DISS. ETH N° 24479

Big Data Summarization Using Submodular Functions

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH

(Dr. sc. ETH Zurich)

presented by

BAHARAN MIRZASOLEIMAN

M. Sc. in Computer Science, Sharif University of Technology

born on 21.09.1986

citizen of Iran

accepted on the recommendation of

Prof. Dr. Andreas Krause (ETH Zurich), examiner
Prof. Dr. Jeff Bilmes (University of Washington), co-examiner
Prof. Dr. Yaron Singer (Harvard University), co-examiner

2017

To my parents.

Abstract

Data summarization, a central challenge in machine learning, is the task of finding a representative subset of manageable size out of a large dataset. It has found numerous applications, including image summarization, document and corpus summarization, recommender systems, and non-parametric learning, to name a few. A general recipe to obtain a faithful summary is to turn the problem into selecting a subset of data elements optimizing a utility function that quantifies "representativeness" of the selected set.

Often times, the choice of utility functions used for summarization exhibit submodularity, a natural diminishing returns property. In words, submodularity implies that the added value of any element from the dataset decreases as we include more data points to the summary. Thus, the data summarization problem can be naturally reduced to that of a constrained submodular maximization, or a submodular cover problem.

Although, there are efficient centralized algorithms for the aforementioned problems, they are highly impractical for massive datasets, as sequentially selecting elements on a single machine is heavily constrained in terms of speed and memory. Hence, in order to solve the above submodular optimization problems at scale, we need to make use of MapReduce-style parallel computation models, or resort to streaming algorithms.

In this Thesis, we develop large scale algorithms for submodular summarization. In particular, we present a simple, parallel protocol, called GreeDi for distributed (not-necessarily monotone) submodular maximization subject to cardinality, and other general types of constraints, including matroid and knapsack constraints. In addition, we develop a distributed algorithm, DisCover, for the submodular cover problem, as well as a fast distributed algorithm, FastCover, that enables us to solve the more general problem of covering multiple submodular functions in one run of the algorithm.

We then consider the streaming setting, where at any point of time, the algorithm has access only to a small fraction of data stored in primary memory. We present a single pass streaming procedure, Streaming Local Search, for maximizing a (not-necessarily monotone) submodular function subject to a collection of independence systems and d knapsack constraints. Furthermore, we introduce the dynamic deletion-robust submodular maximization problem, and propose a resilient streaming algorithm, called Robust-Streaming, that is able to produce concise real-time summaries in the face of data deletion requested by users.

Lastly, as a natural complementary goal to the aforementioned works, we consider developing fast centralized algorithms for submodular maximization that can be integrated into distributed frameworks to provide even more scalable algorithms. In particular, we first develop a randomized linear-time algorithm, Stochastic-Greedy, for maximizing a monotone submodular function subject to a cardinality constraint. We then propose a practical and fast algorithm, Fantom, for maximizing a (not necessarily monotone) submodular function subject to the intersection of a p-system and d knapsacks constraints, and show how we can use Fantom to produce personalized summarization.

In addition to providing algorithms and theoretical analyses, we present extensive empirical evaluation of our approaches on several large-scale and real-world summarization problems. These include, but not limited to, summarizing 80 million Tiny Images, more than 45 million user visits from the Featured Tab of the Today Module on Yahoo! Front Page, finding dominating set in Friendster social network with more than 65 million nodes and 1.8 billion edges, and movie recommendation based on 20 million users' ratings from 138,493 users of the MovieLens database.

Zusammenfassung

Datenzusammenfassung, eine zentrale Herausforderung im maschinellen Lernen, besteht darin, eine repräsentative Untermenge von überschaubarer Größe aus einem großen Datensatz zu finden. Es hat zahlreiche Anwendungen gefunden, darunter Bildzusammenfassung, Empfehlungsdienste, nicht-parametrisches Lernen und Dokumenten- und Korpus-Verdichtung, um nur einige Beispiele zu nennen.

Ein allgemeiner Ansatz, um eine treue Zusammenfassung zu erhalten, besteht darin, das Problem in das Auswählen einer Teilmenge von Datenelementen zu transformieren, welches eine Nutzenfunktion optimiert, die die ”Repräsentativität” des ausgewählten Sets quantifiziert.

Häufig zeigt die Wahl der Nutzenfunktionen, die für die Verdichtung verwendet werden, eine Submodularität, d.h. es treten sinkende Erträge auf. Mit anderen Worten, Submodularität impliziert, dass der Mehrwert eines beliebigen Elements aus dem Datensatz abnimmt, je mehr Datenpunkte wir zu der Zusammenfassung hinzufügen.

Somit kann das Datenzusammenfassungsproblem auf das einer beschränkten submodularen Maximierung oder eines submodularen Abdeckungsproblems reduziert werden.

Obwohl es effiziente zentralisierte Algorithmen für die oben erwähnten Probleme gibt, sind sie für massive Datensätze sehr unpraktisch, da die sequentielle Auswahl von Elementen auf einer einzigen Maschine in Bezug auf Geschwindigkeit und Speicher stark einschränkend ist.

Um die obigen submodularen Optimierungsprobleme im großen Maßstab zu lösen, müssen wir also die parallelen Berechnungsmodelle des MapReduce-Verfahrens nutzen oder auf Streaming-Algorithmen zurückgreifen.

In dieser Dissertation entwickeln wir Algorithmen für die submodulare Verdichtung im großem Ausmaß.

Insbesondere präsentieren wir ein einfaches, paralleles Protokoll namens GreeDi für verteilte (aber nicht notwendigerweise monotone) submodulare Maximierung, die der Kardinalität unterliegt und anderen allgemeinen Arten von Einschränkungen, einschließlich Matroid- und Rucksackproblem-beschränkungen.

Außerdem, entwickeln wir einen verteilten Algorithmus, DisCover, für das Problem der submodularