}{2} = F_7(v) + F_9(v). \tag{2.9}$$

Next, we upper bound the savings from our 2-hop histogram by analyzing the improvement when the only information transmitted across the network to a vertex v is each non-neighboring vertex and its final count $\oplus_{a\in\Gamma(v)} c[p]$. Let

$$h_v \;=\; |\Gamma(\Gamma(v)) \setminus \{\Gamma(v) \cup v\}|.$$

For each v, the difference between full information and histogram information is at most $\sum_{a\in\Gamma(v)}(|\Gamma(a)|-1) - 2h_v$. The exact benefit of (2.9) depends on the internal implementation of the reduce operation $\oplus$ as pairs of neighbors are gathered.

Counting the number of distinct pairs of 2-paths to each 2-hop neighbor, *i.e.* $\frac{1}{2}(c[p]^2 - c[p])$, requires counting the second moment of c taken over h_v terms. Due to a result by Alon ([61], Proposition 3.7), the memory required to count this value exactly (moreover, to approximate it deterministically) is $\Omega(h_v)$. Thus, up to implementation details, our memory use is optimal.

Normalization and Symmetry: Our final local equation comes from summing the local 4-profile across all 17 automorphisms:

$$\sum_{i}^{17} F_i(v) = \binom{|V| - 1}{3}. \tag{2.10}$$

To calculate the global 4-profile, we utilize global symmetry and scaling equations. Let $F_i = \sum_{v \in V} F_i(v)$. Globally, each subgraph count is in exact proportion with the same subgraph counted from a different vertex automorphism. The ratio depends on the subgraph's degree distribution:

$$\begin{aligned} &F_3 = 2F_3^{'}, \quad F_4 = F_4^{'}, \quad F_6 = 3F_6^{'}, \\ &F_8 = F_8^{'}, \quad F_8^{''} = 2F_8, \quad F_9 = F_9^{'}. \end{aligned} \tag{2.11}$$

Global symmetry makes the equation for F_8 and the system (2.7) linearly dependent. We sum across vertices, inverting a single 11×11 system to yield the final global 4-profile $[N_0, \ldots, N_{10}]^\top$ by scaling appropriately:

$$\begin{aligned} &N_0 = \frac{F_0}{4}, \quad N_1 = \frac{F_1}{2}, \quad N_2 = \frac{F_2}{4}, \quad N_3 = F_3^{'}, \\ &N_4 = \frac{F_4^{'}}{2}, \quad N_5 = \frac{F_5}{3}, \quad N_6 = F_6^{'}, \quad N_7 = \frac{F_7}{4}, \\ &N_8 = F_8, \quad N_9 = \frac{F_9}{2}, \quad N_{10} = \frac{F_{10}}{4}. \end{aligned} \tag{2.12}$$

2.3 Sampling, Estimators, and Concentration Bounds

In this section, we describe and analyze an unbiased estimator of the global k-profile based on uniform edge subsampling. Our analysis models the transformation from original to sampled graph as a one step Markov chain with transitions expressed as a function of the sampling probability. Our result is that a random sampling of edges forms a k-profile sparsifier, *i.e.* a subgraph that preserves the elements of the k-profile with sufficient probability concentration. This framework is a generalization of the triangle sparsifiers by Tsourakakis *et al.* [34].

Our first proof relies on a result by Kim and Vu [35] for concentration of multivariate polynomials, similarly to [34]. Unfortunately, the Kim and Vu concentration holds only for a class of polynomials called totally positive and some terms in the 3-profile do not satisfy this condition. For that reason, the proof of [34] does not directly extend beyond triangles. Our technical innovation involves showing that it is still possible to decompose our polynomials as combinations of totally positive polynomials using a sequence of variable changes.

While this technique scales well in theory, typically the estimated errors are orders of magnitude larger than the measured quantities for reasonable graph sizes. Our second proof introduces novel concentration bounds for global k-profile sparsifiers that use a novel information theoretic technique called read-k functions [36]. Our read-k bounds allow usable concentration inequalities for sparsification factors of approximately 0.4 or higher (Section 2.4). Note that removing half the edges of the graph does not accelerate the running time by a factor of 2, but rather by a factor of nearly 8, as shown in our experiments.

2.3.1 k-profile Sparsifier

In this section, we describe the process for approximating the exact number of subgraphs in a graph G. For 3 node subgraphs of type H_0, H_1, H_2 and H_3, as depicted in Figure 2.1(a), denote the exact counts by $\mathbf{n}_3 = [n_0, \ldots, n_3]^\top$ and the estimates by $[X_0, \ldots, X_3]^\top$. Because the vector $\mathbf{n}_3$ is a scaled probability distribution, there are only 3 degrees of freedom. Therefore, we calculate

$$\mathbf{n}_3(G) = \left[\binom{|V|}{3} - n_1 - n_2 - n_3, n_1, n_2, n_3 \right]^\top .$$

We are sparsifying the original graph G by keeping each edge independently with probability p. Denote the random subsampled graph by $\tilde{G}$ and its global 3-profile by $[Y_0, \ldots, Y_3]^\top$. We relate the subsampled 3-profile counts to the original ones through a one step Markov chain involving transition probabilities. The subsampling process is the random step in the chain. Any specific subgraph is preserved with some probability and otherwise transitions to one of the other subgraphs. For example, a 3-clique is preserved with probability p^3. Figure 2.5 illustrates the other transition probabilities.

In expectation, this yields the following linear system:

$$\begin{bmatrix} \mathbb{E}[Y_0] \\ \mathbb{E}[Y_1] \\ \mathbb{E}[Y_2] \\ \mathbb{E}[Y_3] \end{bmatrix} = \begin{bmatrix} 1 & 1-p & (1-p)^2 & (1-p)^3 \\ 0 & p & 2p(1-p) & 3p(1-p)^2 \\ 0 & 0 & p^2 & 3p^2(1-p) \\ 0 & 0 & 0 & p^3 \end{bmatrix} \begin{bmatrix} n_0 \\ n_1 \\ n_2 \\ n_3 \end{bmatrix}, \tag{2.13}$$

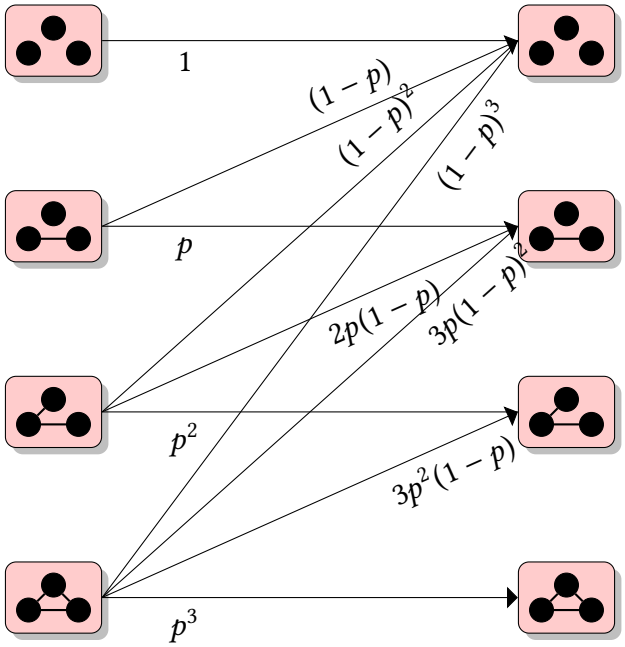


Figure 2.5: Edge sampling process.

from which we obtain unbiased estimators for each entry in $\mathbf{X}(G) = [X_0, \ldots, X_3]^\top$:

$$\begin{aligned} X_0 &= Y_0 - \frac{1-p}{p} Y_1 + \frac{(1-p)^2}{p^2} Y_2 - \frac{(1-p)^3}{p^3} Y_3 \\ X_1 &= \frac{1}{p} Y_1 - \frac{2(1-p)}{p^2} Y_2 + \frac{3(1-p)^2}{p^3} Y_3 \\ X_2 &= \frac{1}{p^2} Y_2 - \frac{3(1-p)}{p^3} Y_3 \\ X_3 &= \frac{1}{p^3} Y_3. \end{aligned} \tag{2.14}$$

Lemma 2.3.1. $\mathbf{X}(G)$ *is an unbiased estimator of* $\mathbf{n}(G)$.

Proof. By substituting (2.14) into (2.13), clearly $\mathbb{E}[X_i] = n_i$ for $i = 0, 1, 2, 3$. □

For the rest of the subsection, we shift our attention to 4-profiles. Denote the exact counts by $\mathbf{n}_4 = [n_0, \ldots, n_{10}]^\top$, the estimates by $[X_0, \ldots, X_{10}]^\top$, and the global 4-profile of $\tilde{G}$ by $[Y_0, \ldots, Y_{10}]^\top$. Just as each triangle survives with probability p^3 and each 4-clique clearly survives with p^6. Therefore, in expectation, $\mathbb{E}[Y_{10}] = p^6 N_{10}$ and $X_{10} = \frac{1}{p^6} Y_{10}$ is unbiased.

This simple correspondence does not hold for other subgraphs: each clique in $\tilde{G}$ can only be a clique in G that survived edge removals, but other subgraphs of $\tilde{G}$ could be originating from multiple subgraphs of G depending on the random sparsification process. We can, however, relate the original 4-profile vector to the expected subsampled 4-profile vector by a matrix multiplication. Let $F(abcd)$ and $\tilde{F}(abcd)$ represent the induced 4-subgraph on the vertices $abcd$ before and after subsampling, respectively. Then define $\mathbf{H}$ by $H_{ij} = \mathbb{P}(\tilde{F}(abcd) = F_i \mid F(abcd) = F_j)$. Thus, we form an unbiased estimator, *i.e.* $\mathbb{E}[X_i] = N_i$, $i = 1, \ldots, 10$, by inverting the edge sampling matrix.

For 3-profiles, this process is described by the system (2.13). For 4-profiles, the vectors are 11 dimensional and a similar linear system can be explicitly computed – the full system of equations is included in Appendix B. This matrix turns out to be invertible and we can therefore calculate the 4-profile exactly if we have access to the expected values of the sparsified 4-profile. Of course, we can only obtain one sample random graph and calculate that 4-profile, which will be an accurate estimate if the 4-profile quantities are sufficiently concentrated around their expectation.

2.3.2 Kim-Vu Concentration Bounds

We now turn to prove our first concentration bounds for the above estimators. We introduce some notation for this purpose. Let X be a real polynomial function of m real random variables $\{t_i\}_{i=1}^m$. Let $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_m) \in \mathbb{Z}_+^m$ and define $\mathbb{E}_{\geq 1}[X] = \max_{\boldsymbol{\alpha}:\|\boldsymbol{\alpha}\|_1 \geq 1} \mathbb{E}(\partial^{\boldsymbol{\alpha}} X)$, where

$$\mathbb{E}(\partial^{\boldsymbol{\alpha}} X) = \mathbb{E}\left[(\frac{\partial}{\partial t_1})^{\alpha_1} \ldots (\frac{\partial}{\partial t_m})^{\alpha_m} [X(t_1, \ldots, t_m)] \right]. \tag{2.15}$$

Further, we call a polynomial totally positive if the coefficients of all the monomials involved are non-negative. We state the main technical tool we use to obtain our concentration results.

Theorem 2.3.1 (Kim-Vu Concentration [35]). *Let X be a random totally positive Boolean polynomial in m Boolean random variables with degree at most k. If $\mathbb{E}[X] \geq \mathbb{E}_{\geq 1}[X]$, then*

$$\mathbb{P}\left(|X - \mathbb{E}[X]| > a_k \sqrt{\mathbb{E}[X]\mathbb{E}_{\geq 1}[X]} \lambda^k \right)$$

$$= O\left(\exp\left(-\lambda + (k-1)\log m\right)\right) \tag{2.16}$$

for any $\lambda > 1$, *where* $a_k = 8^k k!^{1/2}$.

The above theorem was used to analyze 3-profiles of Erdős-Rényi random ensembles ($G_{n,p}$) in [35]. Later, this was used to derive concentration bounds for triangle sparsifiers in [34]. Here, we extend §4.3 of [35] to the 3-profile estimation process, on an arbitrary edge-sampled graph.

Theorem 2.3.2. *(Generalization of triangle sparsifiers to 3-profile sparsifiers) Let* $\mathbf{n}_3(G) = [n_0, n_1, n_2, n_3]$ *be the 3-profile of a graph* $G(V, E)$. *Let* $|V| = n$ *and* $|E| = m$. *Let* $\mathbf{n}_3(\tilde{G}) = [Y_0, Y_1, Y_2, Y_3]$ *be the 3-profile of the subgraph obtained by sampling each edge in* G *with probability* p. *Let* α, β *and* Δ *be the largest collection of* H_1*'s, wedges and triangles that share a common edge. Define* $\mathbf{X}(G)$ *according to* (2.14), $\epsilon > 0$, *and* $\gamma > 0$. *If* p, ϵ *satisfy:*

$$\begin{aligned}
\frac{n_0}{3\max\{\alpha,\beta,\Delta\}} &\geq \frac{a_3^2 \log^6\left(m^{2+\gamma}\right)}{\epsilon^2} \\
\frac{p}{\max\{\frac{1}{\sqrt[3]{n_3}}, \Delta/n_3\}} &\geq \frac{a_3^2 \log^6\left(m^{2+\gamma}\right)}{\epsilon^2} \\
\frac{p}{\alpha/n_1} &\geq \frac{a_1^2 \log^2\left(m^{\gamma}\right)}{\epsilon^2} \\
\frac{p}{\max\{\frac{\beta}{n_2}, \frac{1}{\sqrt{n_2}}\}} &\geq \frac{a_2^2 \log^4\left(m^{1+\gamma}\right)}{\epsilon^2},
\end{aligned} \tag{2.17}$$

then $\|\mathbf{X}(G) - \mathbf{n}(G)\|_\infty \leq 12\epsilon\binom{|V|}{3}$ *with probability at least* $1 - \frac{1}{m^\gamma}$.

Proof. The proof of this theorem is deferred to Appendix A.1. □

The sampling probability p in Theorem 2.3.2 depends poly-logarithmically on the number of edges and linearly on the fraction of each subgraph which occurs on a common edge. For example, if all of the wedges in G depend on a single edge, *i.e.* $\beta = n_2$, then the last equation suggests the presence of that particular edge in the sampled graph will dominate the overall sparsifier quality.

2.3.3 Read-k Concentration Bounds

When applied to graph sparsification for triangle counting, the Kim and Vu polynomial concentration [34, 35] from the previous section unfortunately gives very loose bounds for practical graph sizes. Figure 2.6 compares the accuracy bound derived in this section to the bound predicted by using [35]. Clearly the Kim-Vu concentration does not provide meaningful bounds for the experiments in Section 2.4. However, the bounds in this section match observed sparsifier accuracy much more closely.

The novel concentration results in this section exploit the fact that partial derivatives of the desired quantities are sparse in the number of edge variables. This allows us to use a novel information theoretic concentration technique called read-k functions [36]. For simplicity, we only explain the concentration of 4-cliques (F_{10} subgraphs) here. We establish the general result for all 11 4-profile variables in Appendix B. Our main concentration result is as follows:

Theorem 2.3.3. *Let G be a graph with N_{10} 4-cliques, and let k_{10} be the maximum number of 4-cliques sharing a common edge. Let X_{10} be the 4-clique estimate obtained from subsampling each edge with probability $0 < p \leq 1$, choose $0 < \delta < 1$, and*

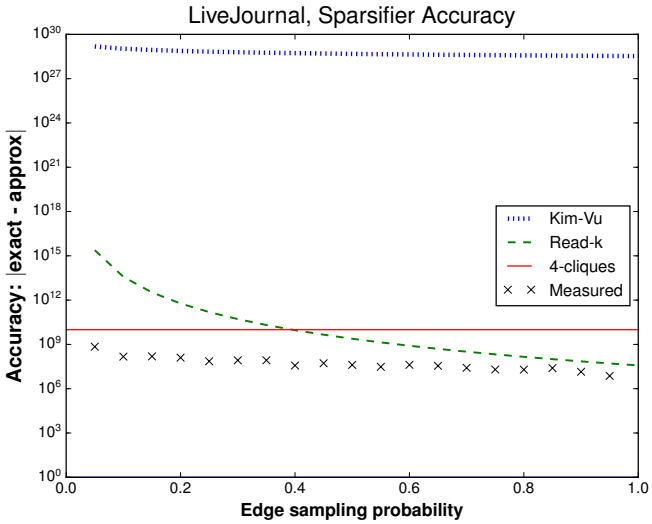


Figure 2.6: Comparison of 4-clique sparsifier concentration bounds with accuracy measured in edge sampling experiments on the LiveJournal graph.

choose $\epsilon_{RK} > 0$. *If*

$$p \geq \left(\frac{\log(2/\delta) k_{10}}{2\epsilon_{RK}^2 N_{10}} \right)^{1/12},$$

then $|N_{10} - X_{10}| \leq \epsilon_{RK} N_{10}$ *with probability at least* $1 - \delta$.

Proof. Our proof relies on read-k function families [36], a recent characterization of dependencies among functions of random variables. Rather than Lipschitz bounding the value of each partial derivative, as in [25, 34, 35], this technical tool benefits from the fact that each first partial derivative is sparse in the number of edge variables.

Definition 2.3.1 (read-k families)**.** *Let* $X_1, \ldots, X_m$ *be independent random variables. For* $j \in [r]$, *let* $P_j \subseteq [m]$ *and let* f_j *be a Boolean function of* $\{X_i\}_{i \in P_j}$. *Assume*

that $|\{j \mid i \in P_j\}| \leq k$ *for every* $i \in [m]$. *Then the random variables* $Z_j = f_j(\{X_i\}_{i \in P_j})$ *are called a read-k family.*

Each variable only affects k of the r Boolean functions. Let G be a graph with N_{10} 4-cliques and a maximum of k_{10} 4-cliques sharing a common edge. The corresponding 4-clique estimator X_{10} follows this exact structure. Each edge sampling variable appears in at most k_{10} of the N_{10} terms. We now state the main result required for our analysis. Note that when applied to estimating the number of 4-cliques, the bound is independent of the number of edges. Therefore, it is much stronger than arguments involving Lipschitz bounded functions such as McDiarmid's inequality.

Proposition 1 (Read-k Concentration [36])**.** *Let* $Z_1, \ldots, Z_r$ *be a family of read-k indicator variables with* $\mathbb{P}(Z_i = 1) = p_i$, *and let* p *be the average of* $p_1, \ldots, p_r$. *Then for any* $\epsilon > 0$,

$$\mathbb{P}\left(\sum_{i=1}^{r} Z_i \geq (p+\epsilon)r\right) \leq \exp\left(-D(p+\epsilon \parallel p)\frac{r}{k}\right)$$
$$\mathbb{P}\left(\sum_{i=1}^{r} Z_i \leq (p-\epsilon)r\right) \leq \exp\left(-D(p-\epsilon \parallel p)\frac{r}{k}\right),$$

where $D(x \parallel y) = x \log\left(\frac{x}{y}\right) + (1-x)\log\left(\frac{1-x}{1-y}\right)$ *is the Kullback-Leibler divergence of* x *and* y. *Both bounds are less than* $\exp(-2\epsilon^2 r/k)$.

Let $Y_{10} = \sum_{\boxtimes(a,b,c,d) \in \mathcal{H}_{10}} t_{ab} t_{bc} t_{cd} t_{da} t_{ac} t_{bd}$. Then

$$\mathbb{P}\left(|Y_{10} - p^6 N_{10}| \geq \epsilon_{RK} N_{10}\right) \leq 2\exp\left(-\frac{2\epsilon_{RK}^2 N_{10}}{k_{10}}\right)$$

$$\Rightarrow \mathbb{P}\left(|X_{10} - N_{10}| \geq \epsilon_{RK} N_{10}\right) \leq 2\exp\left(-\frac{2p^{12}\epsilon_{RK}^2 N_{10}}{k_{10}}\right).$$

The claim follows by setting the right hand side less than δ and solving for p. □

Next, we state conditions under which our method outperforms the Kim and Vu concentration results [35].

Corollary 2.3.1. *Let G be a graph with m edges. If $p = \Omega(1/\log m)$ and $\delta = \Omega(1/m)$, then read-k provides better triangle sparsifier accuracy than Kim-Vu. If additionally $k_{10} \leq N_{10}^{5/6}$, then read-k provides better 4-clique sparsifier accuracy than Kim-Vu.*

Proof. We prove this result for the case of 4-cliques only because the case for triangles is similar. First we must derive a similar 4-clique concentration bound using the techniques in [34, 35].

Lemma 2.3.2. *Let G be a graph with m edges and N_{10} cliques, and k_{10} be the maximum number of 4-cliques sharing a common edge. Let $a_6 = 8^6\sqrt{6!}$, $0 < p \leq 1$, and $\epsilon_{KV} > 0$. Let X_{10} be the 4-clique estimate obtained from subsampling each edge with probability p. If*

$$\frac{p}{\max\left\{\sqrt[6]{1/N_{10}}, \sqrt[3]{k_{10}/N_{10}}\right\}} \geq \frac{a_6^2 \log^{12}(m^{5+\gamma})}{\epsilon_{KV}^2}, \tag{2.18}$$

then $|N_{10} - X_{10}| \leq \epsilon_{KV} N_{10}$ with probability at least $1 - \frac{1}{m^\gamma}$.

Similar to Theorem 2.3.2, the proof of this lemma is an application of the main result in [35]. It can be found in Appendix A.

Now we are ready to prove the corollary by comparing Theorem 2.3.3 and Lemma 2.3.2. Fix $p, \delta, > 0$ and $\gamma > 1$ such that $p = \Omega(1/\log m)$ and $\delta = m^{-\gamma} = \Omega(1/m)$. Now we analyze the bounds ϵ_{KV} and ϵ_{RK}. For any graph and a_6 defined in Lemma 2.3.2,

$$\begin{aligned} \frac{1}{a_6^2} &\leq 1, & \frac{\gamma}{(5+\gamma)^{12}} &\leq 1, \\ \log(2^{1/\gamma} m) &\leq 2\log m, & \left(\frac{k_{10}}{N_{10}}\right)^{2/3} &\leq 1. \end{aligned} \tag{2.19}$$

We further require $k_{10} \leq N_{10}^{5/6}$. Then the condition on p with (2.19) implies

$$\begin{aligned} p^{11} &\geq 1/\log^{11}(m) \\ &\geq \frac{\gamma \log(2^{1/\gamma} m)}{2a_6^2(5+\gamma)^{12}\log^{12}(m)} \min\left\{k_{10}/N_{10}^{5/6}, (k_{10}/N_{10})^{2/3}\right\}. \end{aligned}$$

Rearranging terms,

$$\begin{aligned} \epsilon_{KV}^2 &= \frac{a_6^2 \log^{12}(m^{5+\gamma}) \max\left\{\sqrt[6]{1/N_{10}}, \sqrt[3]{k_{10}/N_{10}}\right\}}{p} \\ \geq \epsilon_{RK}^2 &= \frac{\log(2m^{\gamma}) k_{10}/N_{10}}{2p^{12}}. \end{aligned}$$

□

We note that the asymptotic condition on p in Corollary 2.3.1 includes a constant term much less than 1. This is due to the looseness of inequalities in (2.19) and implies that Theorem 2.3.3 is superior to Lemma 2.3.2 over all p values of practical interest. While these bounds contain the quantities we wish to estimate, they provide guidelines for the performance of sampling heuristics. We also investigate this in the next section for some realistic graphs.

2.4 Experiments

First we describe implementation details common to all experiments in this section. Then we present both running time and accuracy results for 3-profiles, followed by similar results for 4-profiles.

The Systems: We perform the experiments on three systems. The first system is a single power server, further referred to as Asterix. The server is equipped with 256 GB of RAM and two Intel Xeon E5-2699 v3 CPUs, 18 cores each. Since each core has two hardware threads, up to 72 logical cores are available to the GraphLab engine. We implement our algorithms on GraphLab v2.2 (PowerGraph) [11].

The next two systems are EC2 clusters on AWS (Amazon Web Services) [62]. One is comprised of 12 `m3.2xlarge` machines, each having 30 GB RAM and 8 virtual CPUs. Another system is a cluster of 20 `c3.8xlarge` machines, each having 60 GB RAM and 32 virtual CPUs.

The Data: In our experiments we use a total of six real graphs. These graphs represent different datasets: social networks (LiveJournal and Twitter), citations (DBLP), knowledge content (Wikipedia), and WWW structure (PLD – pay level domains and Notre Dame). Notice that the above graphs are originally directed, but since our work deals with undirected graphs, all duplicate edges (*i.e.*, bi-directional) were removed and directionality is ignored. Graph sizes are summarized in Table 2.1.

Table 2.1: Datasets

Name	Vertices	Edges (undirected)
Twitter [63]	41,652,230	1,202,513,046
PLD [64]	39,497,204	582,567,291
LiveJournal [65]	4,846,609	42,851,237
Wikipedia [66]	3,515,067	42,375,912
WEB-NOTRE [65]	325,729	1,090,108
DBLP [65]	317,080	1,049,866

2.4.1 3-profiles

In this section, we describe the experimental setup and results comparing the 3-PROF, EGO-PAR and EGO-SER algorithms. The performance (running time and network usage) of our 3-PROF algorithm is compared with the Undirected Triangles Count Per Vertex (hereinafter referred to as TRIAN) algorithm shipped with GraphLab. We show that in time and network usage comparable to the built-in TRIAN algorithm, our 3-PROF can calculate all the local and global 3-profiles. Then, we compare our parallel implementation of the ego 3-profile algorithm, EGO-PAR, with the naïve serial implementation, EGO-SER. It appears that our parallel approach is much more efficient and scales much better than the serial algorithm. The sampling approach, introduced for the 3-PROF algorithm, yields promising results – reduced running time and network usage while still providing excellent accuracy. We support our findings with several experiments over various datasets and systems. Experimental results are averaged over 3–10 runs.

Local 3-profile vs. Triangle Count: The first result is that our 3-PROF is able to compute all the local 3-profiles in almost the same time as the GraphLab's

built-in TRIAN computes the local triangles (*i.e.*, number of triangles including each vertex). Let us start with the first AWS cluster with less powerful machines (`m3.2xlarge`). In Figure 2.7(a) we can see that for the LiveJournal graph, for each sampling probability p and for each number of nodes (*i.e.*, machines in the cluster), 3-PROF achieves running times comparable to TRIAN. Notice also the benefit in running time achieved by sampling. We can reduce running time almost by half, without significantly sacrificing accuracy (which will be discussed in the sequel). While the running time is decreased as the number of nodes grows (more computing resources become available), the network usage becomes higher (see Figure 2.7(c)) due to the extensive inter-machine communication in GraphLab. We can also see that sampling can significantly reduce network usage. In Figures 2.7(b) and 2.7(d), we can see similar behavior for the Wikipedia graph: running time and network usage of 3-PROF is comparable to TRIAN.

Next, we conduct the experiments on the second AWS cluster with more powerful (`c3.8xlarge`) machines. For LiveJournal, we note modest improvements in running time for nearly the same network bandwidth observed in Figure 2.7. On this system we were able to run 3-PROF and TRIAN on the much larger PLD graph. In Figures 2.8(b) and 2.8(d) we compare the running time and network usage of both algorithms. For the large PLD graph, the benefit of sampling can be seen clearly; by setting $p = 0.1$, the running time of 3-PROF is reduced by a factor of 4 and the network usage is reduced by a factor of 2. Figure 2.9 shows the performance of 3-PROF and TRIAN on the LiveJournal and Wikipedia graphs. We can see that the behavior of running times and the network usage of the 3-PROF al-

gorithm is consistently comparable to TRIAN across the various graphs, sampling, and system parameters.

Let us now show results of the experiments performed on a single powerful machine (Asterix). Figure 2.11(a) shows the running times for 3-PROF and TRIAN for Twitter and PLD graphs. We can see that on the largest graph in our dataset (Twitter), the running time of 3-PROF is less than 5% larger than that of TRIAN, and for the PLD graph the difference is less than 3% (for $p = 1$). Twitter takes roughly twice as long to compute as PLD, implying that these algorithms have running time proportional to the graph's number of edges.

Ego 3-profiles: The next set of experiments evaluates the performance of our EGO-PAR algorithm for counting ego 3-profiles. We show the performance of EGO-PAR for various graphs and systems and also compare it to a naïve serial algorithm EGO-SER. Let us start with the AWS system with `c3.8xlarge` machines. In Figure 2.10 we see the running time of EGO-SER and EGO-PAR on the LiveJournal graph. The task was to find ego 3-profiles of 100, 1K, and 10K randomly selected nodes. Since the running time depends on the size and structure of each induced subgraph, EGO-SER and EGO-PAR operated on the same list of ego vertices. While for 100 random vertices EGO-SER performed well (and even achieved the same running time as EGO-PAR for the PLD graph), its performance drastically degraded for a larger number of vertices. This is due to its iterative nature – it finds ego 3-profiles of the vertices one at a time and is not scalable. Note that the open bars mean that this experiment was not finished. The numbers above them are extrap-

olations, which are reasonable due to the serial design of the Ego-ser.

On the contrary, the Ego-par algorithm scales extremely well and computes ego 3-profiles for 100, 1K, and 10K vertices almost in the same time. In Figure 2.12(a), we can see that as the number of nodes (*i.e.*, machines) increases, the running time of Ego-par decreases since its parallel design allows it to use additional computational resources. However, Ego-ser cannot benefit from more resources and its running time even increases when more machines are used. The increase in running time of Ego-ser is due to the increase in network usage when using more machines (see Figure 2.12(b)). The network usage of Ego-par also increases, but this algorithm compensates by leveraging additional computational power. In Figure 2.13, we can see that Ego-par performs well even when finding ego 3-profiles for all the LiveJournal vertices (4.8M vertices).

Next in Figures 2.11(b) and 2.11(c), we can see the comparison of Ego-par and Ego-ser on the PLD and the DBLP graphs on the Asterix machine. For both graphs, we see a very good scaling of Ego-par, while the running time of Ego-ser scales linearly with the size of the ego vertices list.

Accuracy: Finally, we show that while the sampling approach can significantly reduce running time and network usage, it has negligible effect on the accuracy of the solution. Notice that the sampling accuracy refers to the global 3-profile count (*i.e.*, the sum of all the local 3-profiles over all vertices in a graph). In Figure 2.14 we show accuracy of each scalar in the 3-profile. For the accuracy metrics, we use ratio between the exact count (obtained running 3-prof with $p = 1$) divided by

the estimated count (*i.e.*, the output of our 3-PROF when $p < 1$). It can be seen that for the three graphs, all the 3-profiles are very close to 1. For example, for the PLD graph, even when $p = 0.01$, the accuracy is within 0.004 from the ideal value of 1. Error bars mark one standard deviation from the mean, and across all graphs the largest standard deviation is 0.031. As p decreases, the triangle estimator suffers the greatest loss in both accuracy and consistency.

2.4.2 4-profiles

Let us now describe the implementation and experimental results of our 4-profile algorithm on GraphLab v2.2 (PowerGraph) [11] by measuring its running time and accuracy on large input graphs.[4] First, we show that edge sampling yields very good approximation results for global 4-profile counts and achieves substantial execution speedups and network traffic savings when multiple machines are in use. Due to its distributed nature, we can show 4-PROF runs substantially faster when using multiple CPU cores and/or machines. Notice that multicore and multiple machines can not speed up some centralized algorithms, *e.g.* ORCA [30], which we use as a baseline for our results. Note also that ORCA produces only a partial 4-subgraph count, *i.e.* it calculates only connected 4-subgraphs, while 4-PROF calculates all 17 per vertex.

2-hop Histogram: The first result compares two methods of calculating the left hand side of (2.9) from Section 2.2.3. We show that a simple implementation

[4] Code is available at http://github.com/eelenberg/4-profiles.

in which a vertex gathers its full 2-hop neighborhood (*i.e.*, IDs of its neighbors' neighbors) is much less efficient than the *two-hop histogram* approach used in 4-PROF (see Section 2.2.3). In Figures 2.15 and 2.16 we can see that the histogram approach is an order of magnitude faster for various numbers of machines, and that its network requirements are up to 5x less than that of the simple implementation. Moreover, our algorithm could handle much larger graphs while the simple implementation ran out of memory.

Running Time: Next, we show that 4-PROF can run much faster than the current state-of-the-art graphlet counting implementations. The algorithm and the GraphLab platform on which it runs are both distributed in nature. The latter allows 4-PROF to exploit multiple cores on a single machine as well as a cluster of machines. Figure 2.19(c) shows running time as a function of CPU cores. We compare this result to the running time of a single core, C++ implementation of ORCA [30]. Our 4-PROF algorithm becomes faster after only 25 cores and is 2x faster using 60 cores. Moreover, 4-PROF allows scaling to a large number of machines. In Figure 2.18 we can see how the running time for the LiveJournal graph decreases when the number of machines increases. Since ORCA cannot benefit from multiple machines, we see that 4-PROF runs up to 12x faster than ORCA. This gap widens as the cluster grows larger. In [60], the authors implemented a GPU version of ORCA using CUDA. However, the reported speedup is about 2x which is much less than we show here on the AWS cluster (see Figure 2.18 for $p = 1$). We also note a substantial running time benefit of the sampling approach for *global* 4-profiles.

In Figures 2.18 and 2.17, we see that with $p = 0.1$ we can achieve order of magnitude improvements in both speed and network traffic. This sampling probability maintains very good accuracy, as shown in Figure 2.19(b).

Accuracy: Finally, we show that our edge sampling approach greatly improves running time while maintaining a very good approximation of the global 4-profile. In Figure 2.19(a) we can see that the running time decreases drastically when the sampling probability decreases. At the same time, Figure 2.19(b) shows that the mean ratio of true to estimated global 4-profiles is within $\pm 2.5\%$. Similar to [42], which uses a more complex sampling scheme to count connected 4-subgraphs, this ratio is usually much less than 1%. We show here only profiles $F_7 - F_{10}$ since their counts are the smallest and were observed to have the lowest accuracy. In Figure 2.6 we compare theoretical concentration bounds on a logarithmic scale and show the benefit of Theorem 2.3.3. While the guarantees provided by Kim-Vu [35] bounds are very loose (the additive error is bounded by numbers which are orders of magnitude larger than the true value), the read-k approach is much closer to the measured values. We can see that for large sampling probabilities ($p \geq 0.5$), the measured error is at most 2 orders of magnitude smaller than the value predicted by Theorem 2.3.3.

2.5 Conclusions

We have introduced novel distributed algorithms for estimating 3 and 4-profiles of large graphs. Our concentration theorems and experimental results

confirm that 3-profile estimation via subsampling is comparable in runtime and accuracy to triangle counting. We additionally show that 4-profiles can be estimated with limited 2-hop information and that randomly erasing edges gives sharper approximation guarantees compared to previous analysis. Our scheme outperforms the previous state-of-the-art and can exploit cloud infrastructure to scale.

This chapter offers several directions for future work. First, both local and ego profiles can be used as features to classify vertices in social or bioinformatic networks. While algorithms for global 5-profile counting were recently studied in [31], tractable distributed algorithms for $k > 4$ using similar edge pivot equations remain as future work. Our observed dependence on 4-clique count suggests that an improved graph engine-based clique counting subroutine will improve the parallel algorithm's performance. Subgraph counting could potentially be used to help design graph convolutional network architectures [9, 67, 68]. This perspective may be useful in the growing body of research on vector embeddings [69] and generative models [70, 71] for graphs.

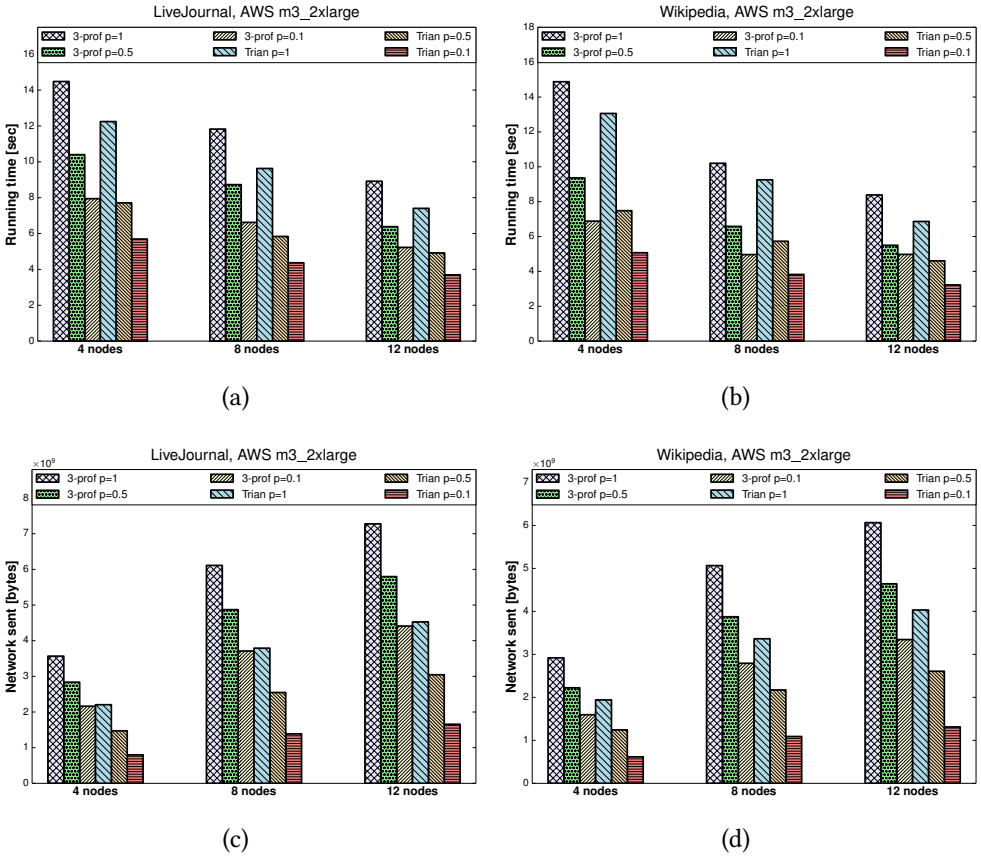


Figure 2.7: AWS `m3.2xlarge` cluster. 3-PROF vs. TRIAN algorithms for LiveJournal and Wikipedia datasets (average of 3 runs). 3-PROF achieves comparable performance to triangle counting. (a,b) – Running time for various numbers of nodes (machines) and various sampling probabilities p. (c,d) – Network bytes sent by the algorithms for various numbers of nodes and various sampling probabilities p.

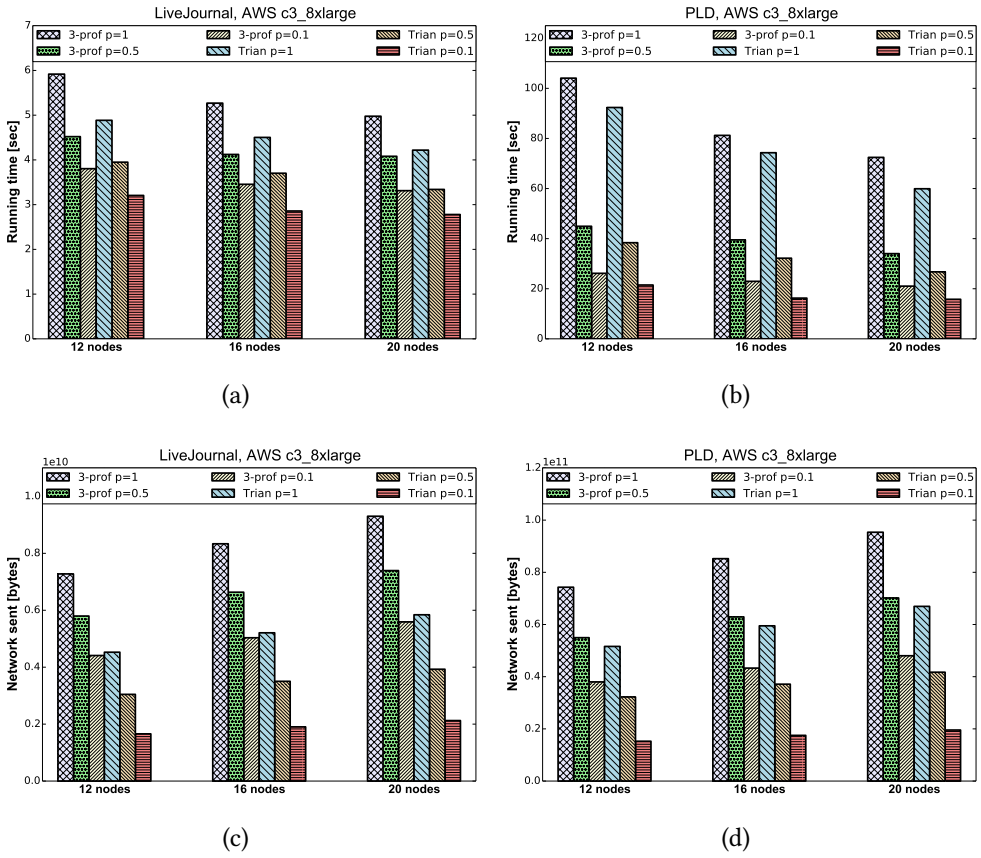
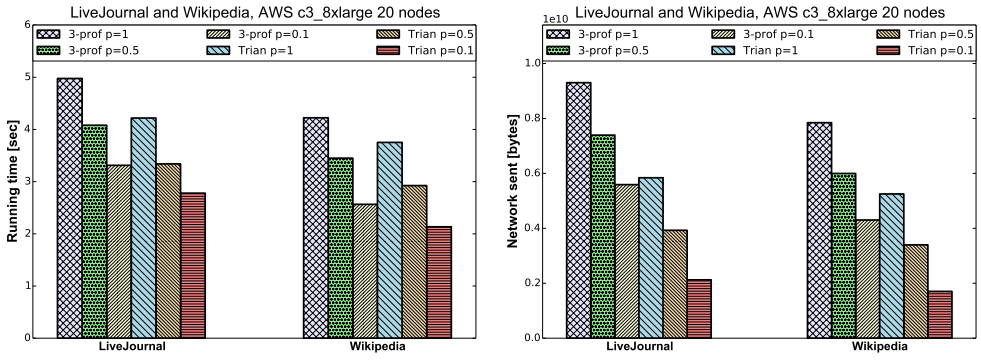
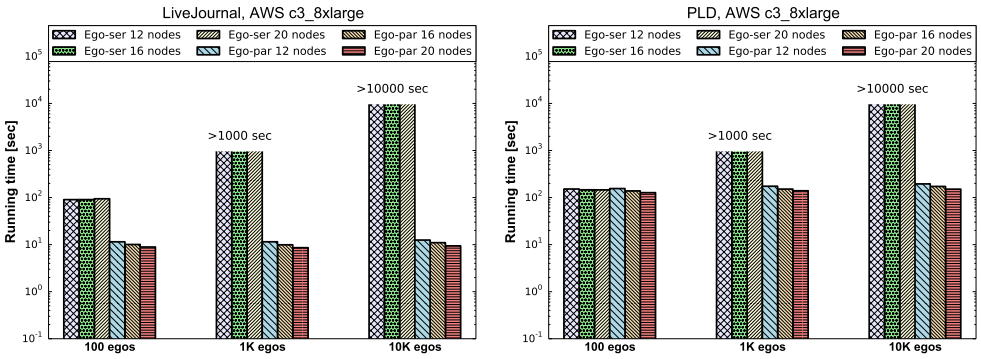


Figure 2.8: AWS `c3.8xlarge` cluster. 3-PROF vs. TRIAN algorithms for LiveJournal and PLD datasets (average of 3 runs). 3-PROF achieves comparable performance to triangle counting. (a,b) – Running time for various numbers of nodes (machines) and various sampling probabilities p. (c,d) – Network bytes sent by the algorithms for various numbers of nodes and various sampling probabilities p.



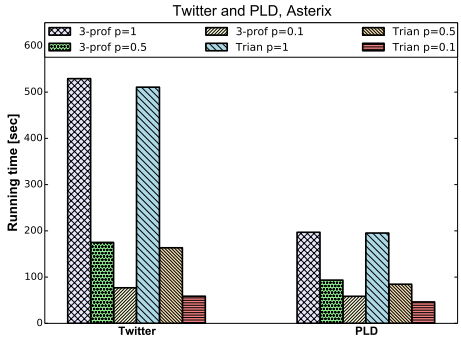
(a) (b)

Figure 2.9: AWS `c3.8xlarge` cluster with 20 nodes. 3-PROF vs. TRIAN results for LiveJournal and Wikipedia datasets (average of 3 runs). (a) – Running time for both graphs for various sampling probabilities p. (b) – Network bytes sent by the algorithms for both graphs for various sampling probabilities p.

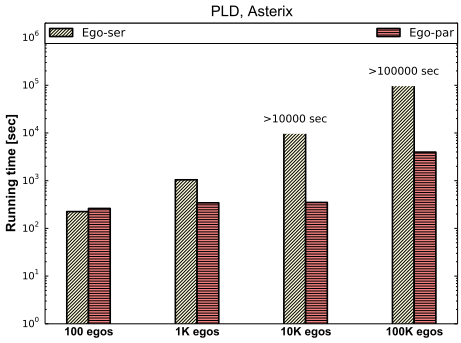


(a) (b)

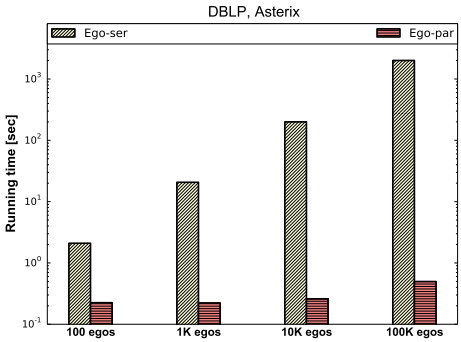
Figure 2.10: AWS `c3.8xlarge` cluster. EGO-PAR vs. EGO-SER results for LiveJournal and PLD datasets (average of 5 runs). Running time of EGO-PAR scales well with the number of ego centers, while EGO-SER scales linearly.



(a)

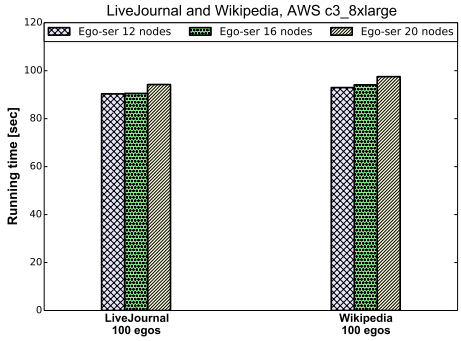


(b)

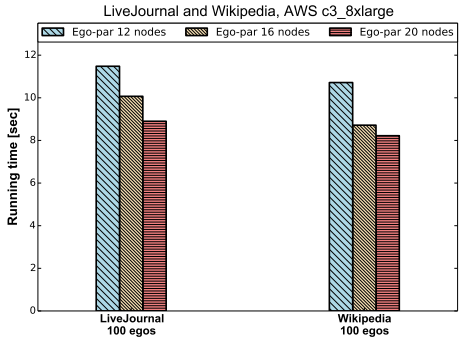


(c)

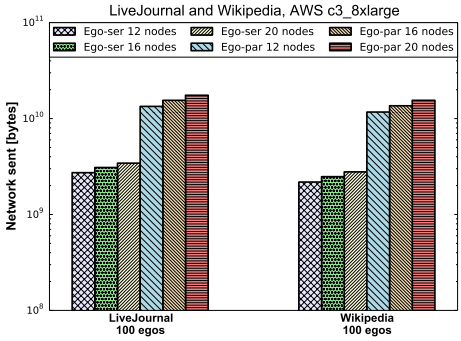
Figure 2.11: Asterix machine. Results for Twitter, PLD, and DBLP datasets. (a) – Running time of 3-PROF vs. TRIAN for various sampling probabilities p. (b,c) – Running time of EGO-PAR vs. EGO-SER for various number of ego centers. Results are averaged over 3, and 3, and 10 runs, respectively.



(a)

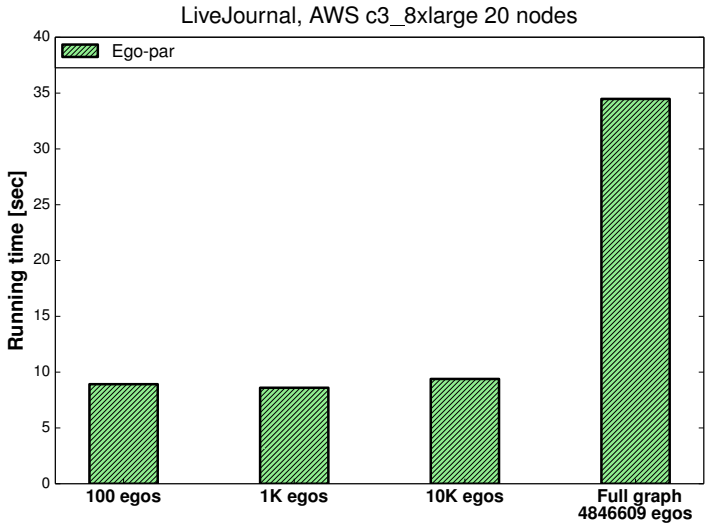


(b)

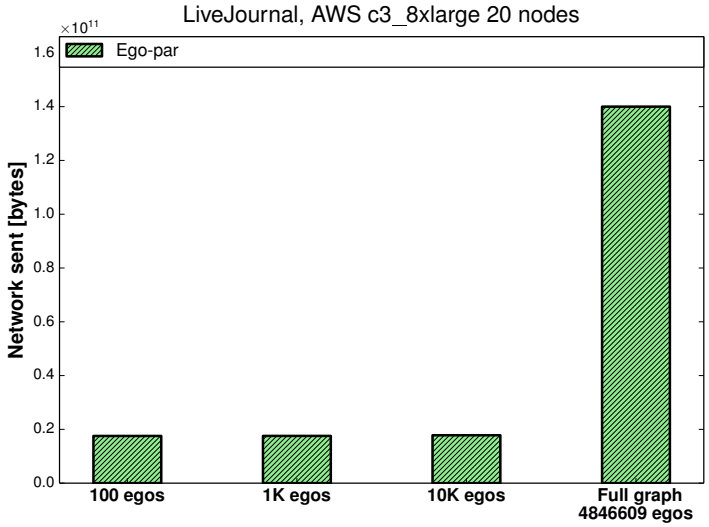


(c)

Figure 2.12: AWS `c3.8xlarge` cluster. Ego-par vs. Ego-ser results for LiveJournal and Wikipedia datasets (average of 5 runs). Running time of Ego-par decreases with the number of machines due to its parallel design. Running time of Ego-ser does not decrease with the number of machines due to its iterative nature. Network usage increases for both algorithms with the number of machines.

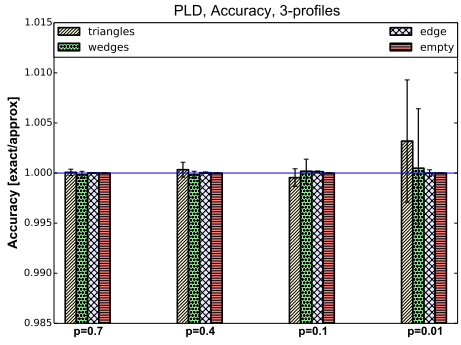


(a)

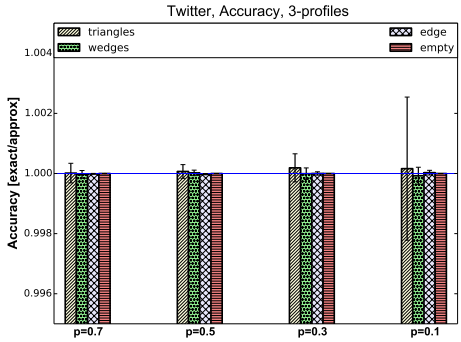


(b)

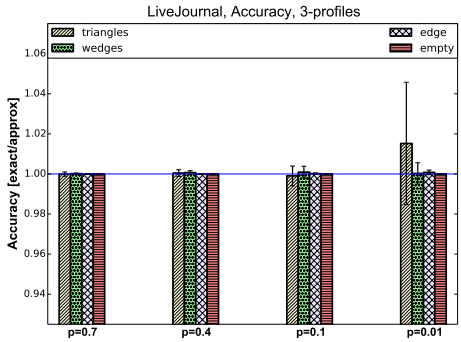
Figure 2.13: AWS `c3.8xlarge` cluster with 20 nodes. EGO-PAR results for LiveJournal dataset (average of 5 runs). The algorithm scales well for various number of ego centers and even full ego centers list. (a) – Running time. (b) – Network bytes sent by the algorithm.



(a)



(b)



(c)

Figure 2.14: Global 3-profiles accuracy achieved by 3-PROF algorithm for various graphs and for each profile count. Results are averaged over 5, and 5, and 10 iterations, respectively. Error bars indicate 1 standard deviation. The metric is a ratio between the exact profile count (when $p = 1$) and the given output for $p < 1$. All results are very close to the optimum value of 1.

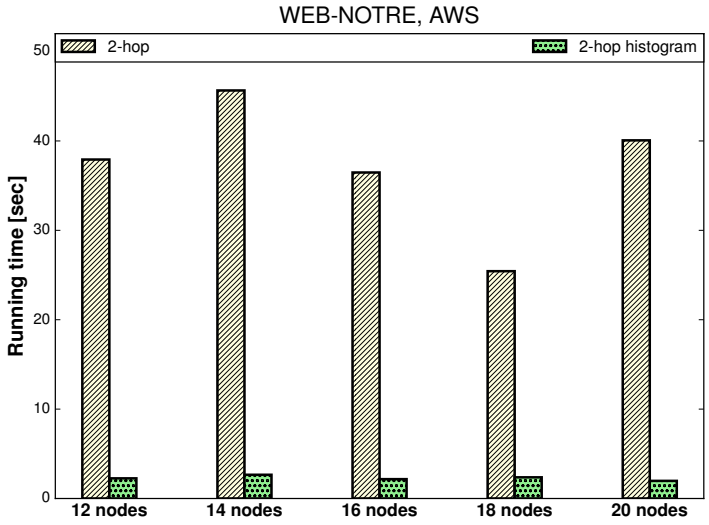


Figure 2.15: AWS cluster of up to 20 machines (nodes), results averaged over 10 iterations. Running time comparing naïve 2-hop implementation and 2-hop histogram approach on the Notre Dame web graph.

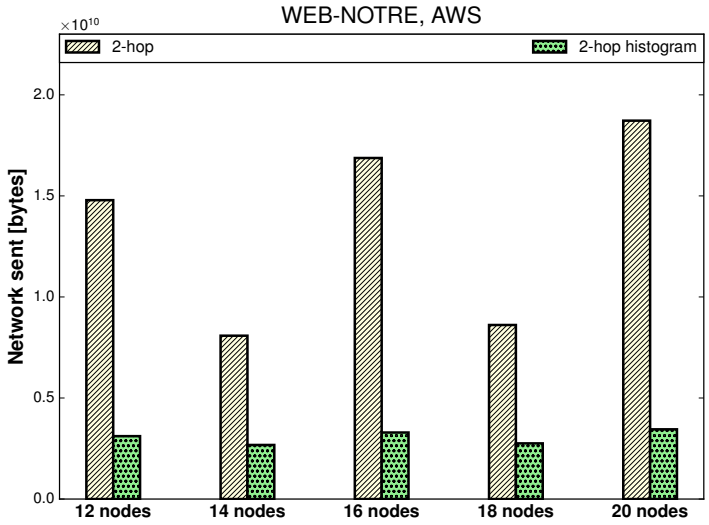


Figure 2.16: Network usage comparing naïve 2-hop implementation and 2-hop histogram approach on the Notre Dame web graph.

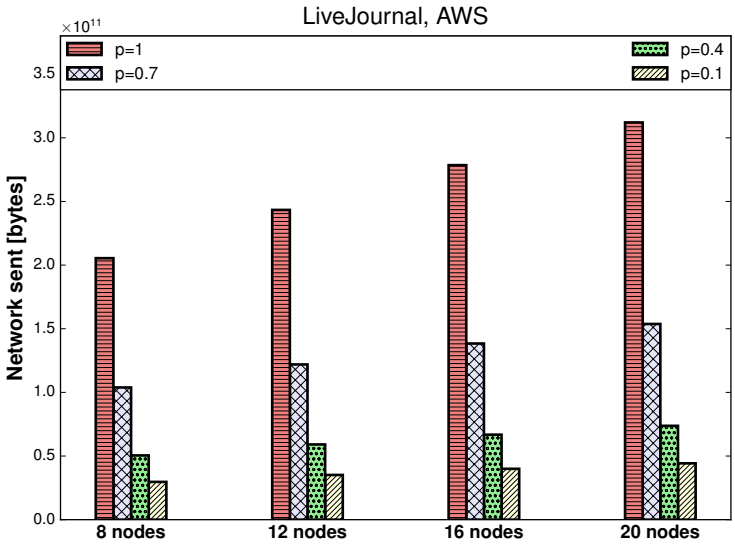


Figure 2.17: Network usage of 4-PROF for various number of compute nodes and sampling probability p, on the LiveJournal graph.

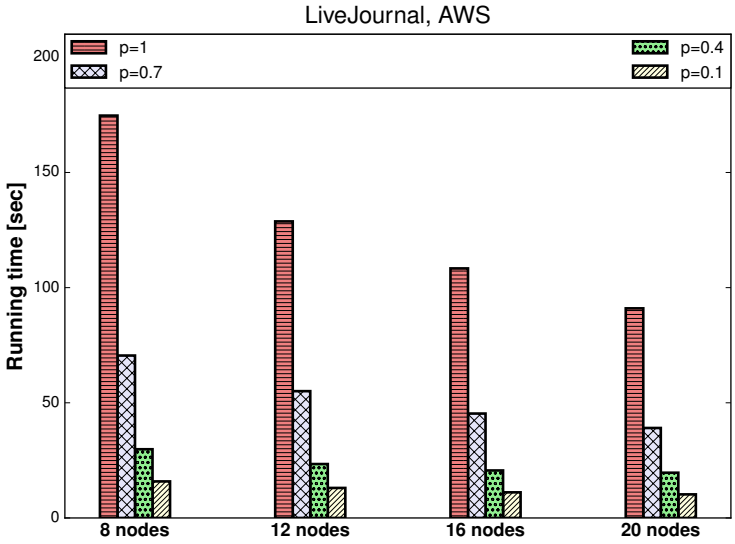
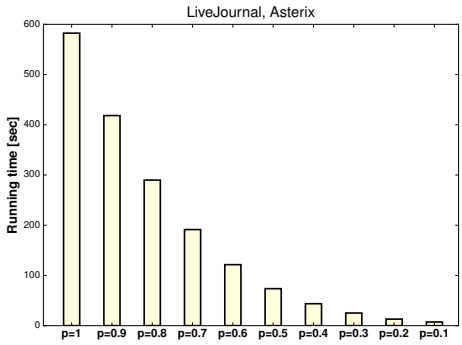
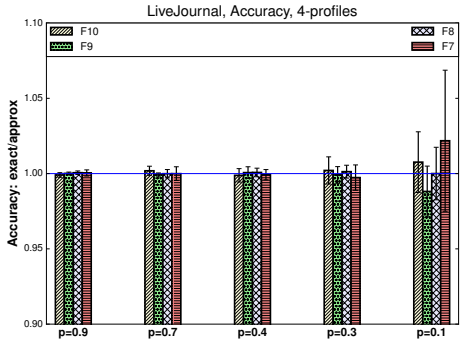


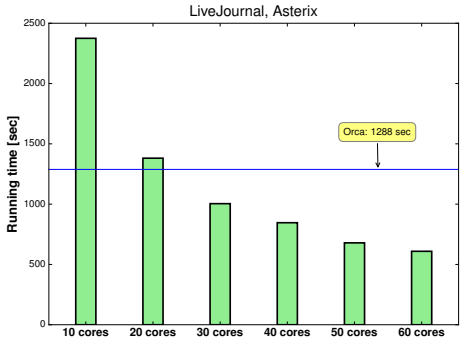
Figure 2.18: Running time of 4-PROF for various number of compute nodes and sampling probability p, on the LiveJournal graph.



(a)



(b)



(c)

Figure 2.19: LiveJournal graph, Asterix system. All the results are averaged over 10 iterations. (a) – Running time as a function of sampling probability. (b) – Accuracy of the $F_7 - F_{10}$ global counts, measured as ratio of the exact count to the estimated count. (c) – Comparison of running times of ORCA and our exact 4-PROF algorithm. Clearly, 4-PROF benefits from the use of multiple cores.

Chapter 3

Weak Submodularity: Restricted Strong Convexity, Subset Selection, and Sparse Regression

3.1 Introduction

Sparse modeling[1] is central in modern data analysis and high-dimensional statistics since it provides interpretability and robustness. Given a large set of p features we wish to build a model using only a small subset of k features: the central combinatorial question is how to choose the optimal feature subset. Specifically, we are interested in optimizing over sparse parameter vectors $\boldsymbol{\beta}$ and consider problems of the form:

$$\bar{\boldsymbol{\beta}}^k \in \underset{\boldsymbol{\beta}:\|\boldsymbol{\beta}\|_0 \leq k}{\operatorname{argmax}} l(\boldsymbol{\beta}) \ , \tag{3.1}$$

for some function $l(\cdot)$. This is a very general framework: the function $l(\cdot)$ can be a linear regression R^2 objective, a generalized linear model (GLM) likelihood, a

[1] The material in this chapter is based on the following workshop paper and journal publication: [72] E. R. Elenberg, R. Khanna, A. G. Dimakis, and S. Negahban. Restricted Strong Convexity Implies Weak Submodularity. In *NIPS Workshop on Learning in High Dimensions with Structure*, 2016. [73] E. R. Elenberg, R. Khanna, A. G. Dimakis, and S. Negahban. Restricted Strong Convexity Implies Weak Submodularity. *The Annals of Statistics*, 2018 (to appear). The dissertation author's primary contributions are the connection of weak submodularity to general sparse regression, submodularity ratio lower bound, generalized approximation guarantees for Oblivious and Forward Stepwise algorithms, sufficient conditions for statistical recovery, and design, implementation, and analysis of experiments. The dissertation author also assisted with other contributions and is the primary contributor of these papers.

graphical model learning objective, or an arbitrary M-estimator [74]. This *subset selection* problem is NP-hard [75] even for the sparse linear regression objective, and a vast body of literature has analyzed different approximation algorithms under various assumptions.

The Restricted Isometry Property (RIP) and the (closely related) Restricted Eigenvalue property are conditions on $l(\boldsymbol{\beta})$ that allow convex relaxations and greedy algorithms to solve the subset selection problem within provable approximation guarantees. In parallel work, several authors have demonstrated that the subset selection problem can be connected to submodular optimization [76–79] and that greedy algorithms are widely used for iteratively building good feature sets.

The mathematical connection between submodularity and RIP was made explicit by Das and Kempe [16] for linear regression. Specifically, they showed that when $l(\boldsymbol{\beta})$ is the R^2 objective, it satisfies a weak form of submodularity when the linear measurements satisfy RIP. Note that for a given set of features S, the function $l(\boldsymbol{\beta}_S)$ with support restricted to S can be thought of as a set function and this is key in this framework. Using this novel concept of *weak submodularity* they established strong multiplicative bounds on the performance of greedy algorithms for subset selection and dictionary selection. Work by Bach [79] in the linear regression setting discusses the notion of *suppressors*; however, that condition is stronger than the weak submodularity assumption. Krause and Cevher [80] draw similar connections between submodularity and sparse regression, but they require a much stronger coherence-based assumption.

In this chapter we extend this machinery beyond linear regression, to any function $l(\boldsymbol{\beta})$. To achieve this we need the proper generalization of the Restricted Eigenvalue and RIP conditions for arbitrary functions. This was obtained by Negahban *et al.* [74] and is called *Restricted Strong Convexity*. Specifically, we show that any objective function that satisfies *restricted strong convexity* (and a natural smoothness assumption) of [74] must be weakly submodular.

We establish multiplicative approximation bounds on the performance of greedy algorithms, including (generalized) Orthogonal Matching Pursuit and Forward Stepwise Regression, for general likelihood functions using our connection. To the best of our knowledge, this is the first analysis of connecting a form of submodularity to the objective function's strong concavity and smoothness. Our approach provides sharp approximation bounds in any setting where these fundamental structural properties are well-understood, *e.g.* generalized linear models.

Contrary to prior work we require no assumptions on the sparsity of the underlying problem. Rather, we obtain a deterministic result establishing multiplicative approximation guarantees from the best-case sparse solution. Our results improve over previous work by providing bounds on a solution that is guaranteed to match the desired sparsity. Convex methods such as ℓ_1 regularized objectives require strong assumptions on the model, such as the irrepresentability conditions on the feature vectors, in order to provide exact sparsity guarantees on the recovered solution.

Our main result is that for any function $l(\cdot)$ that satisfies M-restricted smoothness (RSM) and m-restricted strong convexity (RSC), the corresponding

set function $f(S) = -l(\boldsymbol{\beta}_S)$ is weakly submodular with parameter $\gamma \geq m/M$. The parameters M, m, and γ are defined formally in Section 3.3. We use this result to analyze three greedy algorithms, each progressively better but more computationally intensive: the Oblivious (or Marginal Regression) algorithm computes for each feature the increase in objective and keeps the k individually best features. Orthogonal Matching Pursuit (OMP) greedily adds one feature at a time by picking the feature with the largest inner product with the function gradient. The gradient is the correct generalization of the residual error used in linear regression OMP. Finally, the most sophisticated algorithm is Forward Stepwise Regression: it adds one feature at a time by re-fitting the model repeatedly and keeping the feature that best improves the objective function at each step. We obtain the following performance bounds:

- The Oblivious algorithm produces a (γ/k)-approximation to the best k-subset after k steps.
- Orthogonal Matching Pursuit produces a $(1 - e^{-m/M})$-approximation to the best k-subset after k steps.
- Forward Stepwise Regression produces a $(1 - e^{-\gamma})$-approximation to the best k-subset after k steps.

We also show that if Forward Stepwise Regression is used to select more than k features, we can approximate the best k-sparse feature performance within an *arbitrary* accuracy. Finally, under additional assumptions, we derive statistical guar-

antees for convergence of the greedily selected parameter to the optimal sparse solution. Note that our results yield stronger performance guarantees even for cases that have been previously studied under the same assumptions. For example, for linear regression we obtain a better exponent in the approximation factor of OMP compared to previous state-of-the-art [16] (see Remark 8).

One implication of our work is that weak submodularity seems to be a sharper technical tool than RSC, as any function satisfying the latter also satisfies the former. Das and Kempe [16] noted that it is easy to find problems which satisfy weak submodularity but not RSC, emphasizing the limitations of spectral techniques versus submodularity. We show this holds beyond linear regression, for any likelihood function.

Our connection between restricted strong convexity and weak submodularity has many benefits. First, the weak submodularity framework can now be used to develop theory for additional problems where spectral conditions would be overly restrictive or unwarranted. For example, we show that RSC assumptions play an important role in characterizing distributed greedy maximization [81] and rank constrained matrix optimization [82]. Second, this framework allows statisticians to draw from classical results and recent advances in the field of (weak) submodular set function theory to provide general guarantees on the performance of greedy algorithms [83, 84].

3.2 Related Work

There have been a wide range of techniques developed for solving problems with sparsity constraints. These include using the Lasso, greedy selection methods (such as Forward Stagewise/Stepwise Regressions [85], OMP, and CoSaMP [86]), forward-backward methods [87,88], Pareto optimization [89], exponentially weighted aggregation [90], and truncated gradient methods [91]. Under the restricted strong convexity and smoothness assumptions that will be outlined in the next section, forward-backward methods can in fact recover the correct support of the optimal set of parameters under an assumption on the smallest value of the optimal variable as it relates to the gradient. In contrast, the results derived in our setting for sparse GLMs allow one to provide recovery guarantees at various sparsity levels regardless of the optimal solution, with only information on the desired sparsity level and the RSC and RSM parameters. This is again in contrast to the other work that also needs information on the smallest nonzero value in the optimal set of coefficients, as well as an upper bound on the gradient of the objective at this optimal set.

Focusing explicitly on OMP, most previous results require the strong RIP assumption, whereas we only require the weaker RSC and RSM assumptions. In our setting of arbitrary model conditions, OMP requires RIP as highlighted in Corollary 2.1 of Zhang [92]. However, we do note that under certain stochastic assumptions, for instance independent noise, the results established in those works can provide sharper guarantees with respect to the number of samples required by a factor on the order of $\log[(k \log p)/n]$ (See Section 3.5). Nevertheless,

we emphasize that our results apply under arbitrary assumptions on the noise and use only RSC and RSM assumptions.

In [16], Das and Kempe's framework optimizes the goodness of fit parameter R^2 in linear regression. We derive similar results without relying on the closed-form solution to least squares. Greedy algorithms are prevalent in compressed sensing literature [86] and statistical learning theory [93]. Greedy methods for sparsity constrained regression were analyzed in [87, 88, 91, 94–96] under assumptions similar to ours but without connections to submodularity. Convergence guarantees for ℓ_1 regularized regression were given for exponential families in [97], and for general nonlinear functions in [98]. However, the latter requires additional assumptions such as knowledge of the nonlinearity and bounds on the loss function's derivatives, which can again be derived under appropriate stochasticity and model assumptions.

Classical results on submodular optimization [15, 99, 100] typically do not scale to large-scale applications. Therefore, several recent algorithms improve efficiency at the expense of slightly weaker guarantees [19, 22, 81, 101–104]. Submodularity has been used recently in the context of active learning [17, 76, 78]. In this setup, the task is to select predictive data points instead of features. Recently, [105], [106], and [107] obtained constant factor guarantees for greedy algorithms using techniques from submodularity even though the problems considered were not strictly submodular. These results solve specific problems and do not draw a general connection to RSC. In [83], the authors show tight bounds for maximizing cardinality constrained, weakly submodular set functions that also have bounded

curvature. For maximizing weakly submodular functions subject to more general matroid constraints, [84] recently proved that a randomized greedy forward stepwise algorithm has a constant factor approximation guarantee. Strong convexity, smoothness, and submodularity ratio were used in [108] to study robust support selection and robust Bayesian optimization. More information about related relaxations of submodularity can be found in Appendix C.

There are deep connections between convexity and submodularity [79]. For example, the convex closure of a submodular function can be tractably computed as its Lovász extension [109]. This connection is fundamental to providing polynomial-time minimization algorithms for submodular set functions [110,111]. Similarly, another continuous extension of set functions, called the multilinear extension is vital for algorithmic development of constant factor approximation guarantees for submodular maximization [112]. A more detailed study of convexity and concavity-like properties of submodular functions was presented in [113]. More recent works exploit similar connections to provide constant factor approximation guarantees for a class of non-convex functions [18, 114, 115].

3.3 Preliminaries

Recall that a set function $f(\cdot) : 2^{[p]} \mapsto \mathbb{R}$ is called submodular if and only if for all $A, B \subseteq [p]$, $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$. Intuitively, submodular functions have a *diminishing returns* property. This becomes clear when A and B are disjoint: the sets have less value taken together than they have individually. We also state an equivalent definition:

Definition 3.3.1 (Proposition 2.3 in [79]). *$f(\cdot)$ is submodular if for all $\mathsf{A} \subseteq [p]$ and $j,k \in [p]\backslash \mathsf{A}$,*

$$f(\mathsf{A} \cup \{k\}) - f(\mathsf{A}) \geq f(\mathsf{A} \cup \{j,k\}) - f(\mathsf{A} \cup \{j\}).$$

The function is called normalized if $f(\emptyset) = 0$ and monotone if and only if $f(\mathsf{A}) \leq f(\mathsf{B})$ for all $\mathsf{A} \subseteq \mathsf{B}$. A seminal result by Nemhauser *et al.* [15] shows that greedy maximization of a monotone, submodular function (Algorithm 6 in Section 3.3.2) returns a set with value within a factor of $(1 - 1/e)$ from the optimum set of the same size. This has been the starting point for several algorithmic advances for large-scale combinatorial optimization, including stochastic, distributed, and streaming algorithms [81, 102, 103, 116].

Next, we define the submodularity ratio of a monotone set function.

Definition 3.3.2 (Submodularity Ratio [16], Weak Submodularity). *Let $\mathsf{S}, \mathsf{L} \subset [p]$ be two disjoint sets, and $f(\cdot) : 2^{[p]} \mapsto \mathbb{R}$. The submodularity ratio of L with respect to S is given by*

$$\gamma_{\mathsf{L},\mathsf{S}} := \frac{\sum_{j \in \mathsf{S}} [f(\mathsf{L} \cup \{j\}) - f(\mathsf{L})]}{f(\mathsf{L} \cup \mathsf{S}) - f(\mathsf{L})} \quad . \tag{3.2}$$

The submodularity ratio of a set U with respect to an integer k is given by

$$\gamma_{\mathsf{U},k} := \min_{\substack{\mathsf{L},\mathsf{S}:\mathsf{L}\cap\mathsf{S}=\emptyset, \\ \mathsf{L}\subseteq\mathsf{U}, |\mathsf{S}|\leq k}} \gamma_{\mathsf{L},\mathsf{S}} \quad . \tag{3.3}$$

Let $\gamma > 0$. We call a function γ-weakly submodular at a set U and an integer k if $\gamma_{\mathsf{U},k} \geq \gamma$.

It is straightforward to show that $f(\cdot)$ is submodular if and only if $\gamma_{\mathsf{L},\mathsf{S}} \geq 1$ for all sets L and S. In our application, $0 < \gamma_{\mathsf{L},\mathsf{S}} \leq 1$ which provides a notion of *weak submodularity* in the sense that even though the function is not submodular, it still provides provable bounds of performance of greedy selections. Our notion of weak submodularity based on [16] is different from the definitions in [117,118].[2] In Appendix F of [83], it was shown that our definition does not imply [118] and vice versa.

Next we define the restricted versions of strong concavity and smoothness, consistent with [74, 120].

Definition 3.3.3 (Restricted Strong Concavity, Restricted Smoothness)**.** *A function $l : \mathbb{R}^p \mapsto \mathbb{R}$ is said to be restricted strong concave with parameter m_Ω and restricted smooth with parameter M_Ω on a domain $\Omega \subset \mathbb{R}^p \times \mathbb{R}^p$ if for all $(\mathbf{x}, \mathbf{y}) \in \Omega$,*

$$-\frac{m_\Omega}{2}\|\mathbf{y}-\mathbf{x}\|_2^2 \geq l(\mathbf{y}) - l(\mathbf{x}) - \langle \nabla l(\mathbf{x}), \mathbf{y}-\mathbf{x}\rangle \geq -\frac{M_\Omega}{2}\|\mathbf{y}-\mathbf{x}\|_2^2 \ .$$

Remark 1. *If a function $l(\cdot)$ has restricted strong concavity parameter m, then its negative $-l(\cdot)$ has restricted strong convexity parameter m. In the sequel, we will use these properties interchangeably for maximum likelihood estimation where $l(\cdot)$ is the log likelihood function and $-l(\cdot)$ is the data fit loss.*

If $\Omega' \subseteq \Omega$, then $M_{\Omega'} \leq M_\Omega$ and $m_{\Omega'} \geq m_\Omega$. With slight abuse of notation, unless stated otherwise let (m_k, M_k) denote the RSC and RSM parameters on the domain Ω_k of all pairs of k-sparse vectors that differ in at most k entries, *i.e.* $\Omega_k :=$

[2] A revised version of [118] instead uses the term *proportional submodularity* [119].

$\{(\mathbf{x},\mathbf{y}) : \|\mathbf{x}\|_0 \leq k,\ \|\mathbf{y}\|_0 \leq k,\ \|\mathbf{x}-\mathbf{y}\|_0 \leq k\}$. If $j \leq k$, then $M_j \leq M_k$ and $m_j \geq m_k$. In addition, denote $\tilde{\Omega}_k := \{(\mathbf{x},\mathbf{y}) : \|\mathbf{x}\|_0 \leq k,\ \|\mathbf{y}\|_0 \leq k,\ \|\mathbf{x}-\mathbf{y}\|_0 \leq 1\}$ with corresponding smoothness parameter $\tilde{M}_k$, which is clearly greater than or equal to $\tilde{M}_1 = M_1$.

3.3.1 Sparsity Constrained Generalized Linear Regression

Due to its combinatorial nature, there has been a tremendous amount of effort in developing computationally tractable and fundamentally sound methods to solve the subset selection problem approximately. In this section we provide background on various problems that arise in subset selection. Our focus here will be on sparse regression problems. We will assume that we obtain n observations of the form $(\mathbf{x}_i, y_i)$. For now we make no assumptions regarding how the data is generated, but wish to model the interaction between $\mathbf{x}_i \in \mathbb{R}^p$ and $y_i \in \mathbb{R}$ as

$$y_i = g(\langle \mathbf{x}_i, \boldsymbol{\beta}^* \rangle) + \text{noise},$$

for some known link function g and a sparse vector $\boldsymbol{\beta}^*$. Each feature observation is a row in the $n \times p$ design matrix $\mathbf{X}$. The above is called a generalized linear model, or GLM, and arises as the maximum likelihood estimate of data drawn from a canonical exponential family, *i.e.* normal, Bernoulli, Dirichlet, negative binomial, etc. [121]. Another interpretation is in minimizing the average Bregman divergence between the response y_i and the mean parameter $\langle \mathbf{x}_i, \beta \rangle$. There has been a large body of literature studying this method's statistical properties. These include establishing sparsistency, parameter consistency, and prediction error [74,

97, 122]. We refer the reader to the standard literature for more details on GLMs and exponential families [121, 123].

3.3.2 Support Selection Algorithms

We study general M-estimators of the form (3.1) for some function $l(\cdot)$. Note that $l(\cdot)$ will implicitly depend on our specific dataset, but we hide that for ease of notation. One common choice of $l(\cdot)$ is the log likelihood of a parametric distribution. [16] considers the specific case of maximizing the R^2 objective. Through a simple transformation, that is equivalent to maximizing the log likelihood of the parametric distribution that arises from the model $y_i = \langle \mathbf{x}_i, \boldsymbol{\beta}^* \rangle + w$ where $w \sim N(0, \sigma^2)$. If we let $\widehat{\boldsymbol{\beta}}^s$ be the s-sparse solution derived, and again let $\bar{\boldsymbol{\beta}}^k$ be the best k-sparse parameter, then we wish to bound

$$l(\widehat{\boldsymbol{\beta}}^s) \geq (1 - \epsilon) l(\bar{\boldsymbol{\beta}}^k),$$

without any assumptions on the underlying sparsity or a *true* parameter.

For a concave function $l(\cdot) : \mathbb{R}^p \mapsto \mathbb{R}$, we can define an equivalent set function $\bar{f}(\cdot) : 2^{[p]} \mapsto \mathbb{R}$ so that $\bar{f}(\mathrm{S}) = \max_{\mathrm{supp}(\mathbf{x}) \subseteq \mathrm{S}} l(\mathbf{x})$. The problem of support selection for a given integer k is then $\max_{|\mathrm{S}| \leq k} \bar{f}(\mathrm{S})$. Recall that a vector is k-sparse if it is 0 on all but k indices. We provide approximation guarantees on the *normalized* set function defined as $f(\mathrm{S}) = \bar{f}(\mathrm{S}) - \bar{f}(\emptyset)$. The support selection problem is thus equivalent to finding the k-sparse vector $\boldsymbol{\beta}$ that maximizes $l(\boldsymbol{\beta})$:

$$\max_{\mathrm{S}:|\mathrm{S}| \leq k} f(\mathrm{S}) \Leftrightarrow \max_{\substack{\boldsymbol{\beta}: \boldsymbol{\beta}_{\mathrm{S}^c} = 0 \\ |\mathrm{S}| \leq k}} l(\boldsymbol{\beta}) - l(\mathbf{0}) \quad . \tag{3.4}$$

Let $\boldsymbol{\beta}^{(\mathsf{A})}$ denote the $\boldsymbol{\beta}$ maximizing $f(\mathsf{A})$, and let $\boldsymbol{\beta}_{\mathsf{B}}^{(\mathsf{A})}$ denote $\boldsymbol{\beta}^{(\mathsf{A})}$ restricted to the coordinates specified by B. We present three support selection strategies for the set function $f(\cdot)$ that are simple to implement and are widely used.

Oblivious Algorithm: One natural strategy is to select the top k features ranked by their individual improvement over a null model, using a goodness of fit metric such as R^2 or p-value. This is referred to as the *Oblivious* algorithm, shown as Algorithm 5. In the context of linear regression, this is simply Marginal Regression. While it is computationally inexpensive and parallelizes easily, the Oblivious algorithm does not account for dependencies or redundancies in the span of features.

Forward Stepwise Algorithm: A less extreme greedy approach would check for incremental gain at each step using nested models. This is referred to as the *Forward Stepwise* algorithm, presented as Algorithm 6. Given a set of features S is already selected, choose the feature with largest marginal gain, *i.e.* select $\{j\}$ such that $\mathsf{S} \cup \{j\}$ has the most improvement over S. All regression coefficients are updated each time a new feature is added. In the case of submodular set functions, this returns a solution that is provably within a constant factor of the optimum [15].

Algorithm 5 Oblivious Support Selection

Input: sparsity parameter k, set function $f(\cdot) : 2^{[p]} \mapsto \mathbb{R}$
for $i = 1 \dots p$ **do**
  $\mathbf{v}[i] \leftarrow f(\{i\})$
end for
$\mathrm{S}_k \leftarrow$ indices corresponding to the top k values of $\mathbf{v}$
return S_k, $f(\mathrm{S}_k)$.

Algorithm 6 Forward Stepwise Selection

Input: sparsity parameter k, set function $f(\cdot) : 2^{[p]} \mapsto \mathbb{R}$
$\mathrm{S}_0^G \leftarrow \emptyset$
for $i = 1 \dots k$ **do**
  $s \leftarrow \mathrm{argmax}_{j \in [p] \setminus \mathrm{S}_{i-1}} f(\mathrm{S}_{i-1}^G \cup \{j\}) - f(\mathrm{S}_{i-1}^G)$
  $\mathrm{S}_i^G \leftarrow \mathrm{S}_{i-1}^G \cup \{s\}$
end for
return S_k^G, $f(\mathrm{S}_k^G)$.

Algorithm 7 Orthogonal Matching Pursuit

Input: sparsity parameter k, objective function $l(\cdot) : \mathbb{R}^p \mapsto \mathbb{R}$
$\mathrm{S}_0^P \leftarrow \emptyset$
$\mathbf{r} \leftarrow \nabla l(\mathbf{0})$
for $i = 1 \dots k$ **do**
  $s \leftarrow \mathrm{argmax}_j |\langle \mathbf{e}_j, \mathbf{r} \rangle|$
  $\mathrm{S}_i^P \leftarrow \mathrm{S}_{i-1}^P \cup \{s\}$
  $\boldsymbol{\beta}^{(\mathrm{S}_i^P)} \leftarrow \mathrm{argmax}_{\boldsymbol{\beta}:\mathrm{supp}(\boldsymbol{\beta}) \subseteq \mathrm{S}_i^P} l(\boldsymbol{\beta})$
  $\mathbf{r} \leftarrow \nabla l(\boldsymbol{\beta}^{(\mathrm{S}_i^P)})$
end for
return S_k^P, $l(\boldsymbol{\beta}^{(\mathrm{S}_k^P)})$.

Generalized OMP: Another approach is to choose features which correlate well with the orthogonal complement of what has already been selected. Using

(3.4) and an appropriately chosen model, we can define the gradient evaluated at the current parameter $\boldsymbol{\beta}$ to be a residual term. In *Orthogonal Matching Pursuit*, features are selected to maximize the inner product with this residual, as shown in line 5 of Algorithm 7. Here $\mathbf{e}_j$ represents a unit vector with a 1 in coordinate j and zeros in the other $p-1$ coordinates. OMP requires much less computation than forward stepwise selection, since the feature comparison is done via an n-dimensional inner product rather than a regression score. A detailed discussion can be found in [124].

3.4 Approximation Guarantees

In this section, we derive theoretical lower bounds on the submodularity ratio based on strong concavity and strong smoothness of a function $l(\cdot)$. We show that if the concavity parameter is bounded away from 0 and the smoothness parameter is finite, then the submodularity ratio is also bounded away from 0, which allows approximation guarantees for Algorithms 5–7. While our proof techniques differ substantially, the outline of this section follows that of [16] which obtained approximation guarantees for support selections for linear regression. While our results are applicable to general functions, in [73] we discuss a direct application of maximum likelihood estimation for sparse generalized linear models.

We assume a differentiable function $l : \mathbb{R}^p \mapsto \mathbb{R}$. Recall that we can define the equivalent, normalized, monotone set function $f : 2^{[p]} \mapsto \mathbb{R}$ for a selected support as $f(\mathrm{S}) = \max_{\mathrm{supp}(\boldsymbol{\beta})\subseteq \mathrm{S}} l(\boldsymbol{\beta}) - l(\mathbf{0})$. We will use set functions wherever possible to simplify the notation.

We now present our main result as Theorem 3.4.1, a bound on a function's submodularity ratio $\gamma_{U,k}$ in terms of its strong concavity and smoothness parameters (see Definitions 3.3.2–3.3.3). Proofs of lemmas and theorems omitted from this section can be found in Appendix E.

Theorem 3.4.1 (RSC/RSM Implies Weak Submodularity)**.** *Define $f(S)$ as in (3.4), with a function $l(\cdot)$ that is $(m_{|U|+k}, M_{|U|+k})$-(strongly concave, smooth) on $\Omega_{|U|+k}$ and $\tilde{M}_{|U|+1}$ smooth on $\tilde{\Omega}_{|U|+1}$. Then the submodularity ratio $\gamma_{U,k}$ is lower bounded by*

$$\gamma_{U,k} \geq \frac{m_{|U|+k}}{\tilde{M}_{|U|+1}} \geq \frac{m_{|U|+k}}{M_{|U|+k}} \quad . \tag{3.5}$$

Remark 2. *In the case of linear least-squares regression, m and M become sparse eigenvalues of the covariance matrix, i.e. $m_{|U|+k} = \lambda_{\min}(|U| + k) \geq 0$ and $\tilde{M}_{|U|+1} = \lambda_{\max}(1) = 1$. Thus Theorem 3.4.1 becomes $\gamma_{U,k} \geq \lambda_{\min}(|U| + k)$, i.e. "RIP implies weak submodularity," consistent with Lemma 2.4 of [16].*

Remark 3. *Since $m/M \leq 1$, this method cannot prove that the function is submodular (even on a restricted set of features). However, the guarantees in this section only require weak submodularity.*

Remark 4. *Theorem 3.4.1 has the following geometric interpretation: the submodularity ratio of $f(S)$ is bounded in terms of the maximum curvature of $l(\cdot)$ over the domain $\tilde{\Omega}_{|U|+1}$ and the minimum curvature of $l(\cdot)$ over the (larger) domain $\Omega_{|U|+k}$. The upper-curvature bound effectively controls the maximum amount that each individual function coordinate can influence the function value. The lower-curvature bound provides a lower-bound on the improvement of adding all features at once.*

Hence, loosely speaking the submodularity ratio bound will be the ratio of both of these quantities. This intuition is used more formally in the proof (Appendix E), where for a k-sparse set S and $j \in \mathsf{S}$, $l(\boldsymbol{\beta}^{(\mathsf{U})})$ is perturbed by scaled projections of $\nabla l(\boldsymbol{\beta}^{(\mathsf{U})})$ onto $\boldsymbol{\beta}_j^{(\mathsf{U}\cup\mathsf{S})}$ and $\mathbf{e}_{\mathsf{S}}$, respectively.

Theorem 3.4.1 allows us to generalize several results of [16], starting with the following lemma:

Lemma 3.4.1. *Let $1 \leq k \leq n$.*

$$\begin{aligned} f([k]) &\geq \max\left\{\frac{1}{k}, \frac{m_1}{4M_k}\left(3 + \frac{m_1}{M_1}\right)\right\} \sum_{j=1}^{k} f(j) \\ &\geq \max\left\{\frac{1}{k}, \frac{m_k}{4M_k}\left(3 + \frac{m_k}{M_k}\right)\right\} \sum_{j=1}^{k} f(j) \quad . \end{aligned}$$

Now we present our first performance guarantee for feature selection.

Theorem 3.4.2 (Oblivious Algorithm Guarantee)**.** *Define $f(\mathsf{S})$ as in (3.4), with a function $l(\cdot)$ that is M_k-smooth and m_k-strongly concave on Ω_k. Let f^{OBL} be the value at the set selected by the Oblivious algorithm, and let f^{OPT} be the optimal value over all sets of size k. Then*

$$\begin{aligned} f^{OBL} &\geq \max\left\{\frac{m_k}{kM_1}, \frac{m_k m_1}{4M_k M_1}\left(3 + \frac{m_1}{M_1}\right)\right\} f^{OPT} \\ &\geq \max\left\{\frac{m_k}{kM_k}, \frac{3m_k^2}{4M_k^2}, \frac{m_k^3}{M_k^3}\right\} f^{OPT} \quad . \end{aligned}$$

Remark 5. *When the function is modular, i.e. $m_\Omega = M_\Omega$ for all Ω, then $f^{OBL} = f^{OPT}$ and the bound in Theorem 3.4.2 holds with equality.*

Next, we prove a stronger, constant factor approximation guarantee for the greedy, Forward Stepwise algorithm.

Theorem 3.4.3 (Forward Stepwise Algorithm Guarantee). *Define $f(S)$ as in (3.4), with a function that is M-smooth and m-strongly concave on Ω_{2k}. Let S_k^G be the set selected by the FS algorithm and S^* be the optimal set of size k corresponding to values f^{FS} and f^{OPT}. Then*

$$f^{FS} \geq \left(1 - e^{-\gamma_{S_k^G,k}}\right) f^{OPT} \geq \left(1 - e^{-m/M}\right) f^{OPT} \quad . \tag{3.6}$$

Remark 6. *This constant factor bound can be improved by running the Forward Stepwise algorithm for $r > k$ steps. The proof of Theorem 3.4.3 generalizes to compare performance of r greedy iterations to the optimal k-subset of features. This generalized bound does not necessarily approach 1 as $r \to \infty$, however, since $\gamma_{S_r^G,k}$ is a decreasing function of r.*

Corollary 3.4.1. *Let f^{FS+} denote the solution obtained after r iterations of the Forward Stepwise algorithm, and let f^{OPT} be the objective at the optimal k-subset of features. Let $\gamma = \gamma_{S_r^G,k}$ be the submodularity ratio associated with the output of f^{FS+} and k. Then*

$$f^{FS+} \geq (1 - e^{-\gamma(r/k)}) f^{OPT} \quad .$$

In particular, setting $r = ck$ corresponds to a $(1 - e^{-c\gamma})$-approximation, and setting $r = k \log n$ corresponds to a $(1 - n^{-\gamma})$-approximation.

Corollary 3.4.1 is useful when γ can be bounded on larger support sets. We next present approximation guarantees when γ can only be bounded on smaller support sets.

Theorem 3.4.4. *Define $f(\mathrm{S})$ as in (3.4), with a function $l(\cdot)$ that is m'-strongly concave on Ω_k and M'-smooth on $\tilde{\Omega}_k$. Let S_k^G be the set of features selected by the Forward Stepwise algorithm and S_k be the optimal feature set on k variables corresponding to values f^G and f^{OPT}. Then*

$$f^{FS} \geq \Theta\left(2^{-M'/m'}\right)\left(1 - e^{-m'/M'}\right) f^{OPT} .$$

Remark 7. *We note that our bounds are loose for certain special cases like modular functions and linear regression. These require making use of additional tools and specific properties of the function and data at hand (see [16]).*

Orthogonal Matching Pursuit is more computationally efficient than forward stepwise regression, since step i only fits one regression instead of $p - i$. Thus we have a weaker guarantee than Theorem 3.4.3. Similar to Corollary 3.4.1, this result generalizes to running OMP for $r > k$ iterations.

Theorem 3.4.5 (OMP Algorithm Guarantee)**.** *Define $f(S)$ as in (3.4), with a log-likelihood function that is (M,m)-(smooth, strongly concave) on Ω_{2k}. Let f^{OMP} be the value at the set of features selected by the OMP algorithm and f^{OPT} be the optimal value over all sets of size k. Then*

$$f^{OMP} \geq \left(1 - e^{-m/M}\right) f^{OPT} .$$

Corollary 3.4.2. *Let f^{P_+} denote the solution obtained after r iterations of the OMP algorithm, and let f^{OPT} be the objective at the optimal k-subset of features. Let $\alpha = (m/M)$ be the ratio associated with the output of f^{P_+} and k. Then*

$$f^{P_+} \geq (1 - e^{-\alpha(r/k)}) f^{OPT} .$$

In particular, setting $r = ck$ corresponds to a $(1 - e^{-c\alpha})$-approximation, and setting $r = k \log n$ corresponds to a $(1 - n^{-\alpha})$-approximation.

Remark 8. *Theorem 3.4.5 and Corollary 3.4.2 improve on the approximation guarantee of [16] by a factor of γ in the exponent. Previous work obtained the approximation factor $1 - e^{-\gamma \lambda_{\min}(2k)}$, whereas the proof of Theorem 3.4.5 establishes $1 - e^{-\lambda_{\min}(2k)}$. Therefore we obtain a better exponent for linear regression and also generalize to any likelihood function. Theorem 3.4.3 also gives intuition on when the performance of OMP will differ from that of Forward Selection, i.e. when the inequality (3.6) is loose.*

3.5 Statistical Recovery Guarantees

Understanding optimization guarantees are useful, but they do not clearly translate to bounds on parameter recovery. Below we present a general theorem that allows us to derive parameter bounds. When combined with Section 3.4, it produces recovery guarantees for greedy algorithms as special cases.

Theorem 3.5.1 (Parameter Recovery Guarantees)**.** *Suppose that after r iterations to approximate a function evaluated at a set S_s^* of cardinality s, we have the guarantee that*

$$f(\mathrm{S}_r) \geq C_{s,r} f(\mathrm{S}_s^*) \ .$$

Recall that $f(\mathrm{S}_r) = \max_{supp(\boldsymbol{\beta}) \subset \mathrm{S}_r} l(\boldsymbol{\beta}) - l(\mathbf{0})$. Let $\widehat{\boldsymbol{\beta}}^r$ be the solution to the optimization problem and consider any arbitrary s-sparse vector $\boldsymbol{\beta}^s$ with support on S_s^*. Then, under m_{s+r} RSC on Ω_{s+r} we have that

$$\|\widehat{\boldsymbol{\beta}}^r - \boldsymbol{\beta}^s\|_2^2 \leq \frac{4}{m_{s+r}^2} \|\nabla l(\boldsymbol{\beta}^s)\|_{2,(s+r)}^2 + \frac{4}{m_{s+r}} (1 - C_{s,r})[l(\boldsymbol{\beta}^s) - l(\mathbf{0})] \ .$$

For the remainder of this section, we consider several cases of Theorem 3.5.1 and compare to results from previous work.

3.5.1 Forward Selection with Linear Regression Model

First we will consider a special case of Algorithm 6 for linear regression where the rows of the design matrices are $N(0, \Sigma)$ for a covariance matrix of the form $\Sigma = \mathbf{I} + \mathbf{1}\mathbf{1}^T$. Further, we assume the model

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta}^* + \mathbf{w} \ ,$$

where $\|\boldsymbol{\beta}^*\|_2 \leq 1$ and is s-sparse, the rows of $\mathbf{X} \in \mathbb{R}^{n \times p}$ are $N(0, \Sigma)$, and $w_i \sim N(0, \sigma^2)$ are i.i.d. We also take $l(\boldsymbol{\beta}) = 1/n \|\mathbf{X}\boldsymbol{\beta} - \mathbf{y}\|_2^2$.

Corollary 3.5.1. *Given the above setup, if* $(s+r)\sigma^2 \log p = o(n)$ *and* $r = \Omega(s \log n)$, *then the parameter error goes to zero with high probability as* $n \to \infty$.

3.5.2 Orthogonal Matching Pursuit with Linear Regression Model

Next, we consider the results of Zhang, which provide parameter recovery bounds in the case of OMP (Algorithm 7). The simplest comparison is to contrast our results with Corollary 2.2 of [92]. Consider the linear regression model above with an original s-sparse vector, r iterations of the algorithm, and a spiked identity covariance model, $\Sigma = (1 - a)\mathbf{I} + a\mathbf{1}\mathbf{1}^T$.

Proposition 2. *While Theorem 3.5.1 holds for any a, Zhang [92] requires that a does not exceed* $\frac{1}{s+1}$.

Proof. Zhang requires the RIP condition to hold, namely $M_s \leq 2m_{s+r}$. We know that the difference between means of $2\lambda_{\min}(s+r)$ and $\lambda_{\max}(s)$ is $\Delta = 1-a-as$. Since $\Delta/2 \leq \mu/2$ in both cases and χ^2 variables concentrate within constant factors of their means, we have $M_s \leq 3/2(1-a) + as/2 \leq 2m_{s+r}$. However, $\Delta > 0 \Leftrightarrow s \leq 1/a - 1$. Rearranging, we have $a \leq \frac{1}{s+1}$. Thus, as has been noted in prior work, the RIP condition will not hold for the spiked model in settings where a is much larger than $\frac{1}{s+1}$. □

Nevertheless, we can still proceed and assume that the RIP condition is not required. In that case, the bound established in [92] shows

$$\|\widehat{\boldsymbol{\beta}}^r - \boldsymbol{\beta}^*\|_2^2 \leq 24 M_{s+r} \|\mathbf{X}\boldsymbol{\beta}^* - \mathbf{y}\|_2^2 / m_{s+r}^2 \ .$$

When M_{s+r} and m_{s+r} are of the same order, then this result is better than ours by log factors. However, when we consider a case like the spiked covariance model, then our results are better by a factor of s in terms of statistical accuracy but worse by a factor of $\log n$ with respect to sample complexity.

3.5.3 Orthogonal Matching Pursuit with Logistic Regression Model

Finally, consider our bounds for OMP (Algorithm 7) in the case of logistic regression. Applying our approximation guarantees in Theorem 3.4.5 matches the bound given by Theorem 2 of [94] up to constant factors. However, their guarantee for parameter recovery requires a condition that is only known to be satisfied under incoherence assumptions. Our Theorem 3.5.1 holds more generally. Their conditions on exact recovery are incomparable with our statistical error bounds.

3.6 Experiments

Next we evaluate the performance of our greedy algorithms with feature selection experiments on simulated and real-world datasets. A bias term β_0 is added to the regression by augmenting the design matrix with a column of ones.

The Data: A synthetic experiment was conducted as follows: first each row of a 600×200 design matrix $\mathbf{X}$ is generated independently according to a first order AR process ($\alpha = 0.3$ and noise variance $\sigma^2 = 5$). This ensures that the features are heavily correlated with each other. Bernoulli ± 1 (*i.e.*, Rademacher) random variables are placed on 50 random indices to form the true support $\bar{\boldsymbol{\beta}}^k$, and scaled such that $\|\boldsymbol{\beta}\|_2^2 = 5$. Then responses $\mathbf{y}$ are computed via a logistic model. We also conduct an experiment on a subset of the RCV1 Binary text classification dataset [125]. 10,000 training and test samples are used in 47,236 dimensions. Since there is no ground truth, a logistic regression is fit using a subset of at most 700 features.

Algorithms and Metrics: The Oblivious, Forward Stepwise (FS), and OMP algorithms were implemented using a logistic log likelihood function given $\mathbf{X}$ and $\mathbf{y}$ on the given design matrix and response:

$$l(\boldsymbol{\beta}) = \sum_{i=1}^{n} \log p(y_i \mid \mathbf{x}_i; \boldsymbol{\beta}) = \sum_{i=1}^{n} \log(1 + e^{\langle \mathbf{x}_i, \boldsymbol{\beta} \rangle}) - y_i \langle \mathbf{x}_i, \boldsymbol{\beta} \rangle \ . \tag{3.7}$$

We implemented 3 additional algorithms. *Lasso* fits a logistic regression model with ℓ_1 regularization. *Lasso-Pipeline* recovers the sparse support using Lasso and then fits regression coefficients on this support with a separate, unregularized

model. The regularization parameter was swept to achieve outputs with varying sparsity levels. *Forward Backward* (FoBa) [126] first runs FS at each step and then drops any features if doing so would decrease the objective by less than half the latest marginal gain.

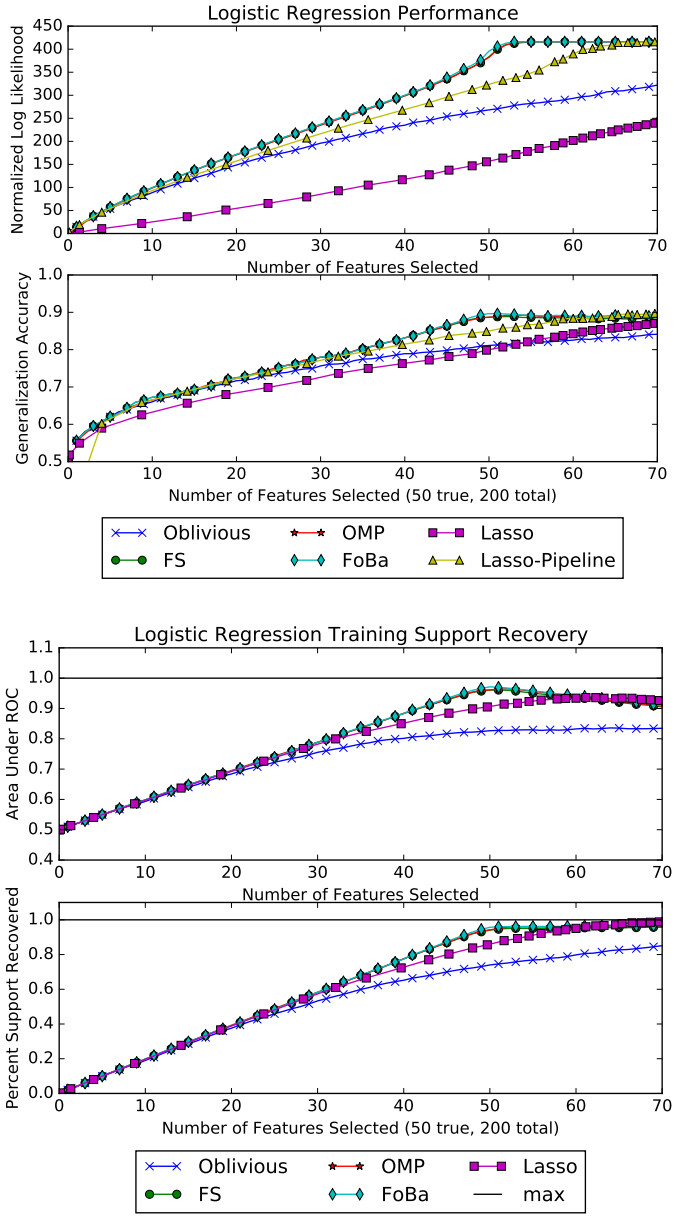
Our main metric for each algorithm is the normalized objective function $l(\widehat{\boldsymbol{\beta}}^s) - l(\mathbf{0})$ for the output sparsity $s \in \{1, \ldots, 70\}$. We also compare the sets $\text{supp}(\widehat{\boldsymbol{\beta}}^s)$ and $\text{supp}(\bar{\boldsymbol{\beta}}^k)$ using area under ROC and percent of true support recovered. Finally, we measure generalization accuracy by drawing additional observations $(\mathbf{x}_i, y_i)$ from the same distribution as the training data.

Results: Figure 3.1 shows the results of our synthetic experiment averaged over 20 runs. For all metrics, Oblivious performs worse than OMP which is slightly worse than FS and FoBa. This matches intuition and the series of bounds in Section 3.4. We also see that the Lasso-Pipeline performs noticeably worse than all algorithms except Oblivious and Lasso. This suggests that greedy feature selection degrades more gracefully than Lasso in the case of correlated features.

Figure 3.2 shows similar results for the high-dimensional RCV1 Binary dataset. Due to their large running time complexity, FS and FoBa were omitted. While all algorithms have roughly the same generalization accuracy using 300 features, OMP has the largest log likelihood.

3.7 Conclusions

We have extended the results of [16] and shown that functions satisfying RSC also satisfy a relaxed form of submodularity that can be used to analyze the performance of greedy algorithms compared to the best sparse solution. Experimental results confirm that greedy feature selection outperforms regularized approaches in a nonlinear regression model. Directions for future work include similar analysis for other greedy algorithms that incorporate group sparsity [74] or thresholding, and applications beyond sparse regression. Bounds on dictionary selection (analogous to those in [16]) also apply to general likelihood functions satisfying RSC and RSM.



(b)

Figure 3.1: Synthetic Dataset - $\alpha = 0.3$, $n = 600$ training and test samples, $p = 200$ dimensions with true support on 50 features, averaged over 20 runs. (a) The greedy algorithms perform better than Lasso and Oblivious algorithms, but beyond 50 steps they overfit to noise in the training data. While Lasso outperforms Oblivious in support recovery (b), its regression suffers from regularization bias.

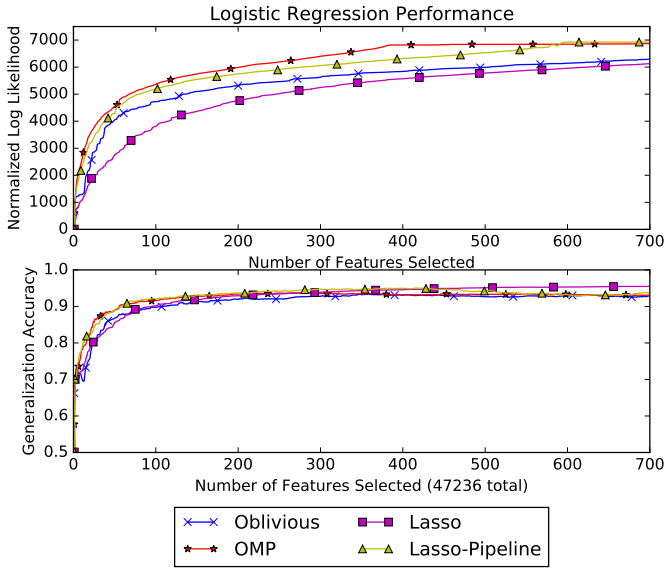


Figure 3.2: RCV1 Binary Dataset - $n = 10{,}000$, $p = 47{,}236$. OMP outperforms Lasso-Pipeline.

Chapter 4

Streaming Weak Submodularity: Interpreting Neural Networks on the Fly

4.1 Introduction

Consider the following[1] combinatorial optimization problem. Given a ground set $[N]$ of N elements and a set function $f\colon 2^{[N]} \mapsto \mathbb{R}_{\geq 0}$, find the set S of size k which maximizes $f(\mathrm{S})$. This formulation is at the heart of many machine learning applications such as sparse regression, data summarization, facility location, and graphical model inference. Although the problem is intractable in general, if f is assumed to be *submodular* then many approximation algorithms have been shown to perform provably within a constant factor from the best solution.

Some disadvantages of the standard greedy algorithm of [15] for this problem are that it requires repeated access to each data element and a large total number of function evaluations. This is undesirable in many large-scale machine learning tasks where the entire dataset cannot fit in main memory, or when a sin-

[1]Parts of the material in this chapter are based on the following conference paper: [116] E. R. Elenberg, A. G. Dimakis, M. Feldman, and A. Karbasi. Streaming Weak Submodularity: Interpreting Neural Networks on the Fly. In *NIPS*, pages 4047–4057, 2017. From that material, the dissertation author's primary contributions are design of the parameter a in the approximation guarantees, as well as design, implementation, and analysis of experiments. The dissertation author also assisted with other contributions and is the primary contributor of this paper. Additional material in this chapter is the dissertation author's contribution.

gle function evaluation is time consuming. In our main application, each function evaluation corresponds to inference on a large neural network and can take a few seconds. In contrast, streaming algorithms make a small number of passes (often only one) over the data and have sublinear space complexity, and thus, are ideal for tasks of the above kind.

Recent ideas, algorithms, and techniques from submodular set function theory have been used to derive similar results in much more general settings. For example, Chapter 3 (and corresponding publications [72, 73]) used the concept of *weak submodularity* to derive approximation and parameter recovery guarantees for nonlinear sparse regression. Thus, a natural question is whether recent results on streaming algorithms for maximizing submodular functions [17, 22, 127] extend to the weakly submodular setting.

This chapter answers the above question by providing the first analysis of a streaming algorithm for any class of approximately submodular functions. We use key algorithmic components of Sieve-Streaming [22], namely greedy thresholding and binary search, combined with a novel analysis to prove a constant factor approximation for γ-weakly submodular functions (defined in Section 4.3). Specifically, our contributions are as follows.

- An impossibility result showing that, even for 0.5-weakly submodular objectives, no randomized streaming algorithm which uses $o(N)$ memory can have a constant approximation ratio when the ground set elements arrive in a worst case order.

- STREAK: a greedy, deterministic streaming algorithm for maximizing γ-weakly submodular functions which uses $O(\epsilon^{-1}k \log k)$ memory and has an approximation ratio of $(1-\epsilon)\gamma \cdot \left[4 + \frac{\gamma}{2} - 2\sqrt{\gamma + 4}\right]$ when the ground set elements arrive in a random order.

- An experimental evaluation of our algorithm in two applications: nonlinear sparse regression using pairwise products of features and interpretability of black-box neural network classifiers.

The above theoretical impossibility result is quite surprising since it stands in sharp contrast to known streaming algorithms for submodular objectives achieving a constant approximation ratio even for worst case stream order.

One advantage of our approach is that, while our approximation guarantees are in terms of γ, our algorithm STREAK runs without requiring prior knowledge about the value of γ. This is important since the weak submodularity parameter γ is hard to compute, especially in streaming applications, as a single element can alter γ drastically.

We use our streaming algorithm for neural network interpretability on Inception V3 [23]. For that purpose, we define a new set function maximization problem similar to LIME [24] and apply our framework to approximately maximize this function. Experimentally, we find that our interpretability method produces explanations of similar quality as LIME, but runs approximately 10 times faster.

4.2 Related Work

Monotone submodular set function maximization has been well studied, starting with the classical analysis of greedy forward selection subject to a matroid constraint [15, 99]. For the special case of a uniform matroid constraint, the greedy algorithm achieves an approximation ratio of $1 - 1/e$ [99], and a more involved algorithm obtains this ratio also for general matroid constraints [128]. In general, no polynomial-time algorithm can have a better approximation ratio even for a uniform matroid constraint [129, 130]. However, it is possible to improve upon this bound when the data obeys some additional guarantees [100, 131, 132]. For maximizing nonnegative, not necessarily monotone, submodular functions subject to a general matroid constraint, the state-of-the-art randomized algorithm achieves an approximation ratio of 0.385 [112]. Moreover, for uniform matroids there is also a deterministic algorithm achieving a slightly worse approximation ratio of $1/e$ [133]. The reader is referred to [79] and [134] for surveys on submodular function theory.

A recent line of work aims to develop new algorithms for optimizing submodular functions suitable for large-scale machine learning applications. Algorithmic advances of this kind include Stochastic-Greedy [102], Sieve-Streaming [22], and several distributed approaches [19, 81, 101, 103, 104]. Our algorithm extends ideas found in Sieve-Streaming and uses a different analysis to handle more general functions. Additionally, submodular set functions have been used to prove guarantees for online and active learning problems [17, 76, 78]. Specifically, in the online setting corresponding to our setting (*i.e.*, maximizing a monotone function subject to a cardinality constraint), [135] achieve a competitive ratio of about

0.3178 when the function is submodular.

The concept of weak submodularity was introduced in [16,80], where it was applied to the specific problem of feature selection in linear regression. Their main results state that if the data covariance matrix is not too correlated (using either incoherence or restricted eigenvalue assumptions), then maximizing the goodness of fit $f(S) = R^2_S$ as a function of the feature set S is weakly submodular. This leads to constant factor approximation guarantees for several greedy algorithms. Weak submodularity was connected with Restricted Strong Convexity in Chapter 3 and the corresponding publications [72, 73]. This showed that the same assumptions which imply the success of regularization also lead to guarantees on greedy algorithms. This framework was later used for additional algorithms and applications [81,82]. Other approximate versions of submodularity were used for greedy selection problems in [18, 105, 106, 108, 136]. To the best of our knowledge, this is the first analysis of streaming algorithms for approximately submodular set functions.

Increased interest in interpretable machine learning models has led to extensive study of sparse feature selection methods. For example, [137] consider greedy algorithms for logistic regression, and [98] solve a more general problem using ℓ_1 regularization. Recently, [24] developed a framework called LIME for interpreting black-box neural networks, and [138] proposed a method that requires access to the network's gradients with respect to its inputs. We compare our algorithm to variations of LIME in Section 4.6.2.

4.3 Preliminaries

First we establish some definitions and notation specific to this section. The number of elements in the input stream is assumed to be N, and all big O notation is assumed to be scaling with respect to N. Given a set function f, we often use the discrete derivative $f(\mathsf{B} \mid \mathsf{A}) := f(\mathsf{A} \cup \mathsf{B}) - f(\mathsf{A})$. f is monotone if $f(\mathsf{B} \mid \mathsf{A}) \geq 0, \forall \mathsf{A}, \mathsf{B}$ and nonnegative if $f(\mathsf{A}) \geq 0, \forall \mathsf{A}$. Using this notation one can define weakly submodular functions based on the following ratio.

Definition 4.3.1 (Weak Submodularity, adapted from [16])**.** *A monotone nonnegative set function $f : 2^{[N]} \mapsto \mathbb{R}_{\geq 0}$ is called γ-weakly submodular for an integer r if*

$$\gamma \leq \gamma_r := \min_{\substack{\mathsf{L},\mathsf{S} \subseteq [N]: \\ |\mathsf{L}|, |\mathsf{S} \setminus \mathsf{L}| \leq r}} \frac{\sum_{j \in \mathsf{S} \setminus \mathsf{L}} f(j \mid \mathsf{L})}{f(\mathsf{S} \mid \mathsf{L})} \ ,$$

where the ratio is considered to be equal to 1 *when its numerator and denominator are both* 0*.*

This generalizes submodular functions by relaxing the *diminishing returns* property of discrete derivatives. It is easy to show that f is submodular if and only if $\gamma_N = 1$. Note that this definition allows the two sets to have nonempty intersection, which differs slightly from the one in Chapter 3.

Definition 4.3.2 (Approximation Ratio)**.** *A given streaming maximization algorithm ALG which returns a set* S *has approximation ratio $R \in [0,1]$ if $\mathbf{E}[f(\mathsf{S})] \geq R \cdot f(\mathsf{OPT})$, where* OPT *is the optimal solution and the expectation is over the random decisions of the algorithm and the randomness of the input stream order (when it is random).*

Formally our problem is as follows. Assume that elements from a ground set $[N]$ arrive in a stream at either random or worst case order. The goal is then to design a one pass streaming algorithm that, given oracle access to a nonnegative set function $f : 2^{[N]} \mapsto \mathbb{R}_{\geq 0}$, maintains at most $o(N)$ elements in memory and returns a set S of size at most k approximating

$$\max_{|\mathsf{T}| \leq k} f(\mathsf{T}) \ ,$$

up to an approximation ratio $R(\gamma_k)$. Ideally, this approximation ratio should be as large as possible, and we also want it to be a function of γ_k and nothing else. In particular, we want it to be independent of k and N.

To simplify notation, we use γ in place of γ_k in the rest of the chapter. Additionally, proofs for all our theoretical results are deferred to Appendix F.

4.4 Impossibility Result

To prove our negative result showing that no streaming algorithm for our problem has a constant approximation ratio against a worst case stream order, we first need to construct a weakly submodular set function f_k. Later we use it to construct a bad instance for any given streaming algorithm.

Fix some $k \geq 1$, and consider the ground set $\mathsf{N}_k = \{u_i, v_i\}_{i=1}^k$. For ease of notation, let us define for every subset $\mathsf{S} \subseteq \mathsf{N}_k$

$$u(\mathsf{S}) = |\mathsf{S} \cap \{u_i\}_{i=1}^k| \ , \quad v(\mathsf{S}) = |\mathsf{S} \cap \{v_i\}_{i=1}^k| \ .$$

Now we define the following set function:

$$f_k(\mathsf{S}) = \min\{2 \cdot u(\mathsf{S}) + 1, 2 \cdot v(\mathsf{S})\} \quad \forall\ \mathsf{S} \subseteq \mathsf{N}_k \ . \tag{4.1}$$

Lemma 4.4.1. *f_k is nonnegative, monotone, and 0.5-weakly submodular for the integer $|\mathsf{N}_k|$.*

Since $|\mathsf{N}_k| = 2k$, the maximum value of f_k is $f_k(\mathsf{N}_k) = 2 \cdot v(\mathsf{N}_k) = 2k$. We now extend the ground set of f_k by adding to it an arbitrary large number d of dummy elements which do not affect f_k at all. Clearly, this does not affect the properties of f_k proved in Lemma 4.4.1. However, the introduction of dummy elements allows us to assume that k is an arbitrary small value compared to N, which is necessary for the proof of the next theorem. In a nutshell, this proof is based on the observation that the elements of $\{u_i\}_{i=1}^{k}$ are indistinguishable from the dummy elements as long as no element of $\{v_i\}_{i=1}^{k}$ has arrived yet.

Theorem 4.4.1. *For every constant $c \in (0,1]$ there is a large enough k such that no randomized streaming algorithm that uses $o(N)$ memory to solve $\max_{|\mathsf{S}|\leq 2k} f_k(\mathsf{S})$ has an approximation ratio of c for a worst case stream order.*

We note that f_k has strong properties. In particular, Lemma 4.4.1 implies that it is 0.5-weakly submodular for every $0 \leq r \leq N$. In contrast, the algorithm we show later assumes weak submodularity only for the cardinality constraint k. Thus, the above theorem implies that worst case stream order precludes a constant approximation ratio even for functions with much stronger properties compared to what is necessary for getting a constant approximation ratio when the order is random.

The proof of Theorem 4.4.1 relies critically on the fact that each element is seen exactly once. In other words, once the algorithm decides to discard an ele-

ment from its memory, this element is gone forever, which is a standard assumption for streaming algorithms. Thus, the theorem does not apply to algorithms that use multiple passes over $[N]$, or non-streaming algorithms that use $o(N)$ writable memory, and their analysis remains an interesting open problem.

4.5 Streaming Algorithms

In this section we give a deterministic streaming algorithm for our problem which works in a model in which the stream contains the elements of $[N]$ in a random order. We first describe in Section 4.5.1 such a streaming algorithm assuming access to a value τ which approximates $a\gamma \cdot f(\mathrm{OPT})$, where a is a shorthand for $a = (\sqrt{2 - e^{-\gamma/2}} - 1)/2$. Then, in Section 4.5.2 we explain how this assumption can be removed to obtain Streak and bound its approximation ratio, space complexity, and running time.

4.5.1 Algorithm With Access to τ

Consider Algorithm 8. In addition to the input instance, this algorithm gets a parameter $\tau \in [0, a\gamma \cdot f(\mathrm{OPT})]$. One should think of τ as close to $a\gamma \cdot f(\mathrm{OPT})$, although the following analysis of the algorithm does not rely on it. We provide an outline of the proof, but defer the technical details to Appendix F.

Theorem 4.5.1. *The expected value of the set produced by Algorithm 8 is at least*

$$\frac{\tau}{a} \cdot \frac{3 - e^{-\gamma/2} - 2\sqrt{2 - e^{-\gamma/2}}}{2} = \tau \cdot (\sqrt{2 - e^{-\gamma/2}} - 1) \ .$$

Proof (Sketch). Let $\mathcal{E}$ be the event that $f(\mathrm{S}) < \tau$, where S is the output produced by

Algorithm 8 ThresholdGreedy(f, k, τ)

```
Let S ← ∅.
while there are more elements do
  Let u be the next element.
  if |S| < k and f(u | S) ≥ τ/k then
    Update S ← S ∪ {u}.
  end if
end while
return S
```

Algorithm 8. Clearly $f(\mathrm{S}) \geq \tau$ whenever $\mathcal{E}$ does not occur, and thus, it is possible to lower bound the expected value of $f(\mathrm{S})$ using $\mathcal{E}$ as follows.

Observation 4.5.2. *Let S denote the output of Algorithm 8, then* $\mathbf{E}[f(\mathrm{S})] \geq (1 - \Pr[\mathcal{E}]) \cdot \tau$.

The lower bound given by Observation 4.5.2 is decreasing in $\Pr[\mathcal{E}]$. Proposition 4.5.1 provides another lower bound for $\mathbf{E}[f(\mathrm{S})]$ which increases with $\Pr[\mathcal{E}]$. An important ingredient of the proof of this proposition is the next observation, which implies that the solution produced by Algorithm 8 is always of size smaller than k when $\mathcal{E}$ happens.

Observation 4.5.3. *If at some point Algorithm 8 has a set* S *of size* k*, then* $f(\mathrm{S}) \geq \tau$.

The proof of Proposition 4.5.1 is based on the above observation and on the observation that the random arrival order implies that every time that an element of OPT arrives in the stream we may assume it is a random element out of all the OPT elements that did not arrive yet.

Proposition 4.5.1. *For the set* S *produced by* ThresholdGreedy,

$$\mathbf{E}[f(\mathrm{S})] \geq \frac{1}{2} \cdot \left(\gamma \cdot [\Pr[\mathcal{E}] - e^{-\gamma/2}] \cdot f(\mathrm{OPT}) - 2\tau\right) \quad .$$

The theorem now follows by showing that for every possible value of $\Pr[\mathcal{E}]$ the guarantee of the theorem is implied by either Observation 4.5.2 or Proposition 4.5.1. Specifically, the former happens when $\Pr[\mathcal{E}] \leq 2 - \sqrt{2 - e^{-\gamma/2}}$ and the latter when $\Pr[\mathcal{E}] \geq 2 - \sqrt{2 - e^{-\gamma/2}}$. □

4.5.2 Algorithm Without Access to τ

In this section we explain how to get an algorithm which does not depend on τ. Instead, Streak (Algorithm 9) receives an accuracy parameter $\epsilon \in (0, 1)$. Then, it uses ϵ to run several instances of Algorithm 8 stored in a collection denoted by I. The algorithm maintains two variables throughout its execution: m is the maximum value of a singleton set corresponding to an element that the algorithm already observed, and u_m references an arbitrary element satisfying $f(u_m) = m$.

The collection I is updated as follows after each element arrival. If previously I contained an instance of Algorithm 8 with a given value for τ, and it no longer should contain such an instance, then the instance is simply removed. In contrast, if I did not contain an instance of Algorithm 8 with a given value for τ, and it should now contain such an instance, then a new instance with this value for τ is created. Finally, if I contained an instance of Algorithm 8 with a given value for τ, and it should continue to contain such an instance, then this instance remains in I as is.

Algorithm 9 STREAK(f, k, ϵ)

Let $m \leftarrow 0$, and let I be an (originally empty) collection of instances of Algorithm 8.
while there are more elements **do**
  Let u be the next element.
  if $f(u) \geq m$ **then**
    Update $m \leftarrow f(u)$ and $u_m \leftarrow u$.
  end if
  Update I so that it contains an instance of Algorithm 8 with $\tau = x$ for every $x \in \{(1-\epsilon)^i \mid i \in \mathbf{Z} \text{ and } (1-\epsilon)m/(9k^2) \leq (1-\epsilon)^i \leq mk\}$, as explained in Section 4.5.2.
  Pass u to all instances of Algorithm 8 in I.
end while
return the best set among all the outputs of the instances of Algorithm 8 in I and the singleton set $\{u_m\}$.

Theorem 4.5.4. *The approximation ratio of* STREAK *is at least*

$$(1-\epsilon)\gamma \cdot \frac{3 - e^{-\gamma/2} - 2\sqrt{2 - e^{-\gamma/2}}}{2} \ .$$

The proof of Theorem 4.5.4 shows that in the final collection I there is an instance of Algorithm 8 whose τ provides a good approximation for $a\gamma \cdot f(\mathrm{OPT})$, and thus, this instance of Algorithm 8 should (up to some technical details) produce a good output set in accordance with Theorem 4.5.1.

It remains to analyze the space complexity and running time of STREAK. We concentrate on bounding the number of elements STREAK keeps in its memory at any given time, as this amount dominates the space complexity as long as we assume that the space necessary to keep an element is at least as large as the space necessary to keep each one of the numbers used by the algorithm.

Theorem 4.5.5. *The space complexity of* STREAK *is* $O(\epsilon^{-1} k \log k)$ *elements.*

The running time of Algorithm 8 is $O(Nf)$ where, abusing notation, f is the running time of a single oracle evaluation of f. Therefore, the running time of Streak is $O(Nf\epsilon^{-1}\log k)$ since it uses at every given time only $O(\epsilon^{-1}\log k)$ instances of the former algorithm. Given multiple threads, this can be improved to $O(Nf+\epsilon^{-1}\log k)$ by running the $O(\epsilon^{-1}\log k)$ instances of Algorithm 8 in parallel.

4.5.3 Improved Guarantees for Streak

Since the initial publication of the results in this chapter [116], the guarantees on both ThresholdGreedy and Streak have been improved. These tighter bounds are proved in Appendix F.

As a warm up, we provide a bound on a variant of Streak for which γ, inverse curvature $\check{\alpha}$ (defined in Appendix C), and $f(\mathrm{OPT})$ are all known in advance. This is a stronger assumption than having access to a good τ (Section 4.5.1).

Theorem 4.5.6. *Let $f(\cdot)$ have submodularity ratio γ and inverse curvature $\check{\alpha}$. If $\tau = \frac{f(\mathrm{OPT})\gamma(1-\check{\alpha})}{1+\gamma(1-\check{\alpha})}$, then ThresholdGreedy has an approximation ratio of $\frac{\gamma(1-\check{\alpha})}{1+\gamma(1-\check{\alpha})}$ irrespective of stream order.*

Remark 4.5.1. *This result is consistent with Theorem 4.4.1 which does not depend on $\check{\alpha}$, see the proof of Proposition C.2.8.*

Remark 4.5.2. *When $f(\cdot)$ is submodular, Theorem 4.5.6 matches the $1/2$ approximation ratio of [22]. However, it is not clear whether this (or any other) algorithm achieves a better approximation for arbitrary $(\gamma,\check{\alpha})$.*

The following tighter bounds are achieved by utilizing a technical lemma from [84].

Theorem 4.5.7. *In expectation,* ThresholdGreedy *has an improved approximation ratio*

$$\mathbb{E}[f(S)] \geq \frac{\tau}{a} \cdot \left[4 + \frac{\gamma}{2} - 2\sqrt{\gamma + 4}\right] \quad ,$$

where

$$a = \sqrt{\gamma'/2 + 1} - 1 = \frac{\sqrt{2\gamma' + 4} - 2}{2} = \frac{\sqrt{\gamma + 4} - 2}{2} \quad .$$

Theorem 4.5.8. *In expectation,* Streak *has an improved approximation ratio*

$$\mathbb{E}[f(S)] \geq (1 - \epsilon)\gamma \cdot \left[4 + \frac{\gamma}{2} - 2\sqrt{\gamma + 4}\right] \quad .$$

4.6 Experiments

We evaluate the performance of our streaming algorithm on two sparse feature selection applications.[2] Features are passed to all algorithms in a random order to match the setting of Section 4.5.

4.6.1 Sparse Regression with Pairwise Features

In this experiment, a sparse logistic regression is fit on 2000 training and 2000 test observations from the Phishing dataset [139]. This setup is known to be weakly submodular under mild data assumptions [73]. First, the categorical

[2]Code is available at https://github.com/eelenberg/streak.

features are one-hot encoded, increasing the feature dimension to 68. Then, all pairwise products are added for a total of $N = 4692$ features. To reduce computational cost, feature products are generated and added to the stream on-the-fly as needed. We compare with 2 other algorithms. RandomSubset selects the first k features from the random stream. LocalSearch first fills a buffer with the first k features, and then swaps each incoming feature with the feature from the buffer which yields the largest nonnegative improvement.

Figure 4.1(a) shows both the final log likelihood and the generalization accuracy for RandomSubset, LocalSearch, and our Streak algorithm for $\epsilon = \{0.75, 0.1\}$ and $k = \{20, 40, 80\}$. As expected, the RandomSubset algorithm has much larger variation since its performance depends highly on the random stream order. It also performs significantly worse than LocalSearch for both metrics, whereas Streak is comparable for most parameter choices. Figure 4.1(b) shows two measures of computational cost: running time and the number of oracle evaluations (regression fits). We note Streak scales better as k increases; for example, Streak with $k = 80$ and $\epsilon = 0.1$ ($\epsilon = 0.75$) runs in about 70% (5%) of the time it takes to run LocalSearch with $k = 40$. Interestingly, our speedups are more substantial with respect to running time. In some cases Streak actually fits more regressions than LocalSearch, but still manages to be faster. We attribute this to the fact that nearly all of LocalSearch's regressions involve k features, which are slower than many of the small regressions called by Streak.

Figure 4.2(a) shows the final log likelihood versus running time for $k = 80$ and $\epsilon \in [0.05, 0.75]$. By varying the precision ϵ, we achieve a gradual tradeoff

between speed and performance. This shows that STREAK can reduce the running time by over an order of magnitude with minimal impact on the final log likelihood.

4.6.2 Black-Box Interpretability

Our next application is interpreting the predictions of black-box machine learning models. Specifically, we begin with the Inception V3 deep neural network [23] trained on ImageNet. We use this network for the task of classifying 5 types of flowers via transfer learning. This is done by adding a final softmax layer and retraining the network.

We compare our approach to the LIME framework [24] for developing sparse, interpretable explanations. The final step of LIME is to fit a k-sparse linear regression in the space of interpretable features. Here, the features are superpixels determined by the SLIC image segmentation algorithm [140] (regions from any other segmentation would also suffice). The number of superpixels is bounded by $N = 30$. After a feature selection step, a final regression is performed on only the selected features. The following feature selection methods are supplied by LIME: *1. Highest Weights:* fits a full regression and keep the k features with largest coefficients. *2. Forward Selection:* standard greedy forward selection. *3. Lasso:* ℓ_1 regularization.

We introduce a novel method for black-box interpretability that is similar to but simpler than LIME. As before, we segment an image into N superpixels. Then, for a subset S of those regions we can create a new image that contains

only these regions and feed this into the black-box classifier. For a given model M, an input image I, and a label $\mathbf{L}_1$, we ask for an explanation: why did model M label image I with label $\mathbf{L}_1$. We propose the following solution to this problem. Consider the set function $f(\mathrm{S})$ giving the likelihood that image $I(\mathrm{S})$ has label $\mathbf{L}_1$. We approximately solve

$$\max_{|\mathrm{S}|\leq k} f(\mathrm{S}) \ ,$$

using Streak. Intuitively, we are limiting the number of superpixels to k so that the output will include only the most important superpixels, and thus, will represent an interpretable explanation. In our experiments we set $k = 5$.

Note that the set function $f(\mathrm{S})$ depends on the black-box classifier and is neither monotone nor submodular in general. Still, we find that the greedy maximization algorithm produces very good explanations for the flower classifier as shown in Figure 4.3 and the additional experiments in Appendix G. Figure 4.2(b) shows that our algorithm is much faster than the LIME approach. This is primarily because LIME relies on generating and classifying a large set of randomly perturbed example images.

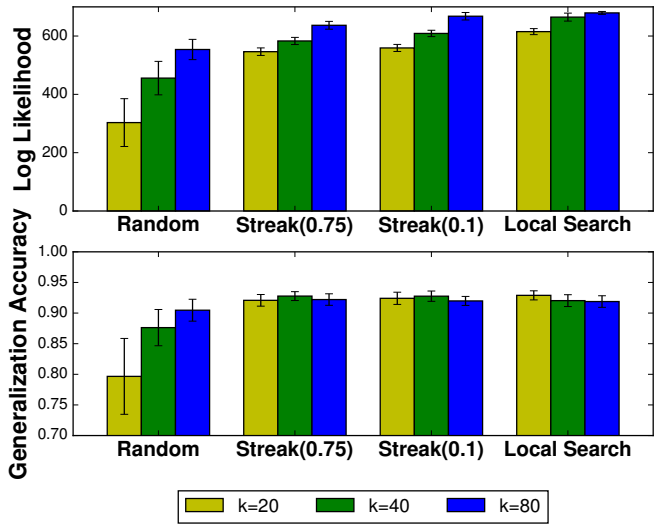
4.7 Conclusions

We propose Streak, the first streaming algorithm for maximizing weakly submodular functions, and prove that it achieves a constant factor approximation assuming a random stream order. This is useful when the set function is not submodular and, additionally, takes a long time to evaluate or has a very large ground set. Conversely, we show that under a worst case stream order no algorithm with

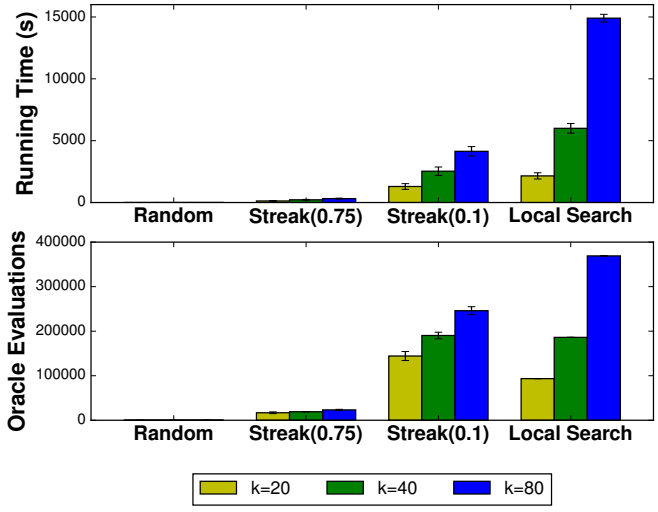
memory sublinear in the ground set size has a constant factor approximation. We formulate interpretability of black-box neural networks as set function maximization, and show that STREAK provides interpretable explanations faster than previous approaches. We also show experimentally that STREAK trades off accuracy and running time in nonlinear sparse regression.

One interesting direction for future work is to tighten the bounds of Theorems 4.5.7 and 4.5.8 further, as they are nontrivial but still somewhat loose. For example, there is a gap between the theoretical guarantee of the state-of-the-art algorithm for submodular functions and our bound for $\gamma = 1$. However, as our algorithm performs the same computation as that state-of-the-art algorithm when the function is submodular, this gap is solely an analysis issue. Hence, the real theoretical performance of our algorithm is better than what we have been able to prove in Section 4.5. Theorem 4.5.6 shows that an algorithm can do significantly better when γ and $\check{\alpha}$ known exactly. This suggests that approximate knowledge of non-submodular parameters may be useful for designing algorithms with improved guarantees.

Analogous to the results in Chapter 3, another future direction is to design neural network architectures for which the LIME or STREAK objective function satisfies a weak submodularity condition. Recently, [14] characterized *perturbation-stable* instances [141] of monotone submodular maximization problems for which greedy algorithms recover the true optimum. It is worth examining the implications of this result for machine learning problems, as well as extensions of the framework to non-submodular set functions.

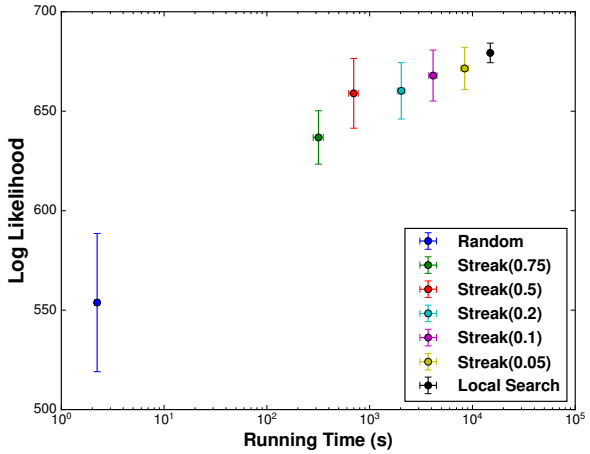


(a) Performance

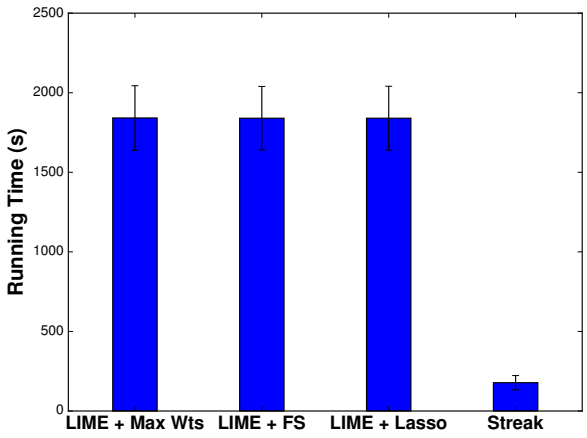


(b) Cost

Figure 4.1: Logistic Regression, Phishing dataset with pairwise feature products. Our algorithm is comparable to LocalSearch in both log likelihood and generalization accuracy, with much lower running time and number of model fits in most cases. Results averaged over 40 iterations, error bars show 1 standard deviation.



(a) Sparse Regression



(b) Interpretability

Figure 4.2: 4.2(a): Logistic Regression, Phishing dataset with pairwise feature products, $k = 80$ features. By varying the parameter ϵ, our algorithm captures a time-accuracy tradeoff between RandomSubset and LocalSearch. Results averaged over 40 iterations, standard deviation shown with error bars. 4.2(b): Running times of interpretability algorithms on the Inception V3 network, $N = 30$, $k = 5$. Streaming maximization runs 10 times faster than the LIME framework. Results averaged over 40 total iterations using 8 example explanations, error bars show 1 standard deviation.

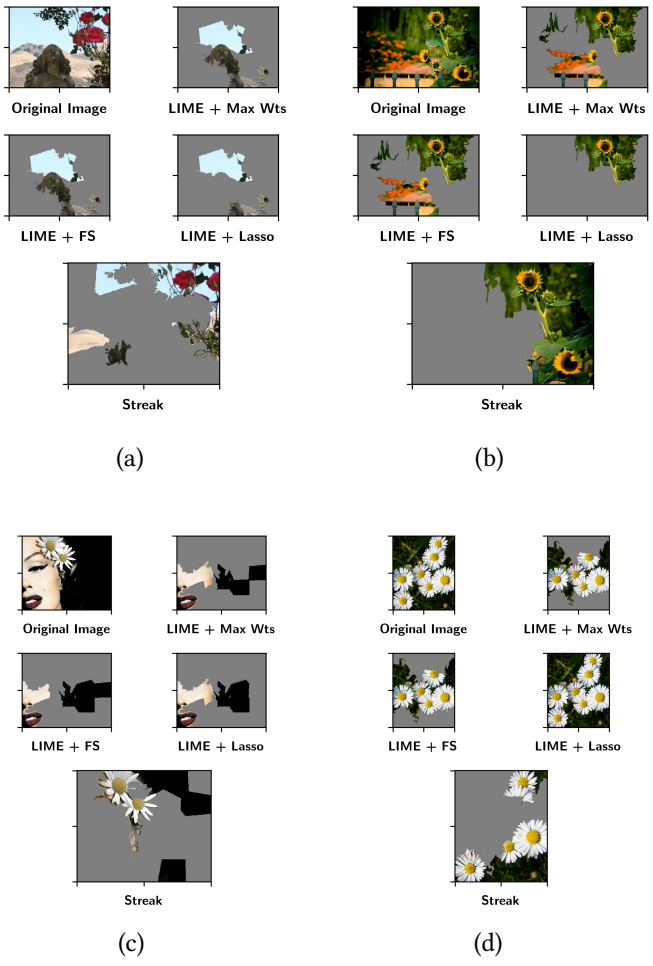


Figure 4.3: Comparison of interpretability algorithms for the Inception V3 deep neural network. We have used transfer learning to extract features from Inception and train a flower classifier. In these four input images the flower types were correctly classified (from (a) to (d): rose, sunflower, daisy, and daisy). We ask the question of interpretability: *why* did this model classify this image as rose. We are using our framework (and the recent prior work LIME [24]) to see which parts of the image the neural network is looking at for these classification tasks. As can be seen STREAK correctly identifies the flower parts of the images while some LIME variations do not. More importantly, STREAK is creating subsampled images on-the-fly, and hence, runs approximately 10 times faster. Since interpretability tasks perform multiple calls to the black-box model, the running times can be quite significant.

Appendices

Appendix A

Proofs for Chapter 2

A.1 Proof of Theorem 2.3.2

Let[1] m be the total number of edges in the original graph G. If e is an edge in the original graph G, let t_e be the random indicator after sampling. $t_e = 1$ if e is sampled and 0 otherwise. Let $\mathcal{H}_0, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3$ denote the set of distinct subgraphs of the kind H_0, H_1, H_2, and H_3 (anti-clique, edge, wedge and triangle) respectively. Let $A, \_(e), \Lambda(e,f)$, and $\Delta(e,f,g)$ denote an anti-clique with no edges, a H_1 with edge e , a H_2 with two edges e, f, and a triangle with edges e, f, g respectively in the original graph G. Our estimators (2.13) are each a function of Y_i's and each Y_i can be written as a polynomial of at most degree 3 in all the variables t_e.

$$Y_0 = n_0 + \sum_{\_(e)\in\mathcal{H}_1} (1-t_e) + \sum_{\Lambda(e,f)\in\mathcal{H}_2} (1-t_e)(1-t_f) + \sum_{\Delta(e,f,g)\in\mathcal{H}_3} (1-t_e)(1-t_f)(1-t_g), \tag{A.1}$$

[1] The material in this appendix is based on the following conference papers: [25] E. R. Elenberg, K. Shanmugam, M. Borokhovich, and A. G. Dimakis. Beyond Triangles: A Distributed Framework for Estimating 3-profiles of Large Graphs. In *KDD*, pages 229–238, 2015. [26] E. R. Elenberg, K. Shanmugam, M. Borokhovich, and A. G. Dimakis. Distributed Estimation of Graph 4-profiles. In *WWW*, pages 483–493, 2016. The dissertation author's primary contribution is applying Kim-Vu concentration inequalities to graph profile sparsifiers. The dissertation author also assisted with other contributions and is the primary contributor of these papers.

$$
\begin{aligned}
Y_1 = & \sum_{\_(e)\in\mathcal{H}_1} t_e + \sum_{\Lambda(e,f)\in\mathcal{H}_2} ((1-t_e)t_f + (1-t_f)t_e) + \\
& \sum_{\Delta(e,f,g)\in\mathcal{H}_3} t_e(1-t_f)(1-t_g) + \\
& \sum_{\Delta(e,f,g)\in\mathcal{H}_3} t_f(1-t_e)(1-t_g) + t_g(1-t_e)(1-t_f),
\end{aligned} \tag{A.2}
$$

$$
\begin{aligned}
Y_2 = & \sum_{\Lambda(e,f)\in\mathcal{H}_2} t_e t_f + \\
& \sum_{\Delta(e,f,g)\in\mathcal{H}_3} (t_e t_f)(1-t_g) + t_f t_g(1-t_e) + t_e(1-t_f)t_g),
\end{aligned} \tag{A.3}
$$

$$
Y_3 = \sum_{\Delta(e,f,g)\in\mathcal{H}_3} t_e t_f t_g, \tag{A.4}
$$

$$
S_1 = \sum_{\_(e)\in\mathcal{H}_1} t_e, \tag{A.5}
$$

$$
D_1 = \sum_{\Lambda(e,f)\in\mathcal{H}_2} (t_e + t_f), \tag{A.6}
$$

$$
D_2 = \sum_{\Lambda(e,f)\in\mathcal{H}_2} t_e t_f, \tag{A.7}
$$

$$
T_1 = \sum_{\Delta(e,f,g)\in\mathcal{H}_3} (t_e + t_f + t_g), \tag{A.8}
$$

$$
T_2 = \sum_{\Delta(e,f,g)\in\mathcal{H}_3} (t_e t_f + t_f t_g + t_g t_e), \tag{A.9}
$$

$$
Y_1 = S_1 + D_1 - 2D_2 + T_1 - 2T_2 + 3Y_3, \tag{A.10}
$$

$$
Y_2 = D_2 + T_2 - 3Y_3. \tag{A.11}
$$

Note that the newly defined polynomials have the following expectations:

$$
\mathbb{E}[S_1] = pn_1, \qquad \mathbb{E}[D_1] = 2pn_2,
$$

$$
\mathbb{E}[D_2] = p^2 n_2, \qquad \mathbb{E}[T_1] = 3pn_3, \qquad \mathbb{E}[T_2] = 3p^2 n_3.
$$

We observe that in the above even by change of variables $y_e = (1 - t_e)$, Y_1

and Y_2 are not totally positive polynomials. This means that Theorem 2.3.1 cannot be applied directly to the Y_i's or X_i's. The strategy we adopt is to split Y_1 and Y_2 into many polynomials, each of which is totally positive, and then apply Theorem 2.3.1 on each of them. $P = \{Y_0, Y_3, S_1, D_1, D_2, T_1, T_2\}$ form the set of totally positive polynomials (proved below). Substituting the above equations into (2.14), we have the following system of equations that connect X_i's and the set of totally positive polynomials P:

$$\begin{aligned} X_0 &= Y_0 - \frac{1-p}{p}(S_1 + D_1 + T_1 - 2D_2 - 2T_2 + 3Y_3) \\ &\quad + \frac{(1-p)^2}{p^2}(D_2 + T_2 - 3Y_3) - \frac{(1-p)^3}{p^3}Y_3 \\ &= Y_0 - \frac{1-p}{p}(S_1 + D_1 + T_1) \\ &\quad + \frac{1-p^2}{p^2}(D_2 + T_2) - \frac{1-p^3}{p^3}Y_3. \end{aligned} \tag{A.12}$$

$$\begin{aligned} X_1 &= \frac{1}{p}(S_1 + D_1 + T_1 - 2D_2 - 2T_2 + 3Y_3) \\ &\quad - \frac{2(1-p)}{p^2}(D_2 + T_2 - 3Y_3) + \frac{3(1-p)^2}{p^3}Y_3 \\ &= \frac{1}{p}(S_1 + D_1 + T_1) - \frac{2}{p^2}(D_2 + T_2) + \frac{3}{p^3}Y_3. \end{aligned} \tag{A.13}$$

$$\begin{aligned} X_2 &= \frac{1}{p^2}(D_2 + T_2 - 3Y_3) - \frac{3(1-p)}{p^3}Y_3 \\ &= \frac{1}{p^2}(D_2 + T_2) - \frac{3}{p^3}Y_3. \end{aligned} \tag{A.14}$$

$$X_3 = \frac{1}{p^3}Y_3. \tag{A.15}$$

Let α_e, β_e, and Δ_e be the maximum number of H_1's, H_2's, and H_3's containing an edge e in the original graph G. Let α, β and Δ be the maximum of α_e, β_e, and

Δ_e over all edges e. We now show concentration results for the totally positive polynomials alone.

Lemma A.1.1. *Define variables $y_e = 1 - t_e$. Then Y_0 is totally positive in y_e. With respect to the variables y_e, $\mathbb{E}_{\geq 1}[Y_0] \leq 3\max\{\alpha, \beta, \Delta\}$.*

Proof. We have the expectation of the following partial derivatives, up to the third order:

$$\begin{aligned}\mathbb{E}\left[\frac{\partial Y_0}{\partial y_e}\right] &= \alpha_e + (1-p)\beta_e + (1-p)^2\Delta_e \\ &\leq 3\max\{\alpha_e, \beta_e, \Delta_e\}. \\ \mathbb{E}\left[\frac{\partial Y_0}{\partial y_e y_f}\right] \leq 1 + (1-p) \leq 2,\ \mathbb{E}\left[\frac{\partial Y_0}{\partial y_e y_f y_g}\right] &\leq 1.\end{aligned}$$

From the above equations, we have $\mathbb{E}_{\geq 1}[Y_1] \leq 3\max\{\alpha, \beta, \Delta\}$ for a nonempty graph. □

To satisfy $\mathbb{E}_{\geq 1}[Y_0] \leq \mathbb{E}[Y_0]$, it is sufficient to have

$$n_0 \geq 3\max\{\alpha, \beta, \Delta\}. \tag{A.16}$$

This is because $Y_0 \geq n_0$ with probability 1.

Lemma A.1.2. *Y_3 is totally positive in t_e. W.r.t the variables t_e, $\mathbb{E}_{\geq 1}[Y_3] \leq \max\{1, p^2\Delta\}$.*

Proof. We have the expectation of the following partial derivatives, up to the third order:

$$\mathbb{E}\left[\frac{\partial Y_3}{\partial t_e}\right] = p^2\Delta_e,\ \mathbb{E}\left[\frac{\partial Y_3}{\partial t_e t_f}\right] = p \leq 1,\ \mathbb{E}\left[\frac{\partial Y_3}{\partial t_e t_f t_g}\right] \leq 1.$$

From the above equations, we have $\mathbb{E}_{\geq 1}[Y_3] \leq \max\{1, p^2\Delta\}$. □

$\mathbb{E}_{\geq 1}[Y_3] \leq \mathbb{E}[Y_3]$ implies

$$p \geq \max\{\frac{1}{\sqrt[3]{n_3}}, \Delta/n_3\}. \tag{A.17}$$

Lemma A.1.3. *S_1 is totally positive in t_e. W.r.t. the variables t_e, $\mathbb{E}_{\geq 1}[S_1] \leq \alpha$.*

Proof. We have the expectation of the following partial derivatives, up to the second order:

$$\mathbb{E}\left[\frac{\partial S_1}{\partial t_e}\right] = \alpha_e,\ \mathbb{E}\left[\frac{\partial S_1}{\partial t_e t_f}\right] = 0.$$

From the above equations, we have $\mathbb{E}_{\geq 1}[S_1] \leq \alpha$. □

$\mathbb{E}_{\geq 1}[S_1] \leq \mathbb{E}[S_3]$ implies

$$p \geq \alpha/n_1. \tag{A.18}$$

Lemma A.1.4. *D_1 is totally positive in t_e. W.r.t the variables t_e, $\mathbb{E}_{\geq 1}[D_1] \leq \beta$.*

Proof. We have the expectation of the following partial derivatives, up to the second order:

$$\mathbb{E}\left[\frac{\partial D_1}{\partial t_e}\right] = \beta_e,\ \mathbb{E}\left[\frac{\partial D_1}{\partial t_e t_f}\right] = 0.$$

From the above equations, we have $\mathbb{E}_{\geq 1}[D_1] \leq \beta$. □

$\mathbb{E}_{\geq 1}[D_1] \leq \mathbb{E}[D_1]$ implies

$$p \geq \beta/(2n_2). \tag{A.19}$$

Lemma A.1.5. *T_1 is totally positive in t_e. W.r.t. the variables t_e, $\mathbb{E}_{\geq 1}[T_1] \leq \Delta$.*

Proof. We have the expectation of the following partial derivatives, up to the second order:

$$\mathbb{E}\left[\frac{\partial T_1}{\partial t_e}\right] = \Delta_e,\ \mathbb{E}\left[\frac{\partial T_1}{\partial t_e t_f}\right] = 0.$$

From the above equations, we have $\mathbb{E}_{\geq 1}[T_1] \leq \Delta$. □

$\mathbb{E}_{\geq 1}[T_1] \leq \mathbb{E}[T_1]$ implies

$$p \geq \Delta/(3n_3). \tag{A.20}$$

Lemma A.1.6. *D_2 is totally positive in t_e. W.r.t. the variables t_e, $\mathbb{E}_{\geq 1}[D_2] \leq \max\{p\beta, 1\}$.*

Proof. We have the expectation of the following partial derivatives, up to the second order:

$$\mathbb{E}\left[\frac{\partial D_2}{\partial t_e}\right] = p\beta_e,\ \mathbb{E}\left[\frac{\partial D_2}{\partial t_e t_f}\right] \leq 1.$$

From the above equations, we have $\mathbb{E}_{\geq 1}[D_2] \leq \max\{p\beta, 1\}$. □

$\mathbb{E}_{\geq 1}[D_2] \leq \mathbb{E}[D_2]$ implies

$$p \geq \max\{\beta/n_2, \frac{1}{\sqrt{n_2}}\}. \tag{A.21}$$

Lemma A.1.7. *T_2 is totally positive in t_e. W.r.t the variables t_e, $\mathbb{E}_{\geq 1}[T_2] \leq \max\{2p\Delta, 1\}$.*

Proof. We have the expectation of the following partial derivatives, up to the second order:

$$\mathbb{E}\left[\frac{\partial T_2}{\partial t_e}\right] = 2p\Delta_e,\ \mathbb{E}\left[\frac{\partial T_2}{\partial t_e t_f}\right] \leq 1.$$

From the above equations, we have $\mathbb{E}_{\geq 1}[T_2] \leq \max\{2p\Delta, 1\}$. □

$\mathbb{E}_{\geq 1}[T_2] \leq \mathbb{E}[T_2]$ implies

$$p \geq \max\{2\Delta/(3n_3), \frac{1}{\sqrt{3n_3}}\}. \tag{A.22}$$

Now merging all the conditions (A.16)-(A.22), we get

$$n_0 \geq 3\max\{\alpha, \beta, \Delta\},\ p \geq \max\left\{\frac{1}{\sqrt[3]{n_3}}, \frac{1}{\sqrt{n_2}}, \frac{\Delta}{n_3}, \frac{\beta}{n_2}, \frac{\alpha}{n_1}\right\}. \tag{A.23}$$

Applying Theorem 2.3.1 to all the totally positive polynomials, along with

(A.23), we get

$$
\begin{aligned}
&\mathbb{P}\left(|Y_0 - \mathbb{E}[Y_0]| > a_3\sqrt{\mathbb{E}[Y_0]\mathbb{E}_{\geq 1}[Y_0]}\lambda_1^3\right) \\
&\qquad = O\left(\exp\left(-\lambda_1 + (2)\log m\right)\right), \\
&\mathbb{P}\left(|Y_3 - \mathbb{E}[Y_3]| > a_3\sqrt{\mathbb{E}[Y_3]\mathbb{E}_{\geq 1}[Y_3]}\lambda_2^3\right) \\
&\qquad = O\left(\exp\left(-\lambda_2 + (2)\log m\right)\right), \\
&\mathbb{P}\left(|S_1 - \mathbb{E}[S_1]| > a_1\sqrt{\mathbb{E}[S_1]\mathbb{E}_{\geq 1}[S_1]}\lambda_3\right) \\
&\qquad = O\left(\exp\left(-\lambda_3\right)\right), \\
&\mathbb{P}\left(|D_1 - \mathbb{E}[D_1]| > a_1\sqrt{\mathbb{E}[D_1]\mathbb{E}_{\geq 1}[D_1]}\lambda_4\right) \\
&\qquad = O\left(\exp\left(-\lambda_4\right)\right), \\
&\mathbb{P}\left(|T_1 - \mathbb{E}[T_1]| > a_1\sqrt{\mathbb{E}[T_1]\mathbb{E}_{\geq 1}[T_1]}\lambda_5\right) \\
&\qquad = O\left(\exp\left(-\lambda_5\right)\right), \\
&\mathbb{P}\left(|D_2 - \mathbb{E}[D_2]| > a_2\sqrt{\mathbb{E}[D_2]\mathbb{E}_{\geq 1}[D_2]}\lambda_6^2\right) \\
&\qquad = O\left(\exp\left(-\lambda_6 + \log m\right)\right), \\
&\mathbb{P}\left(|T_2 - \mathbb{E}[T_2]| > a_2\sqrt{\mathbb{E}[T_2]\mathbb{E}_{\geq 1}[T_2]}\lambda_7^2\right) \\
&\qquad = O\left(\exp\left(-\lambda_7 + \log m\right)\right).
\end{aligned}
\tag{A.24}
$$

Choose an $\epsilon > 0$. We force the following conditions:

$$\begin{aligned}
a_3\sqrt{\mathbb{E}[Y_0]\mathbb{E}_{\geq 1}[Y_0]}\lambda_1^3 &= \epsilon\mathbb{E}[Y_0],\\
a_3\sqrt{\mathbb{E}[Y_3]\mathbb{E}_{\geq 1}[Y_3]}\lambda_2^3 &= \epsilon\mathbb{E}[Y_3],\\
a_1\sqrt{\mathbb{E}[S_1]\mathbb{E}_{\geq 1}[S_1]}\lambda_3 &= \epsilon\mathbb{E}[S_1],\\
a_1\sqrt{\mathbb{E}[D_1]\mathbb{E}_{\geq 1}[D_1]}\lambda_4 &= \epsilon\mathbb{E}[D_1],\\
a_1\sqrt{\mathbb{E}[T_1]\mathbb{E}_{\geq 1}[T_1]}\lambda_5 &= \epsilon\mathbb{E}[T_1],\\
a_2\sqrt{\mathbb{E}[D_2]\mathbb{E}_{\geq 1}[D_2]}\lambda_6^2 &= \epsilon\mathbb{E}[D_2],\\
a_2\sqrt{\mathbb{E}[T_2]\mathbb{E}_{\geq 1}[T_2]}\lambda_7^2 &= \epsilon\mathbb{E}[T_2].
\end{aligned} \tag{A.25}$$

Let $\gamma > 0$. In order for the right hand side of every equation in (A.24) to be $O(\exp(-\gamma \log m))$, assuming all the bounds in Lemmas A.1.1-A.1.7, it is sufficient to have

$$\begin{aligned}
\frac{n_0}{3\max\{\alpha,\beta,\Delta\}} &\geq \frac{a_3^2\log^6\left(m^{2+\gamma}\right)}{\epsilon^2},\\
\frac{p}{\max\{\frac{1}{\sqrt[3]{n_3}},\Delta/n_3\}} &\geq \frac{a_3^2\log^6\left(m^{2+\gamma}\right)}{\epsilon^2},\\
\frac{p}{\max\{\frac{\alpha}{n_1},\frac{\beta}{2n_2},\frac{\Delta}{3n_3}\}} &\geq \frac{a_1^2\log^2\left(m^{\gamma}\right)}{\epsilon^2},\\
\frac{p}{\max\{\frac{\beta}{n_2},\frac{2\Delta}{3n_3},\frac{1}{\sqrt{n_2}},\frac{1}{\sqrt{n_3}}\}} &\geq \frac{a_2^2\log^4\left(m^{1+\gamma}\right)}{\epsilon^2}.
\end{aligned} \tag{A.26}$$

We can see that the conditions in (A.26) imply the conditions in (A.23).

These can be simplified to remove some redundancy as follows:

$$\begin{aligned}
\frac{n_0}{3\max\{\alpha,\beta,\Delta\}} &\geq \frac{a_3^2\log^6\left(m^{2+\gamma}\right)}{\epsilon^2},\\
\frac{p}{\max\{\frac{1}{\sqrt[3]{n_3}},\Delta/n_3\}} &\geq \frac{a_3^2\log^6\left(m^{2+\gamma}\right)}{\epsilon^2},\\
\frac{p}{\alpha/n_1} &\geq \frac{a_1^2\log^2\left(m^{\gamma}\right)}{\epsilon^2},\\
\frac{p}{\max\{\frac{\beta}{n_2},\frac{1}{\sqrt{n_2}}\}} &\geq \frac{a_2^2\log^4\left(m^{1+\gamma}\right)}{\epsilon^2}.
\end{aligned} \tag{A.27}$$

This is due to the fact that $a_3 \geq a_2 \geq a_1$ and $m^3 \geq m^2$. Therefore, subject to (A.27), all totally positive polynomials $Y_0, Y_3, D_1, D_2, S_1, T_1, T_2$ concentrate within a multiplicative factor of $(1 \pm \epsilon)$ with probability at least $1 - O\left(\frac{1}{m^\gamma}\right)$.

Under the above concentration result, let the deviations of X_i's be denoted by δX_i. Now we calculate the deviation of X_0 using (A.12).

$$\begin{aligned}
\delta X_0 &\leq \epsilon\mathbb{E}[Y_0] + \epsilon\frac{1-p}{p}\left(|\mathbb{E}[S_1]| + |\mathbb{E}[D_1]| + |\mathbb{E}[T_1]|\right)\\
&\quad + \epsilon\frac{1-p^2}{p^2}\left(|\mathbb{E}[D_2]| + |\mathbb{E}[T_2]|\right) - \epsilon\frac{1-p^3}{p^3}|\mathbb{E}[Y_3]|\\
&\leq \epsilon(n_0 + n_1 + 3n_2 + 7n_3)\\
&\leq 7\epsilon(n_0 + n_1 + n_2 + n_3).
\end{aligned}$$

Similarly for other X_i's, we get

$$\begin{aligned}
\delta X_1 &\leq 12\epsilon\left(n_1 + n_2 + n_3\right),\\
\delta X_2 &\leq 6\epsilon\left(n_2 + n_3\right),\\
\delta X_3 &\leq \epsilon n_3.
\end{aligned}$$

Therefore, sampling every edge independently with probability p satisfying all conditions in (A.27), all $X_i's$ concentrate within an additive gap of $(1 \pm 12\epsilon)\binom{|V|}{3}$ with probability at least $1 - \frac{1}{m^\gamma}$. The constants in this proof can be tightened by a more accurate analysis.

A.2 Proof of Lemma 2.3.2

This proof is a straightforward application of Theorem 2.3.1, which is the main result of [35]. Let $Y_{10} = \sum_{\boxtimes(a,b,c,d)\in\mathcal{H}_{10}} t_{ab}t_{bc}t_{cd}t_{da}t_{ac}t_{bd}$. Clearly Y_{10} is totally positive. Let $k_{10,ab}$, σ_{abc}, and ν_{abc} be the maximum number of 4-cliques sharing a common edge t_{ab}, wedge Λ_{abc}, and triangle Δ_{abc}, respectively. Taking repeated partial derivatives,

$$\begin{aligned}
\mathbb{E}\left[\frac{\partial Y_{10}}{\partial t_{ab}}\right] &= p^5 k_{10,ab}, \\
\mathbb{E}\left[\frac{\partial Y_{10}}{\partial t_{ab}t_{bc}}\right] = p^4\sigma_{abc}, \quad \mathbb{E}\left[\frac{\partial Y_{10}}{\partial t_{ab}t_{cd}}\right] &= p^4, \\
\mathbb{E}\left[\frac{\partial Y_{10}}{\partial t_{ab}t_{bc}t_{ac}}\right] = p^3\nu_{abc}, \quad \mathbb{E}\left[\frac{\partial Y_{10}}{\partial t_{ab}t_{bc}t_{cd}}\right] &= p^3, \\
\mathbb{E}\left[\frac{\partial Y_{10}}{\partial t_{ab}t_{bc}t_{ac}t_{da}}\right] = \mathbb{E}\left[\frac{\partial Y_{10}}{\partial t_{ab}t_{bc}t_{cd}t_{da}}\right] &= p^2, \\
\mathbb{E}\left[\frac{\partial Y_{10}}{\partial t_{ab}t_{bc}t_{cd}t_{da}t_{ac}}\right] = p, \quad \mathbb{E}\left[\frac{\partial Y_{10}}{\partial t_{ab}t_{bc}t_{cd}t_{da}t_{ac}t_{bd}}\right] &= 1.
\end{aligned}$$

Noting that $\sigma_{abc} \le \min\{k_{10,ab}, k_{10,bc}\} \le k_{10}$, similarly $\nu_{abc} \le k_{10}$, and $p^5 \le p^4 \ldots \le 1$, we have $\mathbb{E}_{\ge 1}[Y_1] \le \max\{1, p^3 k_{10}\}$. $\mathbb{E}_{\ge 1}[Y_{10}] \le \mathbb{E}[Y_{10}] = p^6 N_{10}$ implies

$$p \ge \max\{\sqrt[6]{1/N_{10}}, \sqrt[3]{k_{10}/N_{10}}\}. \tag{A.28}$$

Choose $\epsilon_{KV} \geq 0$ and let $\epsilon_{KV}\mathbb{E}[Y_{10}] = a_6\sqrt{\mathbb{E}[Y_{10}]\mathbb{E}_{\geq 1}[Y_{10}]}\lambda^6$. Applying Theorem 2.3.1 to Y_{10} given (2.18) and (A.28), the right hand side of (2.16) becomes $O(\exp(-\gamma \log m)) = O(1/m^\gamma)$. Therefore, the error of the 4-clique estimator X_{10} is

$$\delta X_{10} = \frac{1}{p^6}\delta Y_{10} = \frac{1}{p^6}(\epsilon_{KV}p^6 N_{10}) = \epsilon N_{10}$$

with probability greater than $1 - \frac{1}{m^\gamma}$.

Appendix B

4-profile Sparsifier Details

Another advantage[1] to read-k function families is that they are simpler to extend to more complex subgraphs. We now state concentration results for the full 4-profile sparsifier evaluated experimentally in Section 2.4. The edge sampling matrix $\mathbf{H}$ is defined by the relations

$$\begin{bmatrix} \mathbb{E}[Y_0] \\ \vdots \\ \mathbb{E}[Y_{10}] \end{bmatrix} = \mathbf{H} \begin{bmatrix} N_0 \\ \vdots \\ N_{10} \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} X_0 \\ \vdots \\ X_{10} \end{bmatrix} = \mathbf{H}^{-1} \begin{bmatrix} Y_0 \\ \vdots \\ Y_{10} \end{bmatrix}, \qquad \text{where}$$

$$\mathbf{H} = \begin{bmatrix}
1 & 1-p & (1-p)^2 & (1-p)^2 & (1-p)^3 & (1-p)^3 & (1-p)^3 & (1-p)^4 & (1-p)^4 & (1-p)^5 & (1-p)^6 \\
0 & p & 2p(1-p) & 2p(1-p) & 3p(1-p)^2 & 3p(1-p)^2 & 3p(1-p)^2 & 4p(1-p)^3 & 4p(1-p)^3 & 5p(1-p)^4 & 6p(1-p)^5 \\
0 & 0 & p^2 & 0 & p^2(1-p) & 0 & 0 & 2p^2(1-p)^2 & p^2(1-p)^2 & 2p^2(1-p)^3 & 3p^2(1-p)^4 \\
0 & 0 & 0 & p^2 & 2p^2(1-p) & 3p^2(1-p) & 3p^2(1-p) & 4p^2(1-p)^2 & 5p^2(1-p)^2 & 8p^2(1-p)^3 & 12p^2(1-p)^4 \\
0 & 0 & 0 & 0 & p^3 & 0 & 0 & 4p^3(1-p) & 2p^3(1-p) & 6p^3(1-p)^2 & 12p^3(1-p)^3 \\
0 & 0 & 0 & 0 & 0 & p^3 & 0 & 0 & p^3(1-p) & 2p^3(1-p)^2 & 4p^3(1-p)^3 \\
0 & 0 & 0 & 0 & 0 & 0 & p^3 & 0 & p^3(1-p) & 2p^3(1-p)^2 & 4p^3(1-p)^3 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & p^4 & 0 & p^4(1-p) & 3p^4(1-p)^2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p^4 & 4p^4(1-p) & 12p^4(1-p)^2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p^5 & 6p^5(1-p) \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p^6
\end{bmatrix}.$$

[1] The material in this appendix is based on the following conference paper: [26] E. R. Elenberg, K. Shanmugam, M. Borokhovich, and A. G. Dimakis. Distributed Estimation of Graph 4-profiles. In *WWW*, pages 483–493, 2016. It is the dissertation author's contribution.

Let $t = \frac{p-1}{p}$. Then the inverse sampling matrix is given by

$$\mathbf{H}^{-1} = \begin{bmatrix}
1 & t & t^2 & t^2 & t^3 & t^3 & t^3 & t^4 & t^4 & t^5 & t^6 \\
0 & \frac{1}{p} & \frac{2t}{p} & \frac{2t}{p} & \frac{3t^2}{p} & \frac{3t^2}{p} & \frac{3t^2}{p} & \frac{4t^3}{p} & \frac{4t^3}{p} & \frac{5t^4}{p} & \frac{6t^5}{p} \\
0 & 0 & \frac{1}{p^2} & 0 & \frac{t}{p^2} & 0 & 0 & \frac{2t^2}{p^2} & \frac{t^2}{p^2} & \frac{2t^3}{p^2} & \frac{3t^4}{p^2} \\
0 & 0 & 0 & \frac{1}{p^2} & \frac{2t}{p^2} & \frac{3t}{p^2} & \frac{3t}{p^2} & \frac{4t^2}{p^2} & \frac{5t^2}{p^2} & \frac{8t^3}{p^2} & \frac{12t^4}{p^2} \\
0 & 0 & 0 & 0 & \frac{1}{p^3} & 0 & 0 & \frac{4t}{p^3} & \frac{2t}{p^3} & \frac{6t^2}{p^3} & \frac{12t^3}{p^3} \\
0 & 0 & 0 & 0 & 0 & \frac{1}{p^3} & 0 & 0 & \frac{t}{p^3} & \frac{2t^2}{p^3} & \frac{4t^3}{p^3} \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{p^3} & 0 & \frac{t}{p^3} & \frac{2t^2}{p^3} & \frac{4t^3}{p^3} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{p^4} & 0 & \frac{t}{p^4} & \frac{3t^2}{p^4} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{p^4} & \frac{4t}{p^4} & \frac{12t^2}{p^4} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{p^5} & \frac{6t}{p^5} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{p^6}
\end{bmatrix}. \tag{B.1}$$

The binomial coefficients in these matrices influence our concentration bounds, which we now state:

Theorem B.0.1 (4-profile sparsifier estimators)**.** *Consider the sampling process described above and in Section 2.3. Let X_i, $0 \le i \le 10$ (and $\mathbf{X}$ be a vector of these estimates), be the actual estimates of 4-profiles. Let k_i be the maximum number of subgraphs F_i sharing a common edge. Let Y_i, $0 \le i \le 10$, be the 4 profile counts of the sparsified graph. Then let N_i, $0 \le i \le 10$, be the actual counts. Choose $0 < \delta < 1$ and $\epsilon > 0$. Let $C = (192)^2/2$. If*

$$p \ge \left(\frac{C\log(2/\delta)k_{10}}{\epsilon^2 N_{10}}\right)^{1/12}, \quad p \ge \left(\frac{C\log(2/\delta)(k_9 + 6k_{10})}{\epsilon^2(N_9 + 6N_{10})}\right)^{1/10},$$
$$p \ge \left(\frac{C\log(2/\delta)(k_8 + 4k_9 + 12k_{10})}{\epsilon^2(N_8 + 4N_9 + 12N_{10})}\right)^{1/8},$$
$$p \ge \left(\frac{C\log(2/\delta)(k_7 + k_9 + 3k_{10})}{\epsilon^2(N_7 + N_9 + 3N_{10})}\right)^{1/8}, \quad p \ge \left(\frac{C\log(2/\delta)(k_6 + k_8 + 2k_9 + 4k_{10})}{\epsilon^2(N_6 + N_8 + 2N_9 + 4N_{10})}\right)^{1/6},$$

$$p \geq \left(\frac{C \log(2/\delta)(k_5 + k_8 + 2k_9 + 4k_{10})}{\epsilon^2 (N_5 + N_8 + 2N_9 + 4N_{10})} \right)^{1/6},$$
$$p \geq \left(\frac{C \log(2/\delta)(k_4 + 4k_7 + 2k_8 + 6k_9 + 12k_{10})}{\epsilon^2 (N_4 + 4N_7 + 2N_8 + 6N_9 + 12N_{10})} \right)^{1/6},$$
$$p \geq \left(\frac{C \log(2/\delta)(k_3 + 2k_4 + 3k_5 + 3k_6 + 4k_7 + 5k_8 + 8k_9 + 12k_{10})}{\epsilon^2 (N_3 + 2N_4 + 3N_5 + 3N_6 + 4N_7 + 5N_8 + 8N_9 + 12N_{10})} \right)^{1/4},$$
$$p \geq \left(\frac{C \log(2/\delta)(k_2 + k_4 + 2k_7 + k_8 + 2k_9 + 3k_{10})}{\epsilon^2 (N_2 + N_4 + 2N_7 + N_8 + 2N_9 + 3N_{10})} \right)^{1/4},$$
$$p \geq \left(\frac{C \log(2/\delta)(k_1 + 2k_2 + 2k_3 + 3k_4 + 3k_5 + 3k_6 + 4k_7 + 4k_8 + 5k_9 + 6k_{10})}{\epsilon^2 (N_1 + 2N_2 + 2N_3 + 3N_4 + 3N_5 + 3N_6 + 4N_7 + 4N_8 + 5N_9 + 6N_{10})} \right)^{1/2},$$
$$n_0 \leq |V|^2 \left(|V|^2 - \frac{C \log(2/\delta)}{\epsilon^2} \right),$$

then $\|\delta \mathbf{X}\|_\infty \leq \epsilon \binom{|V|}{4}$ *with probability at least* $1 - \delta$.

Proof. We apply Proposition 1 a total of 11 times to the sampling-estimator system defined above by $\mathbf{H}$ and $\mathbf{H}^{-1}$. In our context, each sampled subgraph count Y_i is a sum of functions in a read-k_{Y_i} family, where $k_{Y_i} \leq \min\{|V| - 2, N_i\}$. Let $k_{i,e}$ be the maximum number of subgraphs F_i sharing a common edge e, and let $k_i = \max_e k_{i,e}$,

for $i = 0, \ldots, 10$. The Y_i's have the following parameters:

$$
\begin{aligned}
r_{Y_0} &= \binom{|V|}{4}, \quad k_{Y_0} = |V|, \\
r_{Y_1} &= N_1 + 2N_2 + 2N_3 + 3N_4 + 3N_5 + 3N_6 + 4N_7 + 4N_8 + 5N_9 + 6N_{10}, \\
k_{Y_1} &= k_1 + 2k_2 + 2k_3 + 3k_4 + 3k_5 + 3k_6 + 4k_7 + 4k_8 + 5k_9 + 6k_{10}, \\
r_{Y_2} &= N_2 + N_4 + 2N_7 + N_8 + 2N_9 + 3N_{10}, \\
k_{Y_2} &= k_2 + k_4 + 2k_7 + k_8 + 2k_9 + 3k_{10}, \\
r_{Y_3} &= N_3 + 2N_4 + 3N_5 + 3N_6 + 4N_7 + 5N_8 + 8N_9 + 12N_{10}, \\
k_{Y_3} &= k_3 + 2k_4 + 3k_5 + 3k_6 + 4k_7 + 5k_8 + 8k_9 + 12k_{10}, \\
r_{Y_4} &= N_4 + 4N_7 + 2N_8 + 6N_9 + 12N_{10}, \quad k_{Y_4} = k_4 + 4k_7 + 2k_8 + 6k_9 + 12k_{10}, \\
r_{Y_5} &= N_5 + N_8 + 2N_9 + 4N_{10}, \quad k_{Y_5} = k_5 + k_8 + 2k_9 + 4k_{10}, \\
r_{Y_6} &= N_6 + N_8 + 2N_9 + 4N_{10}, \quad k_{Y_6} = k_6 + k_8 + 2k_9 + 4k_{10}, \\
r_{Y_7} &= N_7 + N_9 + 3N_{10}, \quad k_{Y_7} = k_7 + k_9 + 3k_{10}, \\
r_{Y_8} &= N_8 + 4N_9 + 12N_{10}, \quad k_{Y_8} = k_8 + 4k_9 + 12k_{10}, \\
r_{Y_9} &= N_9 + 6N_{10}, \quad k_{Y_9} = k_9 + 6k_{10}, \\
r_{Y_{10}} &= N_{10}, \quad k_{Y_{10}} = k_{10}.
\end{aligned}
\tag{B.2}
$$

We show the application of Proposition 1 for Y_7 through Y_9 because Y_{10} was shown in the proof of Theorem 2.3.3 and the other cases are similar:

$$
\begin{aligned}
\mathbb{P}\left(|Y_7 - (p^4 N_7 + p^4(1-p)N_9 + 3p^4(1-p)^2 N_{10})| \geq p^4 \epsilon (N_7 + N_9 + 3N_{10})\right) \\
\leq 2\exp\left(-\frac{2p^8\epsilon^2(N_7 + N_9 + 3N_{10})}{k_7 + k_9 + 3k_{10}}\right).
\end{aligned}
$$

$$\mathbb{P}\left(|Y_8 - (p^4 N_8 + 4p^4(1-p)N_9 + 12p^4(1-p)^2 N_{10})| \geq p^4\epsilon(N_8 + 4N_9 + 12N_{10})\right)$$
$$\leq 2\exp\left(-\frac{2p^8\epsilon^2(N_8 + 4N_9 + 12N_{10})}{k_8 + 4k_9 + 12k_{10}}\right).$$
$$\mathbb{P}\left(|Y_9 - (p^5 N_9 + 6p^5(1-p)N_{10})| \geq \epsilon(N_9 + 6N_{10})\right)$$
$$\leq 2\exp\left(-\frac{2\epsilon^2(N_9 + 6N_{10})}{k_9 + 6k_{10}}\right)$$
$$\Rightarrow \mathbb{P}\left(|\frac{1}{p^5}Y_9 - (N_9 + 6(1-p)N_{10})| \geq \epsilon(N_9 + 6N_{10})\right)$$
$$\leq 2\exp\left(-\frac{2p^{10}\epsilon^2(N_9 + 6N_{10})}{k_9 + 6k_{10}}\right).$$

$$\mathbb{P}\left(|Y_{10} - p^6 N_{10}| \geq \epsilon N_{10}\right) \leq 2\exp\left(-\frac{2\epsilon^2 N_{10}}{k_{10}}\right)$$
$$\Rightarrow \mathbb{P}\left(|X_{10} - N_{10}| \geq \epsilon N_{10}\right) = \mathbb{P}\left(|Y_3 - p^6 N_{10}| \geq p^6\epsilon N_{10}\right) \leq 2\exp\left(-\frac{2p^{12}\epsilon^2 N_{10}}{k_{10}}\right).$$

Rearranging to solve for p, we have

$$p \geq \left(\frac{\log(2/\delta)k_{10}}{2\epsilon^2 N_{10}}\right)^{1/12}, \quad p \geq \left(\frac{\log(2/\delta)(k_9 + 6k_{10})}{2\epsilon^2(N_9 + 6N_{10})}\right)^{1/10},$$
$$p \geq \left(\frac{\log(2/\delta)(k_8 + 4k_9 + 12k_{10})}{2\epsilon^2(N_8 + 4N_9 + 12N_{10})}\right)^{1/8},$$
$$p \geq \left(\frac{\log(2/\delta)(k_7 + k_9 + 3k_{10})}{2\epsilon^2(N_7 + N_9 + 3N_{10})}\right)^{1/8}, \quad p \geq \left(\frac{\log(2/\delta)(k_6 + k_7 + 2k_9 + 4k_{10})}{2\epsilon^2(N_6 + N_8 + 2N_9 + 4N_{10})}\right)^{1/6},$$
$$p \geq \left(\frac{\log(2/\delta)(k_5 + k_7 + 2k_9 + 4k_{10})}{2\epsilon^2(N_5 + N_8 + 2N_9 + 4N_{10})}\right)^{1/6},$$
$$p \geq \left(\frac{\log(2/\delta)(k_4 + 4k_7 + 2k_8 + 6k_9 + 12k_{10})}{2\epsilon^2(N_4 + 4N_7 + 2N_8 + 6N_9 + 12N_{10})}\right)^{1/6},$$
$$p \geq \left(\frac{\log(2/\delta)(k_3 + 2k_4 + 3k_5 + 3k_6 + 4k_7 + 5k_8 + 8k_9 + 12k_{10})}{2\epsilon^2(N_3 + 2N_4 + 3N_5 + 3N_6 + 4N_7 + 5N_8 + 8N_9 + 12N_{10})}\right)^{1/4},$$
$$p \geq \left(\frac{\log(2/\delta)(k_2 + k_4 + 2k_7 + k_8 + 2k_9 + 3k_{10})}{2\epsilon^2(N_2 + N_4 + 2N_7 + N_8 + 2N_9 + 3N_{10})}\right)^{1/4},$$
$$p \geq \left(\frac{\log(2/\delta)(k_1 + 2k_2 + 2k_3 + 3k_4 + 3k_5 + 3k_6 + 4k_7 + 4k_8 + 5k_9 + 6k_{10})}{2\epsilon^2(N_1 + 2N_2 + 2N_3 + 3N_4 + 3N_5 + 3N_6 + 4N_7 + 4N_8 + 5N_9 + 6N_{10})}\right)^{1/2}. \tag{B.3}$$

The final condition comes from the result for Y_0:

$$n_0 \leq \binom{|V|}{4} - \frac{\log(2/\delta)|V|^2}{2\epsilon^2} \leq |V|^2\left(|V|^2 - \frac{\log(2/\delta)}{2\epsilon^2}\right). \tag{B.4}$$

Plugging into our estimators (given by $\mathbf{H}^{-1}$), we get the following error bounds:

$$\begin{aligned}
\delta X_0 &\leq \epsilon(n_1 + n_2 + n_3) + \epsilon(n_1 + 2n_2 + 3n_3 + n_2 + 3n_3 + n_3) \\
&\leq \epsilon(2n_1 + 4n_2 + 8n_3) \leq 8\epsilon\binom{|V|}{3}. \\
\delta X_1 &\leq \epsilon(N_1 + 2N_2 + 2N_3 + 3N_4 + 3N_5 + 3N_6 + 4N_7 + 4N_8 + 5N_9 + 6N_{10})
\end{aligned}$$

$$
\begin{aligned}
&+ 2\epsilon(N_2 + N_4 + 2N_7 + N_8 + 2N_9 + 3N_{10}) \\
&+ 2\epsilon(N_3 + 2N_4 + 3N_5 + 3N_6 + 4N_7 + 5N_8 + 8N_9 + 12N_{10}) \\
&+ 3\epsilon(N_4 + 4N_7 + 2N_8 + 6N_9 + 12N_{10}) \\
&+ 3\epsilon(N_5 + N_8 + 2N_9 + 4N_{10}) + 3\epsilon(N_6 + N_8 + 2N_9 + 4N_{10}) + 4\epsilon(N_7 + N_9 + 3N_{10}) \\
&+ 4\epsilon(N_8 + 4N_9 + 12N_{10}) + 5\epsilon(N_9 + 6N_{10}) + 6\epsilon(N_{10}) \\
&\leq \epsilon(N_1 + \ldots + 192N_{10}) \leq 192\epsilon\binom{|V|}{4}.
\end{aligned}
$$

$$
\begin{aligned}
\delta X_2 &\leq \epsilon(N_2 + N_4 + 2N_7 + N_8 + 2N_9 + 3N_{10}) + \epsilon(N_4 + 4N_7 + 2N_8 + 6N_9 + 12N_{10}) \\
&+ 2\epsilon(N_7 + N_9 + 3N_{10}) \\
&+ \epsilon(N_8 + 4N_9 + 12N_{10}) + 2\epsilon(N_9 + 6N_{10}) + 3\epsilon(N_{10}) \\
&\leq \epsilon(N_2 + \ldots + 48N_{10}) \leq 48\epsilon\binom{|V|}{4}.
\end{aligned}
$$

$$
\begin{aligned}
\delta X_3 &\leq \epsilon(N_3 + 2N_4 + 3N_5 + 3N_6 + 4N_7 + 5N_8 + 8N_9 + 12N_{10}) \\
&+ 2\epsilon(N_4 + 4N_7 + 2N_8 + 6N_9 + 12N_{10}) \\
&+ 3\epsilon(N_5 + N_8 + 2N_9 + 4N_{10}) + 3\epsilon(N_6 + N_8 + 2N_9 + 4N_{10}) \\
&+ 4\epsilon(N_7 + N_9 + 3N_{10}) + 5\epsilon(N_8 + 4N_9 + 12N_{10}) \\
&+ 8\epsilon(N_9 + 6N_{10}) + 12\epsilon(N_{10}) \\
&\leq \epsilon(N_3 + 4N_4 + 6N_5 + \ldots + 192N_{10}) \leq 192\epsilon\binom{|V|}{4}.
\end{aligned}
$$

$$
\begin{aligned}
\delta X_4 &\leq \epsilon(N_4 + 4N_7 + 2N_8 + 6N_9 + 12N_{10}) + 4\epsilon(N_7 + N_9 + 3N_{10}) + 2\epsilon(N_8 + 4N_9 + 12N_{10}) \\
&+ 6\epsilon(N_9 + 6N_{10}) + 12\epsilon(N_{10}) \\
&\leq \epsilon(N_4 + \ldots + 96N_{10}) \leq 96\epsilon\binom{|V|}{4}.
\end{aligned}
$$

$$
\delta X_5 \leq \epsilon(N_5 + N_8 + 2N_9 + 4N_{10}) + \epsilon(N_8 + 4N_9 + 12N_{10}) + 2\epsilon(N_9 + 6N_{10}) + 4\epsilon(N_{10})
$$

$$\leq \epsilon(N_5 + \ldots + 32N_{10}) \leq 32\epsilon\binom{|V|}{4}.$$

$$\delta X_6 \leq \epsilon(N_6 + N_8 + 2N_9 + 4N_{10}) + \epsilon(N_8 + 4N_9 + 12N_{10}) + 2\epsilon(N_9 + 6N_{10}) + 4\epsilon(N_{10})$$
$$\leq \epsilon(N_6 + \ldots + 32N_{10}) \leq 32\epsilon\binom{|V|}{4}.$$

$$\delta X_7 \leq \epsilon(N_7 + N_9 + 3N_{10}) + \epsilon(N_9 + 6N_{10}) + 3\epsilon(N_{10})$$
$$\leq \epsilon(N_7 + 2N_9 + 12N_{10}) \leq 12\epsilon\binom{|V|}{4}.$$

$$\delta X_8 \leq \epsilon(N_8 + 4N_9 + 12N_{10}) + 4\epsilon(N_9 + 6N_{10}) + 12\epsilon(N_{10})$$
$$\leq \epsilon(N_8 + 8N_9 + 48N_{10}) \leq 48\epsilon\binom{|V|}{4}.$$

$$\delta X_9 \leq \epsilon(N_9 + 6N_{10}) + 6\epsilon(N_{10})$$
$$\leq \epsilon(N_9 + 12N_{10}) \leq 12\epsilon\binom{|V|}{4}.$$

$$\delta X_{10} \leq \epsilon N_{10}.$$

Thus the maximum deviation in any estimator is less than $192\epsilon\binom{|V|}{4}$. Substituting $\tilde{\epsilon}^2 = \epsilon^2/(192)^2 = \epsilon^2/2C$ completes the proof.

□

Appendix C

Non-submodular Parameters

The following appendix surveys several different relaxations of submodularity, along with several bounds relating these quantities. Most of the definitions and results can be found in [16], [73], [83], and [108].

C.1 Definitions

As a reminder, the discrete derivative is defined as $f(\mathsf{B} \mid \mathsf{A}) := f(\mathsf{A} \cup \mathsf{B}) - f(\mathsf{A})$. We assume the ground set to be N and describe properties of a set function $f(\cdot) : 2^{\mathsf{N}} \mapsto \mathbb{R}$.

C.1.1 Submodularity Ratio

Unless otherwise specified, throughout the dissertation we call a set function *weakly submodular* if the submodularity ratio γ is bounded away from zero for all sets of interest.

Definition C.1.1 (Submodularity Ratio, from [16])**.** *Let* $\mathsf{S}, \mathsf{L} \subset \mathsf{N}$ *be two disjoint sets, and* $f(\cdot) : 2^{\mathsf{N}} \mapsto \mathbb{R}$*. The submodularity ratio of* L *with respect to* S *is given by*

$$\gamma_{\mathsf{L},\mathsf{S}} := \frac{\sum_{j \in \mathsf{S}} f(j \mid \mathsf{L})}{f(\mathsf{S} \mid \mathsf{L})} \quad .$$

This is parameterized by an integer in two slightly different ways in Chapters 3 and 4. In both definitions, the integer represents the cardinality of the sets under consideration; the definitions differ depending on whether the sets must be disjoint.

Definition C.1.2 (From [16]). *The submodularity ratio of a set* U *with respect to an integer* k *is given by*

$$\gamma_{\mathsf{U},k} := \min_{\substack{\mathsf{L},\mathsf{S}:\mathsf{L}\cap\mathsf{S}=\emptyset,\\ \mathsf{L}\subseteq\mathsf{U},|\mathsf{S}|\leq k}} \gamma_{\mathsf{L},\mathsf{S}} \quad .$$

Definition C.1.3 (From [116]). *The submodularity ratio with respect to an integer* k *is given by*

$$\gamma_k := \min_{\substack{\mathsf{L},\mathsf{S}:\\ |\mathsf{L}|\leq k,|\mathsf{S}\setminus\mathsf{L}|\leq k}} \gamma_{\mathsf{L},\mathsf{S}} \quad .$$

In [83], the space of feasible sets was restricted further by considering only the sets encountered during the execution of a particular algorithm (*e.g.* Greedy). Recently a complementary parameter was defined over all pairs of disjoint sets.

Definition C.1.4 (From [108]). *The supermodularity ratio is given by*

$$\check{\gamma} := \max_{\mathsf{L},\mathsf{S}:\mathsf{L}\cap\mathsf{S}=\emptyset} \gamma_{\mathsf{L},\mathsf{S}} = \min_{\mathsf{L},\mathsf{S}:\mathsf{L}\cap\mathsf{S}=\emptyset} \frac{f(\mathsf{S}\mid\mathsf{L})}{\sum_{j\in\mathsf{S}} f(j\mid\mathsf{L})} \quad .$$

Clearly $\gamma_k = \min_{\mathsf{U}:|\mathsf{U}|\leq k} \gamma_{\mathsf{U},k}$, and $f(\cdot)$ is modular if and only if $\gamma_{|\mathsf{N}|} = \check{\gamma} = 1$.

Remark C.1.1 (Lattice Weakly Submodular Functions). *The definitions in this section differ from that of [117] for submodular lattice functions. Rather than generalizing submodularity, [117] defines a subclass of submodular functions.*

Definition C.1.5 (From [117]). *Given a lattice $\mathcal{L}$ with join operation $\vee$ and meet operation $\wedge$ (e.g.* max *and* min*, respectively), a function $f(\cdot) : \mathcal{L} \mapsto \mathbb{Z}$ is lattice weakly submodular if for all $\mathsf{A}, \mathsf{B} \in \mathcal{L}$, $f(\mathsf{A}) = f(\mathsf{A} \wedge \mathsf{B}) \Rightarrow f(\mathsf{A} \vee \mathsf{B}) = f(\mathsf{B})$.*

C.1.2 Proportionally Submodular Functions

Next we describe another distinct class of functions called proportional submodular functions [118, 119], which originally shared the name weakly submodular.

Definition C.1.6 (From [119]). *A normalized, monotone function is proportionally submodular if for all $\mathsf{L}, \mathsf{S} \subseteq \mathsf{N}$,*

$$|\mathsf{S}| \cdot f(\mathsf{L}) \; + \; |\mathsf{L}| \cdot f(\mathsf{S}) \geq |\mathsf{L} \cap \mathsf{S}| \cdot f(\mathsf{L} \cup \mathsf{S}) \; + \; |\mathsf{L} \cup \mathsf{S}| \cdot f(\mathsf{L} \cap \mathsf{S}) \quad .$$

C.1.3 Curvature

Curvature was initially defined in [100] to provide tighter bounds than those of [15] for special classes of submodular functions.

Definition C.1.7 (Curvature, from [100]). *The curvature of a function is given by*

$$\kappa := 1 - \min_{j \in \mathsf{N}} \frac{f(j \mid \mathsf{N} \setminus j)}{f(j \mid \emptyset)} \quad .$$

This was later generalized in various ways [83, 131, 132] *e.g.* by replacing the empty set and the ground set with two arbitrary sets.

Definition C.1.8 (Generalized Curvature, from [83]). *The generalized curvature of*

a function is given by

$$\alpha := 1 - \min_{\substack{L,S: \\ j \in S \setminus L}} \frac{f(j \mid (S \setminus j \cup L))}{f(j \mid (S \setminus j))} \quad .$$

Again, we can define a complementary parameter.

Definition C.1.9 (Generalized Inverse Curvature, from [108])**.** *The generalized inverse curvature of a function is given by*

$$\breve{\alpha} := 1 - \max_{\substack{L,S: \\ j \in S \setminus L}} \frac{f(j \mid (S \setminus j \cup L))}{f(j \mid (S \setminus j))} = 1 - \min_{\substack{L,S: \\ j \in S \setminus L}} \frac{f(j \mid (S \setminus j))}{f(j \mid (S \setminus j \cup L))} \quad .$$

C.1.4 Subadditivity Ratio

A relaxed form of subadditivity was used in [81] to analyze the performance of distributed maximization of non-submodular functions.

Definition C.1.10 (From [81])**.** *The bipartite subadditivity ratio with respect to a set* U *is given by*

$$\nu_U := \min_{\substack{L,S: L \cap S = \emptyset \\ L \cup S = U}} \frac{f(L) + f(S)}{f(U)} \quad .$$

Definition C.1.11 (From [81])**.** *The bipartite subadditivity ratio with respect to an integer* k *is given by*

$$\nu_k := \min_{U \subseteq N : |U| = k} \nu_U \quad .$$

Two related quantities were defined recently to analyze the performance of robust non-submodular maximization.

Definition C.1.12 (From [108]). *The subadditivity ratio of a function is given by*

$$\nu := \min_{\mathsf{U}\subseteq\mathsf{N}} \frac{\sum_{j\in\mathsf{U}} f(j)}{f(\mathsf{U})} \quad .$$

Definition C.1.13 (From [108]). *The superadditivity ratio of a function is given by*

$$\check{\nu} := \min_{\mathsf{U}\subseteq\mathsf{N}} \frac{f(\mathsf{U})}{\sum_{j\in\mathsf{U}} f(j)} = \max_{\mathsf{U}\subseteq\mathsf{N}} \frac{\sum_{j\in\mathsf{U}} f(j)}{f(\mathsf{U})} \quad .$$

C.1.5 Submodularity Index

Recently, [142] defined an additive version of the submodularity ratio.

Definition C.1.14 (Submodularity Index, from [142]). *Let* $\mathsf{S},\mathsf{L} \subset \mathsf{N}$ *be two disjoint sets, and* $f(\cdot) : 2^{\mathsf{N}} \mapsto \mathbb{R}$. *The local submodularity index of* L *with respect to* S *is given by*

$$\phi_{\mathsf{L},\mathsf{S}} := \sum_{j\in\mathsf{S}} f(j \mid \mathsf{L}) - f(\mathsf{S} \mid \mathsf{L}) \quad .$$

Definition C.1.15 (From [142]). *The submodularity index of a set* U *with respect to an integer* k *is given by*

$$\phi_{\mathsf{U},k} := \min_{\substack{\mathsf{L},\mathsf{S}:\mathsf{L}\cap\mathsf{S}=\emptyset,\\ \mathsf{L}\subseteq\mathsf{U},|\mathsf{S}|\leq k}} \phi_{\mathsf{L},\mathsf{S}} \quad .$$

The function is called super-submodular if $\phi > 0$ and quasi-submodular if ϕ is only slightly nonnegative.

C.1.6 Approximately Submodular Functions

Recently, approximate submodularity was formulated to capture a notion of noisy oracle access. Query complexity results for maximizing approximate submodular functions were given in [106].

Definition C.1.16 (Approximate Submodularity, from [106])**.** *A function $f(\cdot)$ is ϵ-approximately submodular if there exists a submodular function $g(\cdot)$ such that for all $S \subseteq N$,*

$$(1-\epsilon)g(S) \leq f(S) \leq (1+\epsilon)g(S) \ .$$

C.2 Relations

Several recent results have described relationships between parameters in the previous section. For example, it is clear that $\nu = \gamma_{\emptyset,|N|}$, and more generally we have the following observations:

Proposition C.2.1. *The following statements are equivalent:*

1. *$f(\cdot)$ is submodular.*
2. *$\gamma = 1$.*
3. *$\check{\alpha} = 0$.*

Proposition C.2.2. *The following statements are equivalent:*

1. *$f(\cdot)$ is supermodular.*
2. *$-f(\cdot)$ is submodular.*
3. *$\check{\gamma} = 1$.*
4. *$\alpha = 0$.*

Proposition C.2.3. *The following statements are equivalent:*

1. $f(\cdot)$ *is modular.*

2. $f(\cdot)$ *is both submodular and supermodular.*

Proposition C.2.4. *If $f(\cdot)$ is submodular then $\nu = 1$. If $f(\cdot)$ is supermodular then $\check{\nu} = 1$.*

Proposition C.2.5 (From [108])**.** *$\nu \geq \gamma_{|N|} \geq 1 - \check{\alpha}$ and $\check{\nu} \geq \check{\gamma} \geq 1 - \alpha$.*

Proposition C.2.6 (From [108])**.** *$\theta \geq \check{\nu}\nu$.*

Proposition C.2.7 (From [83])**.** *There exist functions which have submodularity ratio bounded away from 0, have curvature bounded away from 1, and are not proportionally submodular. Furthermore, there exist proportionally submodular functions which do not have submodularity ratio bounded away from 0 and do not have curvature bounded away from 1.*

Proposition C.2.8. *There exists a separation between functions with bounded submodularity ratio and functions with bounded inverse curvature.*

Proof. The function defined in (4.1) has $\gamma = 1/2$ and $\check{\alpha} = 1$. To see this, take $S = \{v_1, v_2, u_1\}$, $L = \{u_2\}$, and $j = \{v_3\}$. We note that this explains how Theorems 4.4.1 and 4.5.6 do not contradict each other. □

Proposition C.2.9. *There exists a separation between functions with bounded subadditivity ratio and functions with bounded submodularity ratio.*

Proof. Let $\mathsf{N} = \{x_1, x_2, x_3\}$ and consider the set function

$$f(S) = \begin{cases} 0, & |S| = 0 \\ 1, & |S| \in \{1, 2\} \\ 2, & |S| = 3 \end{cases} .$$

Clearly, $\nu = 1 > 0$ (take $\mathsf{U} = \{x_1\}$) while $\gamma = 0$ (take $\mathsf{L} = \{x_1\}$ and $\mathsf{S} = \{x_2, x_3\}$). □

Appendix D

Motivating Example: Feature Selection for Linear Regression

To show the impact[1] of submodularity, we construct a linear regression example. Even in $p = 3$ dimensions, the greedy forward selection algorithm's output can be arbitrarily off from the optimal R^2. Consider the following variables:

$$\begin{aligned}
\mathbf{y} &= \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T, \\
\mathbf{x}_1 &= \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T, \\
\mathbf{x}_2 &= \begin{bmatrix} z & \sqrt{1-z^2} & 0 \end{bmatrix}^T, \\
\mathbf{x}_3 &= \begin{bmatrix} 2z & 0 & \sqrt{1-4z^2} \end{bmatrix}^T.
\end{aligned}$$

All variables have unit norm and we wish to choose the 2-subset of $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$ that best estimates $\mathbf{y}$. Since $R^2_1 = 0$, $R^2_2 = z^2$, and $R^2_3 = 4z^2$, $\mathbf{x}_3$ will be selected first ($S^G_1 = \{3\}$) if $z > 0$. $\mathbf{x}_2$ will be chosen next ($S^G_2 = \{3, 2\}$), and solving for R^2 for this pair,

$$R^2_{3,2} = (\mathbf{y}^T \mathbf{X}_{3,2})(\mathbf{X}^T_{3,2} \mathbf{X}_{3,2})^{-1}(\mathbf{X}^T_{3,2} \mathbf{y})$$

[1] The material in this appendix is based on the following journal publication: [73] E. R. Elenberg, R. Khanna, A. G. Dimakis, and S. Negahban. Restricted Strong Convexity Implies Weak Submodularity. *The Annals of Statistics*, 2018 (to appear). It is the dissertation author's contribution.

$$= \frac{1}{1-4z^4}\begin{bmatrix}2z & z\end{bmatrix}\begin{bmatrix}1 & -2z^2\\ -2z^2 & 1\end{bmatrix}\begin{bmatrix}2z\\ z\end{bmatrix} = \frac{5z^2-8z^4}{1-4z^4},$$

which goes to zero as $z \to 0^+$. However, $\mathbf{y} = -\frac{\sqrt{1-z^2}}{z}\mathbf{x}_1 + \frac{1}{z}\mathbf{x}_2$ which makes $R^2_{1,2} = 1$ for the optimal set $\{\mathbf{x}_1, \mathbf{x}_2\}$ ($S_2 = \{1, 2\}$).

Appendix E

Proofs for Chapter 3

E.1 Proof of Theorem 3.4.1

Proof. We proceed[1] by upper bounding the denominator and lower bounding the numerator of (3.2). Let $\overline{k} = |\mathsf{L}| + k$. First, we apply Definition 3.3.3 with $\mathbf{x} = \boldsymbol{\beta}^{(\mathsf{L})}$ and $\mathbf{y} = \boldsymbol{\beta}^{(\mathsf{L}\cup\mathsf{S})}$,

$$\frac{m_{\overline{k}}}{2}\|\boldsymbol{\beta}^{(\mathsf{L}\cup\mathsf{S})} - \boldsymbol{\beta}^{(\mathsf{L})}\|_2^2 \leq l(\boldsymbol{\beta}^{(\mathsf{L})}) - l(\boldsymbol{\beta}^{(\mathsf{L}\cup\mathsf{S})}) + \langle \nabla l(\boldsymbol{\beta}^{(\mathsf{L})}), \boldsymbol{\beta}^{(\mathsf{L}\cup\mathsf{S})} - \boldsymbol{\beta}^{(\mathsf{L})}\rangle. \tag{E.1}$$

Rearranging and noting that $l(\cdot)$ is monotone for increasing supports,

$$\begin{aligned} 0 \leq l(\boldsymbol{\beta}^{(\mathsf{L}\cup\mathsf{S})}) - l(\boldsymbol{\beta}^{(\mathsf{L})}) &\leq \langle \nabla l(\boldsymbol{\beta}^{(\mathsf{L})}), \boldsymbol{\beta}^{(\mathsf{L}\cup\mathsf{S})} - \boldsymbol{\beta}^{(\mathsf{L})}\rangle - \frac{m_{\overline{k}}}{2}\|\boldsymbol{\beta}^{(\mathsf{L}\cup\mathsf{S})} - \boldsymbol{\beta}^{(\mathsf{L})}\|_2^2 \\ &\leq \max_{\mathbf{v}:\mathbf{v}_{(\mathsf{L}\cup\mathsf{S})^c}=0} \langle \nabla l(\boldsymbol{\beta}^{(\mathsf{L})}), \mathbf{v} - \boldsymbol{\beta}^{(\mathsf{L})}\rangle - \frac{m_{\overline{k}}}{2}\|\mathbf{v} - \boldsymbol{\beta}^{(\mathsf{L})}\|_2^2. \end{aligned}$$

Setting $\mathbf{v} = \boldsymbol{\beta}^{(\mathsf{L})} + {}^1\!/_{m_{\overline{k}}}\nabla l(\boldsymbol{\beta}^{(\mathsf{L})})_{\mathsf{S}}$, we have

$$0 \leq l(\boldsymbol{\beta}^{(\mathsf{L}\cup\mathsf{S})}) - l(\boldsymbol{\beta}^{(\mathsf{L})}) \leq \frac{1}{2m_{\overline{k}}}\|\nabla l(\boldsymbol{\beta}^{(\mathsf{L})})_{\mathsf{S}}\|_2^2. \tag{E.2}$$

[1] The material in this appendix is based on the following journal publication: [73] E. R. Elenberg, R. Khanna, A. G. Dimakis, and S. Negahban. Restricted Strong Convexity Implies Weak Submodularity. *The Annals of Statistics*, 2018 (to appear). The dissertation author's primary contributions are the proofs of submodularity ratio lower bound, proofs of all approximation guarantees for Oblivious and Forward Stepwise algorithms, and proof of sufficient conditions for statistical recovery. The dissertation author also assisted with other contributions and is the primary contributor of this paper.

Next, consider a single coordinate $j \in \mathsf{S}$. The function at $\boldsymbol{\beta}^{(\mathsf{L}\cup\{j\})}$ is larger than the function at any other $\boldsymbol{\beta}$ on the same support. In particular $l(\boldsymbol{\beta}^{(\mathsf{L}\cup\{j\})}) \geq l(\mathbf{y}_j)$, where $\mathbf{y}_j := \boldsymbol{\beta}^{(\mathsf{L})} + \alpha_j \boldsymbol{\beta}_j^{(\mathsf{L}\cup\mathsf{S})}$ for some scalar α_i. Noting that $(\mathbf{x} = \boldsymbol{\beta}^{(\mathsf{L})}, \mathbf{y} = \mathbf{y}_j) \in \tilde{\Omega}_{|\mathsf{L}|+1}$ and applying Definition 3.3.3,

$$\begin{aligned} l(\boldsymbol{\beta}^{(\mathsf{L}\cup\{j\})}) - l(\boldsymbol{\beta}^{(\mathsf{L})}) &\geq l(\boldsymbol{\beta}^{(\mathsf{L})} + \alpha_j \boldsymbol{\beta}_j^{(\mathsf{L}\cup\mathsf{S})}) - l(\boldsymbol{\beta}^{(\mathsf{L})}) \\ &\geq \langle \nabla l(\boldsymbol{\beta}^{(\mathsf{L})}), \alpha_j \boldsymbol{\beta}_j^{(\mathsf{L}\cup\mathsf{S})} \rangle - \frac{\tilde{M}_{|\mathsf{L}|+1}}{2} |\alpha_j \boldsymbol{\beta}_j^{(\mathsf{L}\cup\mathsf{S})}|^2. \end{aligned}$$

Summing over all $j \in \mathsf{S}$ and setting

$$\alpha_j = \frac{\langle \nabla l(\boldsymbol{\beta}^{(\mathsf{L})}), \boldsymbol{\beta}_j^{(\mathsf{L}\cup\mathsf{S})} \rangle}{\tilde{M}_{|\mathsf{L}|+1} |\boldsymbol{\beta}_j^{(\mathsf{L}\cup\mathsf{S})}|^2},$$

we have

$$\begin{aligned} l(\boldsymbol{\beta}^{(\mathsf{L}\cup\{j\})}) - l(\boldsymbol{\beta}^{(\mathsf{L})}) &\geq \frac{\left(\langle \nabla l(\boldsymbol{\beta}^{(\mathsf{L})}), \boldsymbol{\beta}_j^{(\mathsf{L}\cup\mathsf{S})} \rangle\right)^2}{2\tilde{M}_{|\mathsf{L}|+1} |\boldsymbol{\beta}_j^{(\mathsf{L}\cup\mathsf{S})}|^2} \\ \Rightarrow \sum_{j\in S} l(\boldsymbol{\beta}^{(\mathsf{L}\cup\{j\})}) - l(\boldsymbol{\beta}^{(\mathsf{L})}) &\geq \frac{1}{2\tilde{M}_{|\mathsf{L}|+1}} \sum_{j\in S} \left(\nabla l(\boldsymbol{\beta}^{(\mathsf{L})})_j\right)^2 \\ &= \frac{1}{2\tilde{M}_{|\mathsf{L}|+1}} \|\nabla l(\boldsymbol{\beta}^{(\mathsf{L})})_{\mathsf{S}}\|_2^2. \end{aligned}$$

Substituting the above line and (E.2) into (3.2), the result follows from taking the minimum over all sets L, S. □

E.2 Proof of Lemma 3.4.1

Proof. Let $\mathsf{S} = [k]$. Since $f(\cdot)$ is monotone, $f(j) \leq f(\mathsf{S})$ for $j \in \mathsf{S}$. Summing over all $j \in \mathsf{S}$ and dividing by k yields the first part of the inequality. The rest of the proof

requires combining several applications of Definition 3.3.3 to the underlying likelihood function l for carefully chosen $\mathbf{x}, \mathbf{y}$. Define a k-sparse $\overline{\boldsymbol{\beta}}$ by $\overline{\boldsymbol{\beta}}_j = \alpha_j \boldsymbol{\beta}_j^{(j)}, j \in \mathrm{S}$ for some positive scalar a_j and 0 elsewhere. First we apply Definition 3.3.3 with $\mathbf{x} = \mathbf{0}$ and $\mathbf{y} = \overline{\boldsymbol{\beta}}$. This implies

$$l(\overline{\boldsymbol{\beta}}) \geq \langle \nabla l(\mathbf{0}), \overline{\boldsymbol{\beta}} \rangle - \frac{M_k}{2} \sum_{j \in \mathrm{S}} |\alpha_j \boldsymbol{\beta}_j^{(j)}|^2. \tag{E.3}$$

Next, applying the same definition k times with $\mathbf{x} = \mathbf{0}$ and $\mathbf{y} = \boldsymbol{\beta}^{(j)}$ and summing over $j \in \mathrm{S}$,

$$\begin{aligned} \langle \nabla l(\mathbf{0}), \alpha_j \boldsymbol{\beta}^{(j)} \rangle &\geq \alpha_j \left(l(\boldsymbol{\beta}^{(j)}) + \frac{m_1}{2} |\boldsymbol{\beta}_j^{(j)}|^2 \right) \\ \Rightarrow \langle \nabla l(\mathbf{0}), \overline{\boldsymbol{\beta}} \rangle &\geq \sum_{j \in \mathrm{S}} \alpha_j l(\boldsymbol{\beta}^{(j)}) + \alpha_j \frac{m_1}{2} |\boldsymbol{\beta}_j^{(j)}|^2. \end{aligned} \tag{E.4}$$

Combining (E.3) with (E.4), and setting

$$\alpha_j = \frac{m_1}{2M_k} + \frac{l(\boldsymbol{\beta}^{(j)})}{M_k |\boldsymbol{\beta}_j^{(j)}|^2},$$

we have

$$l(\overline{\boldsymbol{\beta}}) \geq \sum_{j \in \mathrm{S}} \frac{m_1}{2M_k} l(\boldsymbol{\beta}^{(j)}) + \frac{m_1^2}{8M_k} |\boldsymbol{\beta}^{(j)}|^2 + \frac{(l(\boldsymbol{\beta}^{(j)}))^2}{2M_k |\boldsymbol{\beta}_j^{(j)}|^2}. \tag{E.5}$$

Now applying Definition 3.3.3 with $\mathbf{x} = \boldsymbol{\beta}^{(j)}$ and $\mathbf{y} = \mathbf{0}$,

$$\frac{M_1}{2} |\boldsymbol{\beta}_j^{(j)}|^2 \geq l(\boldsymbol{\beta}^{(j)}) \geq \frac{m_1}{2} |\boldsymbol{\beta}_j^{(j)}|^2. \tag{E.6}$$

Combining (E.5) and (E.6), we have

$$l(\overline{\boldsymbol{\beta}}) \geq \sum_{j \in \mathrm{S}} \frac{m_1}{2M_k} l(\boldsymbol{\beta}^{(j)}) + \frac{m_1^2}{4M_k M_1} l(\boldsymbol{\beta}^{(j)}) + \frac{m_1}{4M_k} l(\boldsymbol{\beta}^{(j)})$$

$$= \sum_{j \in S} \left(\frac{3m_1}{4M_k} + \frac{m_1^2}{4M_k M_1} \right) l(\boldsymbol{\beta}^{(j)}).$$

Since $l(\boldsymbol{\beta}^{(S)})$ optimizes l over all vectors with support in S,

$$f(S) = l(\boldsymbol{\beta}^{(S)}) \geq l(\overline{\boldsymbol{\beta}}) \geq \frac{m_1}{4M_k} \left(3 + \frac{m_1}{M_1} \right) \sum_{j \in S} l(\boldsymbol{\beta}^{(j)})$$
$$= \frac{m_1}{4M_k} \left(3 + \frac{m_1}{M_1} \right) \sum_{j=1}^{k} f(j). \qquad \square$$

E.3 Proof of Theorem 3.4.2

Proof. Let S be the set of size k selected by the Oblivious algorithm and S^* be the optimal set of size k corresponding to values f^{OBL} and f^{OPT}. By definition, $\sum_{j \in S} f(j) \geq \sum_{j \in S^*} f(j)$. Letting $C = \max\{1/k, 3m/4M + m^2/4M^2)\}$ and combining Lemma 3.4.1 with Theorem 3.4.1,

$$f^{OBL} = f(S) \geq C \sum_{j \in S} f(j)$$
$$\geq C \sum_{j \in S^*} f(j) \geq C\gamma_{\emptyset,k} f(S^*) \geq C \left(\frac{m_k}{M_1} \right) f(S^*)$$
$$= C \left(\frac{m_k}{M_1} \right) f^{OPT}. \qquad \square$$

E.4 Proof of Theorem 3.4.3

Proof. Let $l(\cdot)$ be the log likelihood function and let S_i^G be the set selected by the Forward Stepwise algorithm at iteration i. Define $A(i)$ as the incremental greedy gain $f(S_i^G) - f(S_{i-1}^G)$ with $A(0) = 0$. Denote the remainder set at iteration i as

$S_i^R = S^* \backslash S_i^G$, and define $B(i) = f(S^*) - f(S_i^G)$, the incremental gain from adding the optimal set. Lemma E.4.1 relates these two quantities.

Lemma E.4.1. *At iteration i, the incremental gain from selecting the next greedy item is related to the incremental gain from adding the rest of the optimal set S^* by the following:*

$$A(i+1) \geq \frac{\gamma_{S_i^G,k}}{k} B(i) \,.$$

Proof. Let $S = S_i^G$ be the set selected by the greedy algorithm at iteration i, S^* be the optimal feature set on k variables, and S^R be the remainder set $S^* \backslash S$. S^R is a subset of the candidate variables available to the greedy algorithm at iteration $i+1$. Using Definition 3.3.2 and the fact that $k \geq |S^R|$,

$$\begin{aligned} kA(i+1) \geq |S^R| A(i+1) &\geq |S^R| \max_{j \in S^R} f(S \cup j) - f(S) \\ &\geq \sum_{j \in S^R} [f(S \cup j) - f(S)] \\ &\geq \gamma_{S,|S^R|} \left(f(S \cup S^R) - f(S) \right) \geq \gamma_{S,k} B(i), \end{aligned}$$

where the last inequality follows from the fact that $S \cup S^R \supseteq S^*$. □

Given Theorem 3.4.1 and Lemma E.4.1, the rest of the proof follows the standard approximation bound for maximizing a normalized, monotone submodular function (refer to [15] or the survey [134]). Next, observe that $A(i+1) = B(i) - B(i+1)$. Combining this with Lemma E.4.1 and letting $C = \gamma_{S_k^G,k}/k$, we have the following inequality:

$$B(i+1) \leq (1 - C)\, B(i),$$

which implies

$$B(i) \leq (1 - C)^i B(0),$$

for all iterations $1 \leq i \leq k$. Setting $i = k$ and substituting $B(k) = f^{OPT} - f^{FS}$ and $B(0) = f^{OPT}$,

$$\begin{aligned} f^{OPT} - f^{FS} &\leq (1 - C)^k f^{OPT} \\ \Rightarrow f^{FS} &\geq f^{OPT} \left[1 - (1 - C)^k\right] \geq f^{OPT} \left(1 - e^{-\gamma_{S_k^G, k}}\right). \end{aligned}$$

The claim follows from applying Theorem 3.4.1. □

E.5 Proof of Theorem 3.4.4

Proof. First we prove the following lemma which bounds the ratio of the objective between optimal sets S_k and S_{k-1} in terms of their smoothness and convexity parameters.

Lemma E.5.1. *Let S_k be the optimal subset of size k, and let m be the restricted strong concavity parameter on Ω_k. Let k' satisfy $M'/m < k' < k$, where M' is the restricted smoothness parameter of $l(\cdot)$ on $\tilde{\Omega}_k$. Then for large enough k,*

$$l(\boldsymbol{\beta}^{(S_{k'})}) \geq l(\boldsymbol{\beta}^{(S_k)}) \Theta\left(\left(\frac{k'}{k}\right)^{M'/m}\right) \Rightarrow l(\boldsymbol{\beta}^{(S_{k/2})}) \geq l(\boldsymbol{\beta}^{(S_k)}) \Theta\left(2^{-M'/m}\right).$$

Proof. Let j be the index that minimizes $|\boldsymbol{\beta}_j^{(S_k)}|^2$. By M'-smoothness on $\tilde{\Omega}_k$ and the fact that the min is smaller than the average,

$$l(\boldsymbol{\beta}^{(S_{k-1})}) \geq l(\boldsymbol{\beta}^{(S_k)}_{S_k \setminus \{j\}})$$

$$\geq l(\boldsymbol{\beta}^{(S_k)}) + \langle \nabla l(\boldsymbol{\beta}^{(S_k)}), \boldsymbol{\beta}^{(S_k)}_{S_k \setminus \{j\}} - \boldsymbol{\beta}^{(S_k)} \rangle - \frac{M'}{2} \|\boldsymbol{\beta}^{(S_k)}_{S_k \setminus \{j\}} - \boldsymbol{\beta}^{(S_k)}\|_2^2$$
$$= l(\boldsymbol{\beta}^{(S_k)}) - \frac{M'}{2} |\boldsymbol{\beta}^{(S_k)}_j|^2.$$

This implies

$$\frac{l(\boldsymbol{\beta}^{(S_{k-1})})}{l(\boldsymbol{\beta}^{(S_k)})} \geq 1 - \frac{M' \|\boldsymbol{\beta}^{(S_k)}\|_2^2}{2k l(\boldsymbol{\beta}^{(S_k)})} .$$

Assuming that $l(\boldsymbol{\beta}^{(\emptyset)}) = 0$ and using m-strong concavity on Ω_k,

$$l(\boldsymbol{\beta}^{(\emptyset)}) - l(\boldsymbol{\beta}^{(S_k)}) \leq -\frac{m}{2} \|\boldsymbol{\beta}^{(S_k)} - \boldsymbol{\beta}^{(\emptyset)}\|_2^2 \Rightarrow -\frac{\|\boldsymbol{\beta}^{(S_k)}\|_2^2}{l(\boldsymbol{\beta}^{(S_k)})} \geq -\frac{2}{m}$$
$$\Rightarrow \frac{l(\boldsymbol{\beta}^{(S_{k-1})})}{l(\boldsymbol{\beta}^{(S_k)})} \geq 1 - \frac{M'}{km} .$$

Then applying iteratively for M'/m constant, k large, and $M'/m < k' < k$, as in [16] we have

$$l(\boldsymbol{\beta}^{(S_{k'})}) \geq l(\boldsymbol{\beta}^{(S_k)}) \prod_{j=k'+1}^{k} 1 - \frac{M'}{jm} = l(\boldsymbol{\beta}^{(S_k)}) \Theta\left(\left(\frac{k'}{k}\right)^{M'/m}\right). \qquad \square$$

Observe that the assumptions of Lemma E.5.1 are satisfied. Combining with Theorem 3.4.3,

$$l(\boldsymbol{\beta}^{(S_k^G)}) \geq l(\boldsymbol{\beta}^{(S_{k/2}^G)}) \geq l(\boldsymbol{\beta}^{(S_{k/2})}) \left(1 - e^{-\gamma_{S_{k/2}^G, k/2}}\right)$$
$$\geq l(\boldsymbol{\beta}^{(S_k)}) \Theta\left(2^{-M'/m}\right) \left(1 - e^{-\gamma_{S_{k/2}^G, k/2}}\right)$$
$$\Rightarrow l(\boldsymbol{\beta}^{(S_k^G)}) \geq l(\boldsymbol{\beta}^{(S_k)}) \Theta\left(2^{-M'/m'}\right) \left(1 - e^{-m'/M'}\right). \qquad \square$$

E.6 Proof of Theorem 3.4.5

Proof. The key idea at each step i is to lower bound the incremental gain from the index chosen by OMP. This is similar to the proof of Theorem 3.4.3, as well as [82] in

which a matrix completion objective is considered. Let $\mathrm{S} = \mathrm{S}_i^P$ be the set chosen by OMP up to iteration i. Given S, let v be the index that would be selected by running one additional step of OMP. Define $D(i+1) = f(\mathrm{S}_{i+1}^P) - f(\mathrm{S}) = l(\boldsymbol{\beta}^{(\mathrm{S}\cup\{v\})}) - l(\boldsymbol{\beta}^{(\mathrm{S})})$, and define $\tilde{B}(i) = f(\mathrm{S}^*) - f(\mathrm{S})$.

Lemma E.6.1. *At iteration i, the incremental gain from selecting the next item via OMP is related to the incremental gain from adding the rest of the optimal set* S^* *by the following:*

$$D(i+1) \geq \frac{m_{i+k}}{k\tilde{M}_{i+1}}\tilde{B}(i).$$

Proof. We begin similar to the proof of Theorem 3.4.1. Let $M = \tilde{M}_{i+1}$, $m = m_{i+k}$, and $\mathbf{e}_v$ be the unit vector with one at coordinate v. By Definition 3.3.3 with $\mathbf{x} = \boldsymbol{\beta}^{(\mathrm{S})}$ and $\mathbf{y} = \boldsymbol{\beta}^{(\mathrm{S})} + \alpha\mathbf{e}_v$ for any scalar α,

$$\begin{aligned} D(i+1) \geq l(\mathbf{y}) - l(\mathbf{x}) &\geq \left\langle \nabla l(\boldsymbol{\beta}^{(\mathrm{S})}), \alpha\mathbf{e}_v \right\rangle - \frac{M}{2}\alpha^2 \\ &= \alpha\|\nabla l(\boldsymbol{\beta}^{(\mathrm{S})})\|_\infty - \frac{M}{2}\alpha^2 \,, \end{aligned}$$

since OMP chooses the coordinate which maximizes the gradient. Substituting

$$\alpha = \frac{\|\nabla l(\boldsymbol{\beta}^{(\mathrm{S})})\|_\infty}{M} \,,$$

we have

$$D(i+1) \geq \frac{1}{2M}\|\nabla l(\boldsymbol{\beta}^{(\mathrm{S})})\|_\infty^2$$

Let $\mathrm{S}^R = \mathrm{S}^* \backslash \mathrm{S}$. Since $|\mathrm{S}^R| \leq k$,

$$D(i+1) \geq \frac{1}{2kM}\sum_{j\in\mathrm{S}^R}\left\langle \nabla l(\boldsymbol{\beta}^{(\mathrm{S})}), \mathbf{e}_j \right\rangle^2 = \frac{1}{2kM}\|\nabla l(\boldsymbol{\beta}^{(\mathrm{S})})_{\mathrm{S}^R}\|_2^2.$$

Substituting (E.2) into the above and noting that $S \cup S^R \supseteq S^*$, we have

$$D(i+1) \geq \frac{m}{kM}\left(l(\boldsymbol{\beta}^{(S\cup S^R)}) - l(\boldsymbol{\beta}^{(S)})\right) \geq \frac{m}{kM}\tilde{B}(i) \ . \qquad \square$$

Given Lemma E.6.1, the rest of the proof follows that of Theorem 3.4.3. $\square$

E.7 Proof of Theorem 3.5.1

Proof. Let $C = C_{s,r}$ and $\Delta = \widehat{\boldsymbol{\beta}}^r - \boldsymbol{\beta}^s$, which is at most an $(s+r)$-sparse vector. Recall that by the definition of Restricted Strong Concavity on Ω_{s+r} we have

$$l(\widehat{\boldsymbol{\beta}}^r) - l(\boldsymbol{\beta}^s) - \langle \nabla l(\boldsymbol{\beta}^s), \Delta\rangle \leq \frac{-m_{s+r}}{2}\|\Delta\|_2^2. \tag{E.7}$$

Furthermore, simple calculations show that

$$l(\widehat{\boldsymbol{\beta}}^r) - l(\boldsymbol{\beta}^s) \geq (1-C)[l(\mathbf{0}) - l(\boldsymbol{\beta}^s)]. \tag{E.8}$$

Subtracting $\langle \nabla l(\boldsymbol{\beta}^s), \Delta\rangle$ from both sides of (E.8) we have

$$l(\widehat{\boldsymbol{\beta}}^r) - l(\boldsymbol{\beta}^s) - \langle \nabla l(\boldsymbol{\beta}^s), \Delta\rangle \geq -\langle \nabla l(\boldsymbol{\beta}^s), \Delta\rangle + (1-C)[l(\mathbf{0}) - l(\boldsymbol{\beta}^s)] \ .$$

Applying (E.7) yields

$$\frac{-m_{s+r}}{2}\|\Delta\|_2^2 \geq -\langle \nabla l(\boldsymbol{\beta}^s), \Delta\rangle + (1-C)[l(\mathbf{0}) - l(\boldsymbol{\beta}^s)] \ .$$

Next, note that

$$-\langle \nabla l(\boldsymbol{\beta}^s), \Delta\rangle \geq -\|\nabla l(\boldsymbol{\beta}^s)\|_{2,s+r}\|\Delta\|_2 \ .$$

Thus,

$$\frac{-m_{s+r}}{2}\|\Delta\|_2^2 \geq -\|\nabla l(\boldsymbol{\beta}^s)\|_{2,k}\|\Delta\|_2 + (1-C)[l(\mathbf{0}) - l(\boldsymbol{\beta}^s)] \ .$$

Recalling that for any positive numbers $2ab \leq ca^2 + b^2/c$ and flipping the above inequality,

$$\frac{m_{s+r}}{2}\|\Delta\|_2^2 \leq \frac{\|\nabla l(\boldsymbol{\beta}^s)\|_{2,s+r}^2}{m_{s+r}} + \frac{m_{s+r}\|\Delta\|_2^2}{4} + (1-C)[l(\boldsymbol{\beta}^s) - l(\mathbf{0})] \ .$$

Rearranging terms we have the final result. □

E.8 Proof of Corollary 3.5.1

Proof. Using a result of [74], we have that

$$\|\nabla l(\boldsymbol{\beta}^s)\|_{2,s+r}^2 \leq (s+r)\|\nabla l(\boldsymbol{\beta}^s)\|_\infty^2 \leq \frac{(s+r)\sigma^2 \log p}{n} \ ,$$

with probability at least $1 - 1/p$.

The minimum eigenvalue of the matrix Σ is 1, while the maximum s-sparse eigenvalue behaves like $1 + s$. Hence, an RIP type condition will not hold. However, in our setting, we simply require a bound on M_1. It can be shown using tail bounds for χ^2-random variables that with high probability $M_1 \leq 4$. Letting $\rho(\Sigma)^2 = \max_i \Sigma_{ii}$, and using a result by Raskutti et. al. [143], we have that for all $\mathbf{v} \in \mathbb{R}^p$,

$$\begin{aligned}
\frac{\|\mathbf{X}\mathbf{v}\|_2}{\sqrt{n}} &\geq 1/4\|\Sigma^{1/2}\mathbf{v}\|_2 - 9\rho(\Sigma)\sqrt{\frac{\log p}{n}}\|\mathbf{v}\|_1 \\
&\geq \left((1-1/c)\,\frac{\lambda_{\min}(\Sigma)}{16} + (1-c)\,81\rho(\Sigma)^2\frac{\log p}{n}(s+r)\right)\|\mathbf{v}\|_2^2 \\
\Rightarrow m_{s+r} \geq \min_{\substack{\mathbf{v}:\|\mathbf{v}\|_2=1,\\ \|\mathbf{v}\|_0 \leq s+r}} \frac{\|\mathbf{X}\mathbf{v}\|_2^2}{n} &\geq \frac{1}{32} - \frac{162(s+r)\log p}{n} \ ,
\end{aligned}$$

with high probability. Therefore, $\gamma \geq \frac{1}{128} - \frac{81(s+r)\log p}{2n}$ and with probability at least $1 - p^{-\Omega(1)} - e^{-\Omega(n)}$,

$$\|\widehat{\boldsymbol{\beta}}^r - \boldsymbol{\beta}^*\|_2^2 \leq \frac{4}{m_{s+r}^2}\frac{(s+r)\sigma^2 \log p}{n} + \frac{8(s+1)}{m_{s+r}}(1 - C_{s,r}) \ ,$$

where we have used the fact that $l(\boldsymbol{\beta}^*) - l(\mathbf{0}) \leq \lambda_{\max}(\hat{\Sigma}_s) \leq 2(s+1)$ with high probability. Note that using arguments from [120, 144] we can apply the above results to the setting of generalized linear models.

Now let $(s+r)\sigma^2 \log p = o(n)$ and $r = \Omega(s \log n)$. Combined with Corollary 3.4.1, this implies that $\|\widehat{\boldsymbol{\beta}}^r - \boldsymbol{\beta}^*\|_2^2 = n^{-\Omega(1)}$ with probability $1 - p^{-\Omega(1)} - e^{-\Omega(n)}$. □

Appendix F

Proofs for Chapter 4

F.1 Proof of Lemma 4.4.1

The nonnegativity[1] and monotonicity of f_k follow immediately from the fact that $u(\mathsf{S})$ and $v(\mathsf{S})$ have these properties. Thus, it remains to prove that f_k is 0.5-weakly submodular for $|\mathsf{N}_k|$, *i.e.*, that for every pair of arbitrary sets $S, L \subseteq \mathsf{N}_k$ it holds that

$$\sum_{w \in \mathsf{S} \setminus \mathsf{L}} f_k(w \mid \mathsf{L}) \geq 0.5 \cdot f_k(\mathsf{S} \mid \mathsf{L}) \ .$$

There are two cases to consider. The first case is that $f_k(\mathsf{L}) = 2 \cdot u(\mathsf{L}) + 1$. In this case $S \setminus \mathsf{L}$ must contain at least $\lceil f_k(\mathsf{S} \mid \mathsf{L})/2 \rceil$ elements of $\{u_i\}_{i=1}^k$. Additionally, the marginal contribution to L of every element of $\{u_i\}_{i=1}^k$ which does not belong to L is at least 1. Thus, we get

$$\begin{aligned} \sum_{w \in \mathsf{S} \setminus \mathsf{L}} f_k(w \mid \mathsf{L}) &\geq \sum_{w \in (\mathsf{S} \setminus \mathsf{L}) \cap \{u_i\}_{i=1}^k} f_k(w \mid \mathsf{L}) \geq |(S \setminus \mathsf{L}) \cap \{u_i\}_{i=1}^k| \\ &\geq \lceil f_k(\mathsf{S} \mid \mathsf{L})/2 \rceil \geq 0.5 \cdot f_k(\mathsf{S} \mid \mathsf{L}) \ . \end{aligned}$$

[1] Parts of the material in this appendix are based on the following conference paper: [116] E. R. Elenberg, A. G. Dimakis, M. Feldman, and A. Karbasi. Streaming Weak Submodularity: Interpreting Neural Networks on the Fly. In *NIPS*, pages 4047–4057, 2017. From that material, the dissertation author's primary contributions are design of the parameter a and the proof of Theorem 4.5.1. The dissertation author also assisted with other contributions and is the primary contributor of this paper. Additional material in this appendix is the dissertation author's contribution.

The second case is that $f_k(\mathsf{L}) = 2 \cdot v(\mathsf{L})$. In this case $\mathsf{S} \setminus \mathsf{L}$ must contain at least $\lceil f_k(\mathsf{S} \mid \mathsf{L})/2 \rceil$ elements of $\{v_i\}_{i=1}^k$, and in addition, the marginal contribution to L of every element of $\{v_i\}_{i=1}^k$ which does not belong to L is at least 1. Thus, we get in this case again

$$\begin{aligned}\sum_{w \in \mathsf{S} \setminus \mathsf{L}} f_k(w \mid \mathsf{L}) &\geq \sum_{w \in (\mathsf{S} \setminus \mathsf{L}) \cap \{v_i\}_{i=1}^k} f_k(w \mid \mathsf{L}) \geq |(\mathsf{S} \setminus \mathsf{L}) \cap \{v_i\}_{i=1}^k| \\ &\geq \lceil f_k(\mathsf{S} \mid \mathsf{L})/2 \rceil \geq 0.5 \cdot f_k(\mathsf{S} \mid \mathsf{L}) \ . \end{aligned}$$

□

F.2 Proof of Theorem 4.4.1

Consider an arbitrary (randomized) streaming algorithm ALG aiming to maximize $f_k(\mathsf{S})$ subject to the cardinality constraint $|\mathsf{S}| \leq 2k$. Since *ALG* uses $o(N)$ memory, we can guarantee, by choosing a large enough d, that ALG uses no more than $(c/4) \cdot N$ memory. In order to show that ALG performs poorly, consider the case that it gets first the elements of $\{u_i\}_{i=1}^k$ and the dummy elements (in some order to be determined later), and only then it gets the elements of $\{v_i\}_{i=1}^k$. The next lemma shows that some order of the elements of $\{u_i\}_{i=1}^k$ and the dummy elements is bad for ALG.

Lemma F.2.1. *There is an order for the elements of $\{u_i\}_{i=1}^k$ and the dummy elements which guarantees that in expectation ALG returns at most $(c/2) \cdot k$ elements of $\{u_i\}_{i=1}^k$.*

Proof. Let W be the set of the elements of $\{u_i\}_{i=1}^k$ and the dummy elements. Observe that the value of f_k for every subset of W is 0. Thus, ALG has no way to differentiate between the elements of W until it views the first element of $\{v_i\}_{i=1}^k$,

which implies that the probability of every element $w \in W$ to remain in ALG's memory until the moment that the first element of $\{v_i\}_{i=1}^{k}$ arrives is determined only by w's arrival position. Hence, by choosing an appropriate arrival order one can guarantee that the sum of the probabilities of the elements of $\{u_i\}_{i=1}^{k}$ to be at the memory of ALG at this point is at most

$$\frac{kM}{|W|} \leq \frac{k(c/4) \cdot N}{k+d} = \frac{k(c/4) \cdot (2k+d)}{k+d} \leq \frac{kc}{2} \ ,$$

where M is the amount of memory ALG uses. □

The expected value of the solution produced by ALG for the stream order provided by Lemma F.2.1 is at most $ck + 1$. Hence, its approximation ratio for $k > 1/c$ is at most

$$\frac{ck+1}{2k} = \frac{c}{2} + \frac{1}{2k} < c \ . \qquad \square$$

F.3 Proof of Observation 4.5.3

Algorithm 8 adds an element u to the set S only when the marginal contribution of u with respect to S is at least τ/k. Thus, it is always true that

$$f(S) \geq \frac{\tau \cdot |S|}{k} \ . \qquad \square$$

F.4 Proof of Proposition 4.5.1

We begin by proving several intermediate lemmas. Recall that $\gamma := \gamma_k$, and notice that by the monotonicity of f we may assume that OPT is of size k. For every $0 \leq i \leq |OPT| = k$, let OPT_i be the random set consisting of the last i elements of

OPT according to the input order. Note that OPT_i is simply a uniformly random subset of OPT of size i. Thus, we can lower bound its expected value as follows.

Lemma F.4.1. *For every* $0 \leq i \leq k$, $\mathbf{E}[f(\mathsf{OPT}_i)] \geq [1 - (1 - \gamma/k)^i] \cdot f(\mathsf{OPT})$.

Proof. We prove the lemma by induction on i. For $i = 0$ the lemma follows from the nonnegativity of f since

$$f(\mathsf{OPT}_0) \geq 0 = [1 - (1 - \gamma/k)^0] \cdot f(\mathsf{OPT}) \enspace .$$

Assume now that the lemma holds for some $0 \leq i - 1 < k$, and let us prove it holds also for i. Since OPT_{i-1} is a uniformly random subset of OPT of size $i - 1$, and OPT_i is a uniformly random subset of OPT of size i, we can think of OPT_i as obtained from OPT_{i-1} by adding to this set a uniformly random element of $\mathsf{OPT} \setminus \mathsf{OPT}_{i-1}$. Taking this point of view, we get, for every set $T \subseteq \mathsf{OPT}$ of size $i - 1$,

$$\begin{aligned}
\mathbf{E}[f(\mathsf{OPT}_i) \mid \mathsf{OPT}_{i-1} = T] &= f(T) + \frac{\sum_{u \in \mathsf{OPT} \setminus T} f(u \mid T)}{|\mathsf{OPT} \setminus T|} \\
&\geq f(T) + \frac{1}{k} \cdot \sum_{u \in \mathsf{OPT} \setminus T} f(u \mid T) \\
&\geq f(T) + \frac{\gamma}{k} \cdot f(\mathsf{OPT} \setminus T \mid T) \\
&= \left(1 - \frac{\gamma}{k}\right) \cdot f(T) + \frac{\gamma}{k} \cdot f(\mathsf{OPT}) \enspace ,
\end{aligned}$$

where the last inequality holds by the γ-weak submodularity of f. Taking expectation over the set OPT_{i-1}, the last inequality becomes

$$\mathbf{E}[f(\mathsf{OPT}_i)] \geq \left(1 - \frac{\gamma}{k}\right) \mathbf{E}[f(\mathsf{OPT}_{i-1})] + \frac{\gamma}{k} \cdot f(\mathsf{OPT})$$

$$\geq \left(1 - \frac{\gamma}{k}\right) \cdot \left[1 - \left(1 - \frac{\gamma}{k}\right)^{i-1}\right] \cdot f(\mathrm{OPT}) + \frac{\gamma}{k} \cdot f(\mathrm{OPT})$$
$$= \left[1 - \left(1 - \frac{\gamma}{k}\right)^{i}\right] \cdot f(\mathrm{OPT}) \ ,$$

where the second inequality follows from the induction hypothesis. □

Let us now denote by $o_1, o_2, \ldots, o_k$ the k elements of OPT in the order in which they arrive, and, for every $1 \leq i \leq k$, let S_i be the set S of Algorithm 8 immediately before the algorithm receives o_i. Additionally, let A_i be an event fixing the arrival time of o_i, the set of elements arriving before o_i and the order in which they arrive. Note that conditioned on A_i, the sets S_i and OPT_{k-i+1} are both deterministic.

Lemma F.4.2. *For every $1 \leq i \leq k$ and event A_i, $\mathbf{E}[f(o_i \mid \mathrm{S}_i) \mid \mathrm{A}_i] \geq (\gamma/k) \cdot [f(\mathrm{OPT}_{k-i+1}) - f(\mathrm{S}_i)]$, where OPT_{k-i+1} and S_i represent the deterministic values these sets take given A_i.*

Proof. By the monotonicity and γ-weak submodularity of f, we get

$$\begin{aligned}\sum_{u \in \mathrm{OPT}_{k-i+1}} f(u \mid \mathrm{S}_i) &\geq \gamma \cdot f(\mathrm{OPT}_{k-i+1} \mid \mathrm{S}_i) \\ &= \gamma \cdot [f(\mathrm{OPT}_{k-i+1} \cup \mathrm{S}_i) - f(\mathrm{S}_i)] \\ &\geq \gamma \cdot [f(\mathrm{OPT}_{k-i+1}) - f(\mathrm{S}_i)] \ .\end{aligned}$$

Since o_i is a uniformly random element of OPT_{k-i+1}, even conditioned on A_i, the last inequality implies

$$\mathbf{E}[f(o_i \mid \mathrm{S}_i) \mid \mathrm{A}_i] = \frac{\sum_{u \in \mathrm{OPT}_{k-i+1}} f(u \mid \mathrm{S}_i)}{k - i + 1}$$

$$\geq \frac{\sum_{u \in \mathrm{OPT}_{k-i+1}} f(u \mid \mathrm{S}_i)}{k}$$
$$\geq \frac{\gamma \cdot [f(\mathrm{OPT}_{k-i+1}) - f(\mathrm{S}_i)]}{k} \quad . \qquad \square$$

Let Δ_i be the increase in the value of S in the iteration of Algorithm 8 in which it gets o_i.

Lemma F.4.3. *Fix $1 \leq i \leq k$ and event A_i, and let OPT_{k-i+1} and S_i represent the deterministic values these sets take given A_i. If $f(\mathrm{S}_i) < \tau$, then $\mathbf{E}[\Delta_i \mid \mathrm{A}_i] \geq [\gamma \cdot f(\mathrm{OPT}_{k-i+1}) - 2\tau]/k$.*

Proof. Notice that by Observation 4.5.3 the fact that $f(\mathrm{S}_i) < \tau$ implies that S_i contains less than k elements. Thus, conditioned on A_i, Algorithm 8 adds o_i to S whenever $f(o_i \mid \mathrm{S}_i) \geq \tau/k$, which means that

$$\Delta_i = \begin{cases} f(o_i \mid \mathrm{S}_i) & \text{if } f(o_i \mid \mathrm{S}_i) \geq \tau/k \quad , \\ 0 & \text{otherwise} \quad . \end{cases}$$

One implication of the last equality is

$$\mathbf{E}[\Delta_i \mid \mathrm{A}_i] \geq \mathbf{E}[f(o_i \mid \mathrm{S}_i) \mid \mathrm{A}_i] - \tau/k \quad ,$$

which intuitively means that the contribution to $\mathbf{E}[f(o_i \mid \mathrm{S}_i) \mid \mathrm{A}_i]$ of values of $f(o_i \mid \mathrm{S}_i)$ which are too small to make the algorithm add o_i to S is at most τ/k. The lemma now follows by observing that Lemma F.4.2 and the fact that $f(\mathrm{S}_i) < \tau$ guarantee

$$\mathbf{E}[f(o_i \mid \mathrm{S}_i) \mid \mathrm{A}_i] \geq (\gamma/k) \cdot [f(\mathrm{OPT}_{k-i+1}) - f(\mathrm{S}_i)]$$
$$> (\gamma/k) \cdot [f(\mathrm{OPT}_{k-i+1}) - \tau]$$
$$\geq [\gamma \cdot f(\mathrm{OPT}_{k-i+1}) - \tau]/k \quad . \qquad \square$$

We are now ready to put everything together and get a lower bound on $\mathbf{E}[\Delta_i]$.

Lemma F.4.4. *For every* $1 \le i \le k$,

$$\mathbf{E}[\Delta_i] \ge \frac{\gamma \cdot [\Pr[\mathcal{E}] - (1-\gamma/k)^{k-i+1}] \cdot f(\mathrm{OPT}) - 2\tau}{k} \ .$$

Proof. Let $\mathcal{E}_i$ be the event that $f(\mathrm{S}_i) < \tau$. Clearly $\mathcal{E}_i$ is the disjoint union of the events A_i which imply $f(\mathrm{S}_i) < \tau$, and thus, by Lemma F.4.3,

$$\mathbf{E}[\Delta_i \mid \mathcal{E}_i] \ge [\gamma \cdot \mathbf{E}[f(\mathrm{OPT}_{k-i+1}) \mid \mathcal{E}_i] - 2\tau]/k \ .$$

Note that Δ_i is always nonnegative due to the monotonicity of f. Thus,

$$\begin{aligned}\mathbf{E}[\Delta_i] = \ & \Pr[\mathcal{E}_i] \cdot \mathbf{E}[\Delta_i \mid \mathcal{E}_i] + \Pr[\bar{\mathcal{E}}_i] \cdot \mathbf{E}[\Delta_i \mid \bar{\mathcal{E}}_i] \ge \Pr[\mathcal{E}_i] \cdot \mathbf{E}[\Delta_i \mid \mathcal{E}_i] \\ & \ge [\gamma \cdot \Pr[\mathcal{E}_i] \cdot \mathbf{E}[f(\mathrm{OPT}_{k-i+1}) \mid \mathcal{E}_i] - 2\tau]/k \ . \end{aligned} \tag{F.1}$$

It now remains to lower bound the expression $\Pr[\mathcal{E}_i] \cdot \mathbf{E}[f(\mathrm{OPT}_{k-i+1}) \mid \mathcal{E}_i]$ on the rightmost hand side of the last inequality.

$$\begin{aligned}\Pr[\mathcal{E}_i] \cdot \mathbf{E}[f(\mathrm{OPT}_{k-i+1}) \mid \mathcal{E}_i] &= \mathbf{E}[f(\mathrm{OPT}_{k-i+1})] - \Pr[\bar{\mathcal{E}}_i] \cdot \mathbf{E}[f(\mathrm{OPT}_{k-i+1}) \mid \bar{\mathcal{E}}_i] \\ &\ge [1 - (1-\gamma/k)^{k-i+1} - (1 - \Pr[\mathcal{E}_i])] \cdot f(\mathrm{OPT}) \\ &\ge [\Pr[\mathcal{E}] - (1-\gamma/k)^{k-i+1}] \cdot f(\mathrm{OPT}) \ ,\end{aligned}$$

where the first inequality follows from Lemma F.4.1 and the monotonicity of f, and the second inequality holds since $\mathcal{E}$ implies $\mathcal{E}_i$ which means that $\Pr[\mathcal{E}_i] \ge \Pr[\mathcal{E}]$ for every $1 \le i \le k$. □

Proposition 4.5.1 follows quite easily from the last lemma.

Proof of Proposition 4.5.1. Lemma F.4.4 implies, for every $1 \le i \le \lceil k/2 \rceil$,

$$\begin{aligned}\mathbf{E}[\Delta_i] &\ge \frac{\gamma}{k} f(\mathrm{OPT})[\Pr[\mathcal{E}] - (1-\gamma/k)^{k-\lceil k/2\rceil+1}] - \frac{2\tau}{k} \\ &\ge \frac{\gamma}{k} f(\mathrm{OPT})[\Pr[\mathcal{E}] - (1-\gamma/k)^{k/2}] - \frac{2\tau}{k} \\ &\ge \left(\gamma \cdot [\Pr[\mathcal{E}] - e^{-\gamma/2}] \cdot f(\mathrm{OPT}) - 2\tau\right)/k \ .\end{aligned}$$

The definition of Δ_i and the monotonicity of f imply together

$$\mathbf{E}[f(\mathrm{S})] \ge \sum_{i=1}^{b} \mathbf{E}[\Delta_i]$$

for every integer $1 \le b \le k$. In particular, for $b = \lceil k/2 \rceil$, we get

$$\begin{aligned}\mathbf{E}[f(\mathrm{S})] &\ge \frac{b}{k} \cdot \left(\gamma \cdot [\Pr[\mathcal{E}] - e^{-\gamma/2}] \cdot f(\mathrm{OPT}) - 2\tau\right) \\ &\ge \frac{1}{2} \cdot \left(\gamma \cdot [\Pr[\mathcal{E}] - e^{-\gamma/2}] \cdot f(\mathrm{OPT}) - 2\tau\right) \ . \end{aligned}$$ □

F.5 Proof of Theorem 4.5.1

In this section we combine the previous results to prove Theorem 4.5.1. Recall that Observation 4.5.2 and Proposition 4.5.1 give two lower bounds on $\mathbf{E}[f(\mathrm{S})]$ that depend on $\Pr[\mathcal{E}]$. The following lemmata use these lower bounds to derive another lower bound on this quantity which is independent of $\Pr[\mathcal{E}]$. For ease of the reading, we use in this section the shorthand $\gamma' = e^{-\gamma/2}$.

Lemma F.5.1. $\mathbf{E}[f(\mathrm{S})] \ge \frac{\tau}{2a}(3 - \gamma' - 2\sqrt{2-\gamma'}) = \frac{\tau}{a} \cdot \frac{3-e^{-\gamma/2}-2\sqrt{2-e^{-\gamma/2}}}{2}$ *whenever* $\Pr[\mathcal{E}] \ge 2 - \sqrt{2-\gamma'}$.

Proof. By the lower bound given by Proposition 4.5.1,

$$\begin{aligned}
\mathbf{E}[f(\mathrm{S})] &\geq \frac{1}{2} \cdot \{\gamma \cdot [\Pr[\mathcal{E}] - \gamma'] \cdot f(\mathrm{OPT}) - 2\tau\} \\
&\geq \frac{1}{2} \cdot \left\{\gamma \cdot \left[2 - \sqrt{2-\gamma'} - \gamma'\right] \cdot f(\mathrm{OPT}) - 2\tau\right\} \\
&= \frac{1}{2} \cdot \left\{\gamma \cdot \left[2 - \sqrt{2-\gamma'} - \gamma'\right] \cdot f(\mathrm{OPT}) - (\sqrt{2-\gamma'} - 1) \cdot \frac{\tau}{a}\right\} \\
&\geq \frac{\tau}{2a} \cdot \left\{2 - \sqrt{2-\gamma'} - \gamma' - \sqrt{2-\gamma'} + 1\right\} \\
&= \frac{\tau}{a} \cdot \frac{3 - \gamma' - 2\sqrt{2-\gamma'}}{2} \ ,
\end{aligned}$$

where the first equality holds since $a = (\sqrt{2-\gamma'} - 1)/2$, and the last inequality holds since $a\gamma \cdot f(\mathrm{OPT}) \geq \tau$. □

Lemma F.5.2. $\mathbf{E}[f(\mathrm{S})] \geq \frac{\tau}{2a}(3 - \gamma' - 2\sqrt{2-\gamma'}) = \frac{\tau}{a} \cdot \frac{3-e^{-\gamma/2}-2\sqrt{2-e^{-\gamma/2}}}{2}$ *whenever* $\Pr[\mathcal{E}] \leq 2 - \sqrt{2-\gamma'}$.

Proof. By the lower bound given by Observation 4.5.2,

$$\begin{aligned}
\mathbf{E}[f(\mathrm{S})] &\geq (1 - \Pr[\mathcal{E}]) \cdot \tau \geq \left(1 - 2 + \sqrt{2-\gamma'}\right) \cdot \tau \\
&= \left(\sqrt{2-\gamma'} - 1\right) \cdot \frac{\sqrt{2-\gamma'} - 1}{2} \cdot \frac{\tau}{a} = \frac{3 - \gamma' - 2\sqrt{2-\gamma'}}{2} \cdot \frac{\tau}{a} \ .
\end{aligned}$$ □

Combining Lemmata F.5.1 and F.5.2 we get the theorem. □

F.6 Proof of Theorem 4.5.4

There are two cases to consider. If $\gamma < 4/3 \cdot k^{-1}$, then we use the following simple observation.

Observation F.6.1. *The final value of the variable m is* $f^{\max} := \max\{f(u) \mid u \in [N]\} \geq \frac{\gamma}{k} \cdot f(\mathrm{OPT})$.

Proof. The way m is updated by Algorithm 9 guarantees that its final value is $f^{\max}$. To see why the other part of the observation is also true, note that the γ-weak submodularity of f implies

$$\begin{aligned} f^{\max} &\geq \max\{f(u) \mid u \in \mathrm{OPT}\} = f(\emptyset) + \max\{f(u \mid \emptyset) \mid u \in \mathrm{OPT}\} \\ &\geq f(\emptyset) + \frac{1}{k} \sum_{u \in \mathrm{OPT}} f(u \mid \emptyset) \geq f(\emptyset) + \frac{\gamma}{k} f(\mathrm{OPT} \mid \emptyset) \geq \frac{\gamma}{k} \cdot f(\mathrm{OPT}) \ . \end{aligned}$$ □

By Observation F.6.1, the value of the solution produced by Streak is at least

$$\begin{aligned} f(u_m) = m &\geq \frac{\gamma}{k} \cdot f(\mathrm{OPT}) \geq \frac{3\gamma^2}{4} \cdot f(\mathrm{OPT}) \\ &\geq (1-\epsilon)\gamma \cdot \frac{3(\gamma/2)}{2} \cdot f(\mathrm{OPT}) \\ &\geq (1-\epsilon)\gamma \cdot \frac{3 - 3e^{-\gamma/2}}{2} \cdot f(\mathrm{OPT}) \\ &\geq (1-\epsilon)\gamma \cdot \frac{3 - e^{-\gamma/2} - 2\sqrt{2 - e^{-\gamma/2}}}{2} \cdot f(\mathrm{OPT}) \ , \end{aligned}$$

where the second to last inequality holds since $1 - \gamma/2 \leq e^{-\gamma/2}$, and the last inequality holds since $e^{-\gamma} + e^{-\gamma/2} \leq 2$.

It remains to consider the case $\gamma \geq 4/3 \cdot k^{-1}$, which has a somewhat more involved proof. Observe that the approximation ratio of Streak is 1 whenever $f(\mathrm{OPT}) = 0$ because the value of any set, including the output set of the algorithm, is nonnegative. Thus, we can safely assume in the rest of the analysis of the approximation ratio of Algorithm 9 that $f(\mathrm{OPT}) > 0$.

Let τ^* be the maximal value in the set $\{(1-\epsilon)^i \mid i \in \mathbf{Z}\}$ which is not larger than $a\gamma \cdot f(\mathrm{OPT})$. Note that τ^* exists by our assumption that $f(\mathrm{OPT}) > 0$. Moreover, we also have $(1-\epsilon) \cdot a\gamma \cdot f(\mathrm{OPT}) < \tau^* \leq a\gamma \cdot f(\mathrm{OPT})$. The following lemma

gives an interesting property of τ^*. To understand the lemma, it is important to note that the set of values for τ in the instances of Algorithm 8 appearing in the final collection I is deterministic because the final value of m is always $f^{\max}$.

Lemma F.6.1. *If there is an instance of Algorithm 8 with $\tau = \tau^*$ in I when* Streak *terminates, then in expectation* Streak *has an approximation ratio of at least*

$$(1-\epsilon)\gamma \cdot \frac{3 - e^{-\gamma/2} - 2\sqrt{2 - e^{-\gamma/2}}}{2} \quad .$$

Proof. Consider a value of τ for which there is an instance of Algorithm 8 in I when Algorithm 9 terminates, and consider the moment that Algorithm 9 created this instance. Since the instance was not created earlier, we get that m was smaller than τ/k before this point. In other words, the marginal contribution of every element that appeared before this point to the empty set was less than τ/k. Thus, even if the instance had been created earlier it would not have taken any previous elements.

An important corollary of the above observation is that the output of every instance of Algorithm 8 that appears in I when Streak terminates is equal to the output it would have had if it had been executed on the entire input stream from its beginning (rather than just from the point in which it was created). Since we assume that there is an instance of Algorithm 8 with $\tau = \tau^*$ in the final collection I, we get by Theorem 4.5.1 that the expected value of the output of this instance is at least

$$\frac{\tau^*}{a} \cdot \frac{3 - e^{-\gamma/2} - 2\sqrt{2 - e^{-\gamma/2}}}{2} > (1-\epsilon)\gamma \cdot f(\mathrm{OPT}) \cdot \frac{3 - e^{-\gamma/2} - 2\sqrt{2 - e^{-\gamma/2}}}{2} \quad .$$

The lemma now follows since the output of STREAK is always at least as good as the output of each one of the instances of Algorithm 8 in its collection I. □

We complement the last lemma with the next one.

Lemma F.6.2. *If $\gamma \geq 4/3 \cdot k^{-1}$, then there is an instance of Algorithm 8 with $\tau = \tau^*$ in I when STREAK terminates.*

Proof. We begin by bounding the final value of m. By Observation F.6.1 this final value is $f^{\max} \geq \frac{\gamma}{k} \cdot f(\mathrm{OPT})$. On the other hand, $f(u) \leq f(\mathrm{OPT})$ for every element $u \in [N]$ since $\{u\}$ is a possible candidate to be OPT, which implies $f^{\max} \leq f(\mathrm{OPT})$. Thus, the final collection I contains an instance of Algorithm 8 for every value of τ within the set

$$\begin{aligned} &\left\{(1-\epsilon)^i \mid i \in \mathbf{Z} \quad \text{and} \quad (1-\epsilon) \cdot f^{\max}/(9k^2) \leq (1-\epsilon)^i \leq f^{\max} \cdot k\right\} \\ \supseteq &\left\{(1-\epsilon)^i \mid i \in \mathbf{Z} \quad \text{and} \quad (1-\epsilon) \cdot f(\mathrm{OPT})/(9k^2) \leq (1-\epsilon)^i \leq \gamma \cdot f(\mathrm{OPT})\right\}. \end{aligned}$$

To see that τ^* belongs to the last set, we need to verify that it obeys the two inequalities defining this set. On the one hand, $a = (\sqrt{2 - e^{-\gamma/2}} - 1)/2 < 1$ implies

$$\tau^* \leq a\gamma \cdot f(\mathrm{OPT}) \leq \gamma \cdot f(\mathrm{OPT}) \ .$$

On the other hand, $\gamma \geq 4/3 \cdot k^{-1}$ and $1 - e^{-\gamma/2} \geq \gamma/2 - \gamma^2/8$ imply

$$\begin{aligned} \tau^* &> (1-\epsilon) \cdot a\gamma \cdot f(\mathrm{OPT}) = (1-\epsilon) \cdot (\sqrt{2 - e^{-\gamma/2}} - 1) \cdot \gamma \cdot f(\mathrm{OPT})/2 \\ &\geq (1-\epsilon) \cdot (\sqrt{1 + \gamma/2 - \gamma^2/8} - 1) \cdot \gamma \cdot f(\mathrm{OPT})/2 \\ &\geq (1-\epsilon) \cdot (\sqrt{1 + \gamma/4 + \gamma^2/64} - 1) \cdot \gamma \cdot f(\mathrm{OPT})/2 \end{aligned}$$

$$= (1-\epsilon)\cdot(\sqrt{(1+\gamma/8)^2}-1)\cdot\gamma\cdot f(\mathrm{OPT})/2 \geq (1-\epsilon)\cdot\gamma^2\cdot f(\mathrm{OPT})/16$$
$$\geq (1-\epsilon)\cdot f(\mathrm{OPT})/(9k^2) \ . \qquad \square$$

Combining Lemmata F.6.1 and F.6.2 we get the desired guarantee on the approximation ratio of Streak. $\square$

F.7 Proof of Theorem 4.5.5

Observe that Streak keeps only one element (u_m) in addition to the elements maintained by the instances of Algorithm 8 in I. Moreover, Algorithm 8 keeps at any given time at most $O(k)$ elements since the set S it maintains can never contain more than k elements. Thus, it is enough to show that the collection I contains at every given time at most $O(\epsilon^{-1}\log k)$ instances of Algorithm 8. If $m = 0$ then this is trivial since $I = \emptyset$. Thus, it is enough to consider the case $m > 0$. Note that in this case

$$|I| \leq 1 - \log_{1-\epsilon}\frac{mk}{(1-\epsilon)m/(9k^2)} = 2 - \frac{\ln(9k^3)}{\ln(1-\epsilon)}$$
$$= 2 - \frac{\ln 9 + 3\ln k}{\ln(1-\epsilon)} = 2 - \frac{O(\ln k)}{\ln(1-\epsilon)} \ .$$

We now need to upper bound $\ln(1-\epsilon)$. Recall that $1-\epsilon \leq e^{-\epsilon}$. Thus, $\ln(1-\epsilon) \leq -\epsilon$. Plugging this into the previous inequality gives

$$|I| \leq 2 - \frac{O(\ln k)}{-\epsilon} = 2 + O(\epsilon^{-1}\ln k) = O(\epsilon^{-1}\ln k) \ . \qquad \square$$

F.8 Proof of Theorems 4.5.6–4.5.8

We present proofs for Theorems 4.5.6 and 4.5.7. Given the latter, the proof of Theorem 4.5.8 is similar to that of Theorem 4.5.4.

F.8.1 Proof of Theorem 4.5.6

This bound uses the *inverse curvature* parameter from Definition C.1.9, recently defined in [108].

Proof. Assume γ, $\check{\alpha}$, and $f(\mathrm{OPT})$ are all known, and let S be the set returned by ThresholdGreedy. Let S_i be the set already selected by the algorithm when element i arrives in the stream. We set $\tau = \frac{f(\mathrm{OPT})\gamma(1-\check{\alpha})}{1+\gamma(1-\check{\alpha})}$. First consider the case $|\mathrm{S}| = k$. Each selected element has marginal gain greater than τ/k. This implies

$$f(\mathrm{S}) = \sum_{i \in \mathrm{S}} f(i \mid \mathrm{S}_i) \geq \tau = f(\mathrm{OPT}) \frac{\gamma(1-\check{\alpha})}{1+\gamma(1-\check{\alpha})} \quad .$$

Next consider the case $|\mathrm{S}| < k$. Let $\mathrm{S}^R = \mathrm{OPT} \setminus \mathrm{S}$.

$$f(\mathrm{OPT}) - f(\mathrm{S}) = f(\mathrm{S}^R \cup \mathrm{S}) - f(\mathrm{S}) \leq \frac{1}{\gamma} \sum_{i \in \mathrm{S}^R} f(i \mid \mathrm{S}) \leq \frac{1}{\gamma(1-\check{\alpha})} \sum_{i \in \mathrm{S}^R} f(i \mid \mathrm{S}_i) \quad .$$

Since these elements are not selected by the algorithm, we know that each term in the summation is less than τ/k.

$$\begin{aligned} f(\mathrm{OPT}) - f(\mathrm{S}) &\leq \frac{|\mathrm{S}|^R}{\gamma(1-\check{\alpha})} \frac{\tau}{k} \leq \frac{|\mathrm{S}|^R}{k\gamma(1-\check{\alpha})} \frac{f(\mathrm{OPT})\gamma(1-\check{\alpha})}{1+\gamma(1-\check{\alpha})} \\ \Rightarrow f(\mathrm{S}) &\geq f(\mathrm{OPT}) \left[1 - \frac{1}{1+\gamma(1-\check{\alpha})}\right] = f(\mathrm{OPT}) \left[\frac{\gamma(1-\check{\alpha})}{1+\gamma(1-\check{\alpha})}\right] \quad . \end{aligned}$$

□

F.8.2 Proof of Theorem 4.5.7

We will use the following lemma from [84] and an inequality for approximating a binomial.

Lemma F.8.1 (adapted from Lemma 2.3 in [84]). *For every* $0 \le i \le k$,

$$\mathbb{E}\left[f(\mathsf{OPT}_i)\right] \ge \left[1 - \left(\frac{k-i+1}{k+1}\right)^{\gamma}\right] \cdot f(\mathsf{OPT}) \ .$$

Proposition F.8.1 (Bernoulli's Inequality). *For* $0 \le \gamma \le 1$ *and* $x \ge -1$,

$$(1+x)^{\gamma} \le 1 + \gamma x \ .$$

First we will use the above ingredients to get a lower bound on $\mathbf{E}[\Delta_i]$.

Lemma F.8.2. *For every* $1 \le i \le k$,

$$\mathbf{E}[\Delta_i] \ge \frac{\gamma \cdot \left[\Pr[\mathcal{E}] - \left(\frac{i}{k+1}\right)^{\gamma}\right] \cdot f(\mathsf{OPT}) - 2\tau}{k} \ .$$

Proof. We begin with (F.1) from the proof of Lemma F.4.4, and lower bound the expression $\Pr[\mathcal{E}_i] \cdot \mathbf{E}[f(\mathsf{OPT}_{k-i+1}) \mid \mathcal{E}_i]$ as follows.

$$\begin{aligned}
&\Pr[\mathcal{E}_i] \cdot \mathbf{E}[f(\mathsf{OPT}_{k-i+1}) \mid \mathcal{E}_i] \\
&= \mathbf{E}[f(\mathsf{OPT}_{k-i+1})] - \Pr[\bar{\mathcal{E}}_i] \cdot \mathbf{E}[f(\mathsf{OPT}_{k-i+1}) \mid \bar{\mathcal{E}}_i] \\
&\ge \left[1 - \left(\frac{k+1-(k+1-i)}{k+1}\right)^{\gamma} - (1 - \Pr[\mathcal{E}_i])\right] \cdot f(\mathsf{OPT}) \\
&\ge \left[\Pr[\mathcal{E}] - \left(\frac{i}{k+1}\right)^{\gamma}\right] \cdot f(\mathsf{OPT}) \ ,
\end{aligned}$$

where the first inequality follows from Lemma F.8.1 and the monotonicity of f, and the second inequality holds since $\mathcal{E}$ implies $\mathcal{E}_i$ which means that $\Pr[\mathcal{E}_i] \ge \Pr[\mathcal{E}]$ for $1 \le i \le k$. □

Proposition F.8.2 follows from the last lemma. Then we use this result to prove Theorem 4.5.7. Theorem 4.5.8 is proved similarly to Theorem 4.5.4, with Theorem 4.5.7 in place of Theorem 4.5.1.

Proposition F.8.2. *For the set* S *produced by* ThresholdGreedy,

$$\mathbf{E}[f(\mathrm{S})] \geq \gamma \cdot f(\mathrm{OPT}) \left[\Pr[\mathcal{E}] + \frac{\gamma}{2} - 1\right] - 2\tau \quad .$$

Proof. Lemma F.8.2 and Proposition F.8.1 imply for every $1 \leq i \leq k$,

$$\mathbf{E}[\Delta_i] \geq \frac{\gamma}{k} f(\mathrm{OPT}) \left[\Pr[\mathcal{E}] - \left(1 - \gamma + \gamma \cdot \frac{i}{k+1}\right)\right] - \frac{2\tau}{k} \quad .$$

The definition of Δ_i and the monotonicity of f imply together

$$\begin{aligned}
\mathbf{E}[f(\mathrm{S})] &\geq \sum_{i=1}^{k} \mathbf{E}[\Delta_i] \\
&\geq \gamma \cdot f(\mathrm{OPT}) \Pr[\mathcal{E}] - \frac{\gamma \cdot f(\mathrm{OPT})}{k} \sum_{i=1}^{k} \left(1 - \gamma + \gamma \cdot \frac{i}{k+1}\right) - 2\tau \\
&= \gamma \cdot f(\mathrm{OPT}) \left[\Pr[\mathcal{E}] + \frac{\gamma}{2} - 1\right] - 2\tau \quad . \qquad \square
\end{aligned}$$

Now to prove Theorem 4.5.7, we combine Observation 4.5.2 and Proposition F.8.2. In this section, we let $a = \sqrt{\gamma'/2 + 1} - 1 = \frac{\sqrt{2\gamma'+4}-2}{2}$ and $\gamma' = \gamma/2$.

Lemma F.8.3. $\mathbf{E}[f(\mathrm{S})] \geq \frac{\tau}{a} \cdot \left(4 + \gamma' - 2\sqrt{2\gamma' + 4}\right) = \frac{\tau}{a} \cdot \left(4 + \gamma/2 - 2\sqrt{\gamma + 4}\right)$ *whenever* $\Pr[\mathcal{E}] \geq 3 - \sqrt{2\gamma' + 4}$.

Proof. By the lower bound given by Proposition F.8.2,

$$\mathbf{E}[f(\mathrm{S})] \geq \gamma \cdot f(\mathrm{OPT}) \left[\Pr[\mathcal{E}] + \gamma' - 1\right] - 2\tau$$

$$
\begin{aligned}
&\geq \gamma \cdot f(\mathrm{OPT}) \left[2 - \sqrt{2\gamma' + 4} + \gamma'\right] - 2\tau \\
&= \gamma \cdot f(\mathrm{OPT}) \left[2 - \sqrt{2\gamma' + 4} + \gamma'\right] - \frac{2\tau}{a}\left(\sqrt{\gamma'/2 + 1} - 1\right) \\
&\geq \frac{\tau}{a} \cdot \left(2 - \sqrt{2\gamma' + 4} + \gamma' - 2\sqrt{\gamma'/2 + 1} + 2\right) \\
&= \frac{\tau}{a} \cdot \left(4 + \gamma' - 2\sqrt{2\gamma' + 4}\right) \quad ,
\end{aligned}
$$

where the first equality holds since $a = (\sqrt{2\gamma' + 4} - 2)/2$, and the last inequality holds since $a\gamma \cdot f(\mathrm{OPT}) \geq \tau$. □

Lemma F.8.4. $\mathbf{E}[f(S)] \geq \frac{\tau}{a} \cdot \left(4 + \gamma' - 2\sqrt{2\gamma' + 4}\right) = \frac{\tau}{a} \cdot \left(4 + \gamma/2 - 2\sqrt{\gamma + 4}\right)$ *whenever* $\Pr[\mathcal{E}] \leq 3 - \sqrt{2\gamma' + 4}$.

Proof. By the lower bound given by Observation 4.5.2,

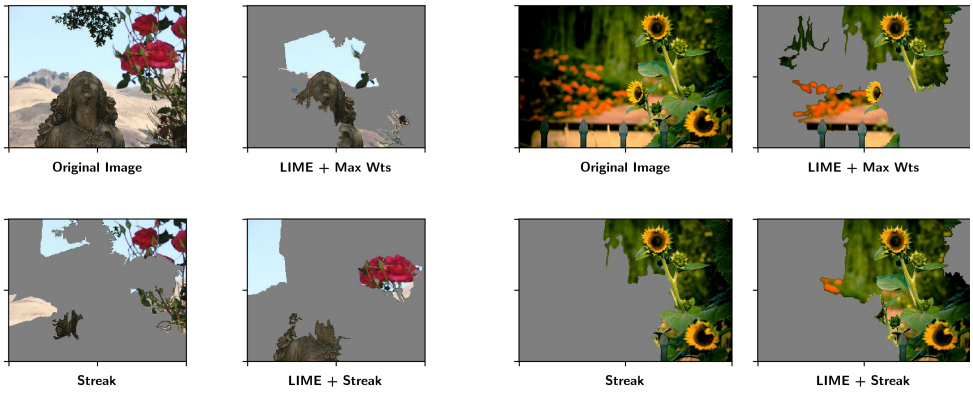
$$
\begin{aligned}
\mathbf{E}[f(S)] &\geq (1 - \Pr[\mathcal{E}]) \cdot \tau \geq \left(1 - 3 + \sqrt{2\gamma' + 4}\right) \cdot \tau \\
&= \left(\sqrt{2\gamma' + 4} - 2\right) \cdot \frac{\sqrt{2\gamma' + 4} - 2}{2} \cdot \frac{\tau}{a} = \frac{2\gamma' + 8 - 4\sqrt{2\gamma' + 4}}{2} \cdot \frac{\tau}{a} \quad .
\end{aligned}
$$

□

Combining Lemmata F.8.3 and F.8.4 we get the theorem. □

Appendix G

Additional Images



(a) (b)

Figure G.1: In addition[2] to the experiment in Section 4.6.2, we also replaced LIME's default feature selection algorithms with STREAK and then fit the same sparse regression on the selected superpixels. This method is captioned "LIME + Streak." Since LIME fits a series of nested regression models, the corresponding set function is guaranteed to be monotone, but is not necessarily submodular. We see that results look qualitatively similar and are in some instances better than the default methods. However, the running time of this approach is similar to the other LIME algorithms.

[2]The material in this appendix is based on the following conference paper: [116] E. R. Elenberg, A. G. Dimakis, M. Feldman, and A. Karbasi. Streaming Weak Submodularity: Interpreting Neural Networks on the Fly. In *NIPS*, pages 4047–4057, 2017. It is the dissertation author's contribution.

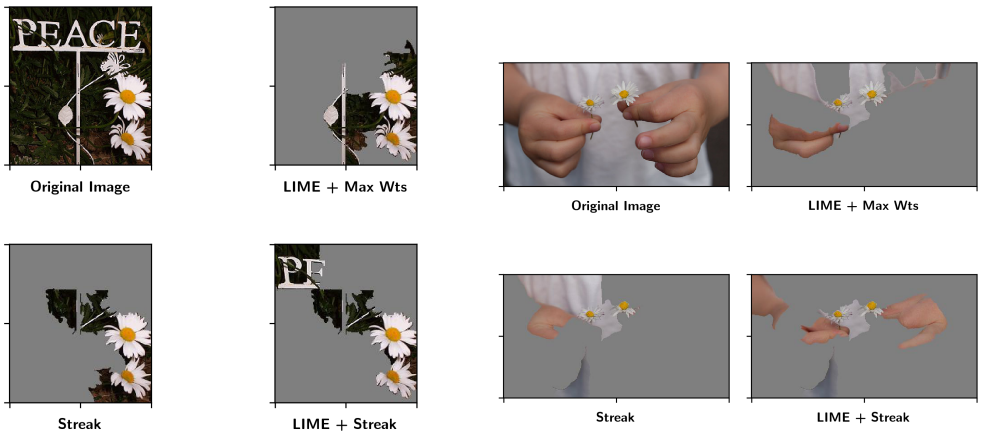
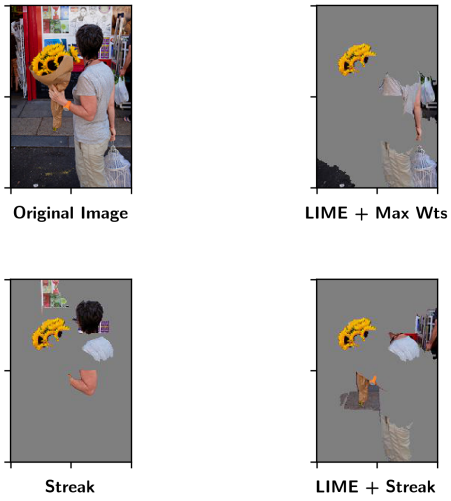
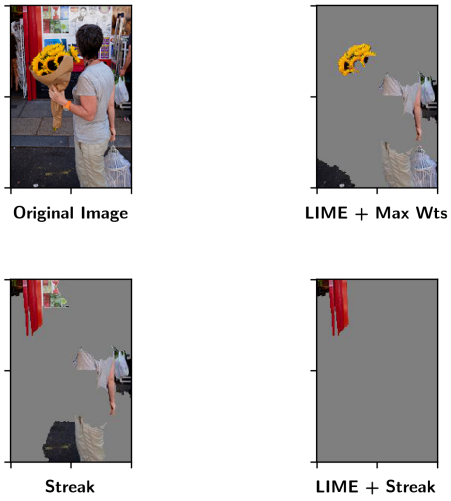


Figure G.2: Here we used the same setup described in Figure G.1.



(a)



(b)

Figure G.3: Here we used the same setup described in Figure G.1, but compared explanations for predicting 2 different classes for the same base image: (a) – The highest likelihood label (sunflower). (b) – The second-highest likelihood label (rose). All algorithms perform similarly for the sunflower label, but our algorithms identify the most rose-like parts of the image.

Bibliography

[1] Derek O'Callaghan, Martin Harrigan, Joe Carthy, and Pádraig Cunningham. Identifying Discriminating Network Motifs in YouTube Spam. http://arxiv.org/abs/1202.5216, 2012. 3, 13

[2] Nataša Pržulj. Biological Network Comparison Using Graphlet Degree Distribution. *Bioinformatics*, 23(2):177–183, 2007. 3, 13, 21, 30

[3] Austin R. Benson, David F. Gleich, and Jure Leskovec. Higher-order organization of complex networks. *Nature*, 356(6295):163–166, 2016. 3, 13

[4] Christian Borgs, Jennifer Chayes, and Katalin Vesztergombi. Counting Graph Homomorphisms. *Topics in Discrete Mathematics*, pages 315–371, 2006. 3, 13

[5] László Lovász. *Large Networks and Graph Limits*, volume 60. American Mathematical Soc., 2012. 3, 13

[6] P-A. G. Maugis, C. E. Priebe, S. C. Olhede, and P. J. Wolfe. Statistical Inference for Network Samples Using Subgraph Counts. https://arxiv.org/abs/1701.00505, 2017. 3

[7] Olaf Sporns, Christopher J. Honey, and Rolf Kötter. Identification and Classification of Hubs in Brain Networks. *PLoS ONE*, 2(10), 2007. 3

[8] Rahmtin Rotabi, Krishna Kamath, Jon Kleinberg, and Aneesh Sharma. Detecting Strong Ties Using Network Motifs. In *WWW*, pages 983–992, 2017. 3

[9] Changping Meng, S Chandra Mouli, Bruno Ribeiro, and Jennifer Neville. Subgraph Pattern Neural Networks for High-Order Graph Evolution Prediction. In *AAAI*, 2018. 3, 55

[10] Nino Shervashidze, Kurt Mehlhorn, and Tobias H Petri. Efficient graphlet kernels for large graph comparison. In *AISTATS*, pages 488–495, 2009. 3, 21

[11] Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 17–30, 2012. 3, 16, 18, 20, 29, 47, 52

[12] Joseph E. Gonzalez, Reynold S. Xin, Ankur Dave, Daniel Crankshaw, Michael J. Franklin, and Ion Stoica. GraphX: Graph Processing in a Distributed Dataflow Framework. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 599–613, 2014. 3

[13] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: A System for Large-Scale Graph Processing. In *SIGMOD*, pages 135–145, 2010. 3

[14] Vaggos Chatziafratis, Tim Roughgarden, and Jan Vondrak. Stability and Recovery for Independence Systems. In *25th Annual European Symposium*

on Algorithms (ESA 2017), pages 26:1–26:15, 2017. 5, 110

[15] George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An Analysis of Approximations for Maximizing Submodular Set Functions - I. *Mathematical Programming*, 14(1):265–294, 1978. 5, 8, 72, 74, 78, 93, 96, 137, 149

[16] Abhimanyu Das and David Kempe. Submodular meets Spectral: Greedy Algorithms for Subset Selection, Sparse Approximation and Dictionary Selection. In *ICML*, pages 1057–1064, 2011. 6, 67, 70, 72, 74, 75, 77, 80, 81, 82, 84, 85, 90, 97, 98, 135, 136, 151

[17] Niv Buchbinder, Moran Feldman, and Roy Schwartz. Online Submodular Maximization with Preemption. In *SODA*, pages 1202–1216, 2015. 6, 72, 94, 96

[18] Andrew An Bian, Baharan Mirzasoleiman, Joachim M. Buhmann, and Andreas Krause. Guaranteed Non-convex Optimization: Submodular Maximization over Continuous Domains. In *AISTATS*, pages 111–120, 2017. 6, 73, 97

[19] Xinghao Pan, Stefanie Jegelka, Joseph E. Gonzalez, Joseph K. Bradley, and Michael I. Jordan. Parallel Double Greedy Submodular Maximization. In *NIPS*, pages 118–126, 2014. 6, 72, 96

[20] Sandra Wachter, Brent Mittelstadt, and Luciano Florid. Why a Right to Explanation of Automated Decision-Making Does Not Exist in the General

Data Protection Regulation. *International Data Privacy Law*, 7:76–99, 2017. 7

[21] Andrew D. Selbst and Julia Powles. Meaningful information and the right to explanation. *International Data Privacy Law*, 7:233–242, 2017. 7

[22] Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming Submodular Maximization: Massive Data Summarization on the Fly. In *KDD*, pages 671–680, 2014. 8, 72, 94, 96, 105

[23] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. In *CVPR*, pages 2818–2826, 2016. 9, 95, 108

[24] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?" Explaining the Predictions of Any Classifier. In *KDD*, pages 1135–1144, 2016. 9, 95, 97, 108, 113

[25] Ethan R. Elenberg, Karthikeyan Shanmugam, Michael Borokhovich, and Alexandros G. Dimakis. Beyond Triangles: A Distributed Framework for Estimating 3-profiles of Large Graphs. In *KDD*, pages 229–238, 2015. 12, 13, 43, 115

[26] Ethan R. Elenberg, Karthikeyan Shanmugam, Michael Borokhovich, and Alexandros G. Dimakis. Distributed Estimation of Graph 4-profiles. In *WWW*, pages 483–493, 2016. 12, 13, 22, 115, 127

[27] Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. Efficient Semi-Streaming Algorithms for Local Triangle Counting in Massive Graphs. In *KDD*, 2008. 13, 20

[28] Johan Ugander, Lars Backstrom, Menlo Park, and Jon Kleinberg. Subgraph Frequencies: Mapping the Empirical and Extremal Geography of Large Graph Collections. In *WWW*, pages 1307–1318, 2013. 13, 15, 20, 26

[29] Virginia Vassilevska Williams, Josh Wang, Ryan Williams, and Huacheng Yu. Finding Four-Node Subgraphs in Triangle Time. *SODA*, pages 1671–1680, 2014. 13, 21, 30

[30] Tomaž Hočevar and Janez Demšar. A Combinatorial Approach to Graphlet Counting. *Bioinformatics*, 30(4):559–65, February 2014. 13, 18, 21, 22, 30, 52, 53

[31] Ali Pinar, C. Seshadri, and V. Vishal. ESCAPE: Efficiently Counting All 5-Vertex Subgraphs. In *WWW*, pages 1431–1440, 2017. 13, 55

[32] Nesreen K. Ahmed, Ted Willke, and Ryan A. Rossi. Estimation of Local Subgraph Counts. In *IEEE International Conference on Big Data*, pages 1–10, 2016.

[33] Charalampos E. Tsourakakis, U. Kang, Gary L. Miller, and Christos Faloutsos. DOULION: Counting Triangles in Massive Graphs with a Coin. In *SIGKDD*, 2009. 17, 19

[34] Charalampos E. Tsourakakis, Mihail Kolountzakis, and Gary L. Miller. Triangle Sparsifiers. *Journal of Graph Theory and Applications*, 15(6):703–726, 2011. 17, 19, 36, 41, 42, 43, 45

[35] Jeong Han Kim and Van H. Vu. Concentration of Multivariate Polynomials and Its Applications. *Combinatorica*, 20(3):417–434, 2000. 17, 19, 36, 40, 41, 42, 43, 45, 54, 125

[36] Dmitry Gavinsky, Shachar Lovett, Michael Saks, and Srikanth Srinivasan. A Tail Bound for Read-k Families of Functions. *Random Structures & Algorithms*, 47(1):99–108, 2015. 17, 19, 36, 42, 43, 44

[37] Charalampos E. Tsourakakis. Fast Counting of Triangles in Large Real Networks: Algorithms and Laws. In *IEEE International Conference on Data Mining*, 2008. 19, 21

[38] Nesreen K Ahmed, Nick Duffield, Jennifer Neville, and Ramana Kompella. Graph Sample and Hold : A Framework for Big-Graph Analytics Categories and Subject Descriptors. In *KDD*, 2014. 19

[39] C. Seshadhri, Ali Pinar, and Tamara G. Kolda. Triadic measures on graphs: The power of wedge sampling. In *Proc. SDM*, pages 10–18, 2013. 19

[40] T. Eden, A. Levi, D. Ron, and C. Seshadhri. Approximately Counting Triangles in Sublinear Time. In *FOCS*, pages 614–633, 2015. 19

[41] Mansurul A. Bhuiyan, Mahmudur Rahman, Mahmuda Rahman, and Mohammad Al Hasan. GUISE: Uniform Sampling of Graphlets for Large Graph

Analysis. *IEEE 12th International Conference on Data Mining*, pages 91–100, December 2012. 19

[42] Madhav Jha, C Seshadhri, and Ali Pinar. Path Sampling: A Fast and Provable Method for Estimating 4-Vertex Subgraph Counts. In *WWW*, pages 495–505, 2015. 19, 54

[43] Svante Janson, Krzysztof Oleszkiewicz, and Andrzej Ruciński. Upper Tails for Subgraph Counts in Random Graphs. *Israel Journal of Mathematics*, 142(1):61–92, 2004. 19

[44] Nadathur Satish, Narayanan Sundaram, Mostofa Ali Patwary, Jiwon Seo, Jongsoo Park, M Amber Hassaan, Shubho Sengupta, Zhaoming Yin, and Pradeep Dubey. Navigating the Maze of Graph Analytics Frameworks using Massive Graph Datasets. In *SIGMOD*, pages 979–990, 2014. 20

[45] Thomas Schank. *Algorithmic Aspects of Triangle-Based Network Analysis*. PhD thesis, 2007. 20

[46] Shumo Chu and James Cheng. Triangle listing in massive networks and its applications. In *Proc. 17th ACM SIGKDD*, page 672, New York, New York, USA, 2011. ACM Press. 20

[47] Rasmus Pagh and Charalampos E. Tsourakakis. Colorful triangle counting and a MapReduce implementation. *Information Processing Letters*, 112(7):277–281, 2012. 20

[48] Siddharth Suri and Sergei Vassilvitskii. Counting Triangles and the Curse of the Last Reducer. In *Proc. 20th International World Wide Web Conference*, page 607. ACM Press, 2011. 20

[49] Xifeng Yan and Jiawei Han. gSpan: Graph-Based Substructure Pattern Mining. In *International Conference on Data Mining*, 2002. 20

[50] Jinsoo Lee, Wook-Shin Han, Romans Kasperovics, and Jeong-Hoon Lee. An In-depth Comparison of Subgraph Isomorphism Algorithms in Graph Databases. *Proc. VLDB Endowment*, 6(2):133–144, December 2012. 20

[51] WS Han and JH Lee. Turbo$_{iso}$: Towards Ultrafast and Robust Subgraph Isomorphism Search in Large Graph Databases. In *SIGMOD*, pages 337–348, 2013. 20

[52] Fei Fei, Biao Jie, and Daoqiang Zhang. Frequent and Discriminative Subnetwork Mining for Mild Cognitive Impairment Classification. *Brain Connectivity*, 4(5):347–360, June 2014. 21

[53] Fereydoun Hormozdiari, Petra Berenbrink, Natasa Przulj, and S Cenk Sahinalp. Not All Scale-free Networks are Born Equal: The Role of the Seed Graph in PPI Network Evolution. *PLoS Computational Biology*, 3(7):e118, July 2007. 21

[54] L. R. Varshney, B. L. Chen, E. Paniagua, D. H. Hall, and D. B. Chklovskii. Structural Properties of the Caenorhabditis elegans Neuronal Network. *PLoS Computional Biology*, 7(2), 2011. 21

[55] N. Alon, R. Yuster, and U. Zwick. Finding and Counting Given Length Cycles. *Algorithmica*, 17(3):209–223, March 1997. 21

[56] Ton Kloks, Dieter Kratsch, and Haiko Müller. Finding and Counting Small Induced Subgraphs Efficiently. *Information Processing Letters*, 74(3-4):115–121, 2000. 21

[57] Miros law Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Counting and Detecting Small Subgraphs via Equations. *SIAM Journal of Discrete Mathematics*, 27(2):892–909, 2013. 21, 30

[58] D. Marcus and Y. Shavitt. RAGE - A Rapid Graphlet Enumerator for Large Networks. *Computer Networks*, 56(2):810–819, February 2012. 21

[59] Nesreen K. Ahmed, Jennifer Neville, Ryan A. Rossi, and Nick Duffield. Efficient Graphlet Counting for Large Networks. In *IEEE International Conference on Data Mining*, 2015. 21, 22, 30

[60] Aleksandar Milinković, Stevan Milinković, and Ljubomir Lazić. A Contribution to Acceleration of Graphlet Counting. In *Infoteh Jahorina Symposium*, volume 14, pages 741–745, 2015. 22, 53

[61] Noga Alon, Yossi Matias, and Mario Szegedy. The Space Complexity of Approximating the Frequency Moments. In *STOC*, pages 20–29, 1996. 35

[62] Amazon Web Services. http://aws.amazon.com, 2015. 47

[63] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is Twitter, a Social Network or a News Media? In *Proc. 19th International World Wide Web Conference*, pages 591–600, 2010. 48

[64] Robert Meusel, Sebastiano Vigna, Oliver Lehmberg, and Christian Bizer. Graph structure in the web – revisited. In *Proc. 23rd International World Wide Web Conference, Web Science Track*. ACM, 2014. 48

[65] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data, June 2014. 48

[66] T. A. Davis and Y. Hu. The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software*, 38(1):1–25, 2011. 48

[67] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In *NIPS*, pages 2224–2232, 2015. 55

[68] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*, 2017. 55

[69] Aditya Grover and Jure Leskovec. node2vec: Scalable Feature Learning for Graphs. In *KDD*, pages 855–864, 2016. 55

[70] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. GraphGAN: Graph Representation Learning with Generative Adversarial Nets. In *AAAI*, 2018. 55

[71] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. NetGAN: Generating Graphs via Random Walks. http://arxiv.org/abs/1803.00816, 2018. 55

[72] Ethan R. Elenberg, Rajiv Khanna, Alexandros G. Dimakis, and Sahand Negahban. Restricted Strong Convexity Implies Weak Submodularity. In *NIPS Workshop on Learning in High Dimensions with Structure*, 2016. 66, 94, 97

[73] Ethan R. Elenberg, Rajiv Khanna, Alexandros G. Dimakis, and Sahand Negahban. Restricted Strong Convexity Implies Weak Submodularity. *The Annals of Statistics*, 2018 (to appear). 66, 80, 94, 97, 106, 135, 143, 145

[74] Sahand Negahban, Pradeep Ravikumar, Bin Yu, and Martin J. Wainwright. A Unified Framework for High-Dimensional Analysis of M-Estimators with Decomposable Regularizers. *Statistical Science*, 27(4), 2012. 67, 68, 75, 76, 90, 154

[75] Balas Kausik Natarajan. Sparse Approximate Solutions to Linear Systems. *SIAM Journal on Computing*, 24(2):227–234, 1995. 67

[76] Steven C. H. Hoi, Rong Jin, Jianke Zhu, and Michael R. Lyu. Batch Mode Active Learning and its Application to Medical Image Classification. In *ICML*, pages 417–424, 2006. 67, 72, 96

[77] Kai Wei, Yuzong Liu, Katrin Kirchhoff, and Jeff Bilmes. Using Document Summarization Techniques for Speech Data Subset Selection. In *NAACL-HLT*, pages 721–726, 2013. 67

[78] Kai Wei, Iyer Rishabh, and Jeff Bilmes. Submodularity in Data Subset Selection and Active Learning. In *ICML*, pages 1954–1963, 2015. 67, 72, 96

[79] Francis R. Bach. Learning with Submodular Functions: A Convex Optimization Perspective. *Foundations and Trends in Machine Learning*, 6, 2013. 67, 73, 74, 96

[80] Andreas Krause and Volkan Cevher. Submodular Dictionary Selection for Sparse Representation. In *ICML*, pages 567–574, 2010. 67, 97

[81] Rajiv Khanna, Ethan R. Elenberg, Alexandros G. Dimakis, Sahand Negahban, and Joydeep Ghosh. Scalable Greedy Feature Selection via Weak Submodularity. In *AISTATS*, pages 1560–1568, 2017. 70, 72, 74, 96, 97, 138

[82] Rajiv Khanna, Ethan R. Elenberg, Alexandros G. Dimakis, Joydeep Ghosh, and Sahand Negahban. On Approximation Guarantees for Greedy Low Rank Optimization. In *ICML*, pages 1837–1846, 2017. 70, 97, 151

[83] Andrew An Bian, Joachim M. Buhmann, Andreas Krause, and Sebastian Tschiatschek. Guarantees for Greedy Maximization of Non-submodular Functions with Applications. In *ICML*, pages 498–507, 2017. 70, 72, 75, 135, 136, 137, 141

[84] Lin Chen, Moran Feldman, and Amin Karbasi. Weakly Submodular Maximization Beyond Cardinality Constraints: Does Randomization Help Greedy? https://arxiv.org/abs/1707.04347, 2017. 70, 73, 106, 170

[85] Rahul Mazumder, Robert Freund, and Paul Grigas. A New Perspective on Boosting in Linear Regression via Subgradient Optimization and Relatives. *The Annals of Statistics*, 45(6):2328–2364, 2017. 71

[86] Deanna Needell and Joel A Tropp. CoSaMP : Iterative Signal Recovery from Incomplete and Inaccurate Samples. *Applied and Computational Harmonic Analysis*, 3(26):301–321, 2009. 71, 72

[87] Ali Jalali, Chris Johnson, and Pradeep Ravikumar. On Learning Discrete Graphical Models Using Greedy Methods. In *NIPS*, pages 1935–1943, 2011. 71, 72

[88] Ji Liu, Jieping Ye, and Ryohei Fujimaki. Forward-Backward Greedy Algorithms for General Convex Smooth Functions Over a Cardinality Constraint. In *ICML*, pages 503–511, 2014. 71, 72

[89] Chao Qian, Yang Yu, and Zhi-Hua Zhou. Subset Selection by Pareto Optimization. In *NIPS*, pages 1774–1782, 2015. 71

[90] Philippe Rigollet and Alexandre Tsybakov. Exponential Screening and Optimal Rates of Sparse Estimation. *The Annals of Statistics*, 39(2):731–771, 2011. 71

[91] Prateek Jain, Ambuj Tewari, and Purushottam Kar. On Iterative Hard Thresholding Methods for High-dimensional M-Estimation. In *NIPS*, pages 685–693, 2014. 71, 72

[92] T. Zhang. Sparse Recovery With Orthogonal Matching Pursuit Under RIP. *IEEE Transactions on Information Theory*, 57(9):6215–6221, September 2011. 71, 86, 87

[93] Andrew R. Barron, Albert Cohen, Wolfgang Dahmen, and Ronald A. DeVore. Approximation and Learning by Greedy Algorithms. *The Annals of Statistics*, 36(1):64–94, 2008. 72

[94] Aurélie C. Lozano, Grzegorz Świrszcz, and Naoki Abe. Group Orthogonal Matching Pursuit for Logistic Regression. In *AISTATS*, pages 452–460, 2011. 72, 87

[95] Ambuj Tewari, Pradeep Ravikumar, and Inderjit S. Dhillon. Greedy Algorithms for Structurally Constrained High Dimensional Problems. In *NIPS*, pages 882–890, 2011. 72

[96] Xiao-Tong Yuan, Ping Li, and Tong Zhang. Gradient Hard Thresholding Pursuit for Sparsity-Constrained Optimization. In *ICML*, pages 127–135, 2014. 72

[97] Sham M. Kakade, Ohad Shamir, Karthik Sridharan, and Ambuj Tewari. Learning Exponential Families in High-Dimensions: Strong Convexity and Sparsity. In *AISTATS*, pages 381–388, 2010. 72, 76

[98] Zhuoran Yang, Zhaoran Wang, Han Liu, Yonina C. Eldar, and Tong Zhang. Sparse Nonlinear Regression: Parameter Estimation and Asymptotic Inference. In *ICML*, pages 2472–2481, 2016. 72, 97

[99] Marshall L. Fisher, George L. Nemhauser, and Laurence A. Wolsey. An analysis of approximations for maximizing submodular set functions–II. In M. L. Balinski and A. J. Hoffman, editors, *Polyhedral Combinatorics: Dedicated to the memory of D.R. Fulkerson*, pages 73–87. Springer Berlin Heidelberg, Berlin, Heidelberg, 1978. 72, 96

[100] Michele Conforti and Gérard Cornuéjols. Submodular set functions, matroids and the greedy algorithm: Tight worst-case bounds and some generalizations of the Rado-Edmonds theorem. *Discrete Applied Mathematics*, 7(3):251–274, March 1984. 72, 96, 137

[101] Baharan Mirzasoleiman, Amin Karbasi, Rik Sarkar, and Andreas Krause. Distributed Submodular Maximization: Identifying Representative Elements in Massive Data. *NIPS*, pages 2049–2057, 2013. 72, 96

[102] Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrák, and Andreas Krause. Lazier Than Lazy Greedy. In *AAAI*, pages 1812–1818, 2015. 72, 74, 96

[103] Rafael da Ponte Barbosa, Alina Ene, Huy L. Nguyen, and Justin Ward. The Power of Randomization: Distributed Submodular Maximization on Massive Datasets. In *ICML*, pages 1236–1244, 2015. 72, 74, 96

[104] Rafael da Ponte Barbosa, Alina Ene, Huy L. Nguyen, and Justin Ward. A New Framework for Distributed Submodular Maximization. In *FOCS*, pages 645–654, 2016. 72, 96

[105] Jason Altschuler, Aditya Bhaskara, Gang (Thomas) Fu, Vahab Mirrokni, Afshin Rostamizadeh, and Morteza Zadimoghaddam. Greedy Column Subset Selection: New Bounds and Distributed Algorithms. In *ICML*, pages 2539–2548, 2016. 72, 97

[106] Thibaut Horel and Yaron Singer. Maximization of Approximately Submodular Functions. In *NIPS*, pages 3045–3053, 2016. 72, 97, 139, 140

[107] Edo Liberty and Maxim Sviridenko. Greedy Minimization of Weakly Supermodular Set Functions. In *APPROX/RANDOM*, pages 19:1–19:22, 2017. 72

[108] Ilija Bogunovic, Junyao Zhao, and Volkan Cevher. Robust Maximization of Non-Submodular Objectives. In *AISTATS*, pages 890–899, 2018. 73, 97, 135, 136, 138, 139, 141, 169

[109] L. Lovász. *Submodular Functions and Convexity*, pages 235–257. Springer Berlin Heidelberg, 1983. 73

[110] S. Fujishige. *Submodular Functions and Optimization*. Annals of Discrete Mathematics. Elsevier Science, 2005. 73

[111] Elad Hazan and Satyen Kale. Online Submodular Minimization. *Journal of Machine Learning Research*, 13:2903–2922, 2012. 73

[112] Niv Buchbinder and Moran Feldman. Constrained Submodular Maximization via a Non-symmetric Technique. http://arxiv.org/abs/1611.03253, 2016. 73, 96

[113] Rishabh K. Iyer and Jeff Bilmes. Polyhedral Aspects of Submodularity, Convexity and Concavity. http://arxiv.org/abs/1506.07329, 2015. 73

[114] Reza Eghbali and Maryam Fazel. Designing Smoothing Functions for Improved Worst-Case Competitive Ratio in Online Optimization. In *NIPS*, pages 3279–3287, 2016. 73

[115] Hamed Hassani, Mahdi Soltanolkotabi, and Amin Karbasi. Gradient Methods for Submodular Maximization. In *NIPS*, pages 5843–5853, 2017. 73

[116] Ethan R. Elenberg, Alexandros G. Dimakis, Moran Feldman, and Amin Karbasi. Streaming Weak Submodularity: Interpreting Neural Networks on the Fly. In *NIPS*, pages 4047–4057, 2017. 74, 93, 105, 136, 156, 173

[117] Marcel Wild. Weakly submodular rank functions, supermatroids, and the flat lattice of a distributive supermatroid. *Discrete Mathematics*, 308:999–1017, 2008. 75, 136, 137

[118] Allan Borodin, Dai Tri Man Le, and Yuli Ye. Weakly Submodular Functions. http://arxiv.org/abs/1401.6697, 2014. 75, 137

[119] Allan Borodin, Dai Tri Man Le, and Yuli Ye. Proportionally Submodular Functions. http://www.cs.toronto.edu/~bor/Papers/proportional-talg-submit.pdf, 2015. 75, 137

[120] Po-Ling Loh and Martin J. Wainwright. Regularized M-estimators with Nonconvexity: Statistical and Algorithmic Theory for Local Optima. *Journal of Machine Learning Research*, 16(1):559–616, Jan. 2015. 75, 155

[121] Lawrence D. Brown. Fundamentals of Statistical Exponential Families with Applications in Statistical Decision Theory. *Lecture Notes–Monograph Series*, 9, 1986. 76, 77

[122] S. A. van de Geer. High-dimensional Generalized Linear Models and the Lasso. *The Annals of Statistics*, 36:614–645, 2008. 76

[123] Dipak K. Dey, Sujit K. Ghosh, and Bani K. Mallick, editors. *Generalized Linear Models: A Bayesian Perspective*. CRC Press, 1 edition, 2000. 77

[124] Thomas Blumensath and Mike E Davies. On the Difference Between Orthogonal Matching Pursuit and Orthogonal Least Squares. https://eprints.soton.ac.uk/142469/, 2007. 80

[125] David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. RCV1: A New Benchmark Collection for Text Categorization Research. *Journal of Machine Learning Research*, 5:361–397, 2004. 88

[126] Tong Zhang. Adaptive Forward-Backward Greedy Algorithm for Sparse Learning with Linear Models. In *NIPS*, pages 1921–1928, 2008. 89

[127] Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. Streaming Algorithms for Submodular Function Maximization. In *ICALP*, pages 318–330, 2015. 94

[128] Gruia Călinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a Monotone Submodular Function Subject to a Matroid Constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011. 96

[129] George L. Nemhauser and Laurence A. Wolsey. Best Algorithms for Approximating the Maximum of a Submodular Set Function. *Math. Oper. Res.*, 3(3):177–188, August 1978. 96

[130] Uriel Feige. A Threshold of ln n for Approximating Set Cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998. 96

[131] Jan Vondrák. Submodularity and curvature: the optimal algorithm. *RIMS Kôkyûroku Bessatsu B23*, pages 253–266, 2010. 96, 137

[132] Maxim Sviridenko, Jan Vondrák, and Justin Ward. Optimal approximation for submodular and supermodular optimization with bounded curvature. In *SODA*, pages 1134–1148, 2015. 96, 137

[133] Niv Buchbinder and Moran Feldman. Deterministic Algorithms for Submodular Maximization Problems. In *SODA*, pages 392–403, 2016. 96

[134] Andreas Krause and Daniel Golovin. Submodular Function Maximization. *Tractability: Practical Approaches to Hard Problems*, 3:71–104, 2014. 96, 149

[135] T-H. Hubert Chan, Zhiyi Huang, Shaofeng H.-C. Jiang, Ning Kang, and Zhihao Gavin Tang. Online Submodular Maximization with Free Disposal: Randomization Beats 1/4 for Partition Matroids. In *SODA*, pages 1204–1223, 2017. 96

[136] Avinatan Hassidim and Yaron Singer. Submodular Optimization Under Noise. In *COLT*, pages 1069–1122, 2017. 97

[137] Sohail Bahmani, Bhiksha Raj, and Petros T. Boufounos. Greedy Sparsity-Constrained Optimization. *Journal of Machine Learning Research*, 14:807–841, 2013. 97

[138] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic Attribution for Deep Networks. In *ICML*, pages 3319–3328, 2017. 97

[139] Moshe Lichman. UCI machine learning repository. http://archive.ics.uci.edu/ml, 2013. 106

[140] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. SLIC Superpixels Compared to State-of-the-art Superpixel Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2274–2282, 2012. 108

[141] Yonatan Bilu and Nathan Linial. Are Stable Instances Easy? *Combinatorics, Probability and Computing*, 21:643–660, 2012. 110

[142] Yuxun Zhou and Costas J. Spanos. Causal meets Submodular: Subset Selection with Directed Information. In *NIPS*, pages 2657–2665, 2016. 139

[143] Garvesh Raskutti, Martin J. Wainwright, and Bin Yu. Restricted Eigenvalue Properties for Correlated Gaussian Designs. *Journal of Machine Learning Research*, 11:2241–2259, 2010. 154

[144] Sahand Negahban, Pradeep Ravikumar, Bin Yu, and Martin J. Wainwright. A Unified Framework for High-Dimensional Analysis of M-Estimators with Decomposable Regularizers. http://arxiv.org/abs/1010.2731v1, 2010. 155

Vita

Ethan R. Elenberg obtained his B.E. in Electrical Engineering from The Cooper Union for the Advancement of Science and Art in 2012, and his M.S. in Electrical and Computer Engineering from The University of Texas at Austin in 2014. He was a Wireless Intern at Apple in 2013, a Summer Research Intern at MIT Lincoln Laboratory in 2014, and a Recommendations Intern at Twitter in 2017. His research interests include Combinatorial Optimization, Interpretable Machine Learning, Graph Algorithms, Image Processing, and Coding Theory.

Permanent address: elenberg@utexas.edu

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.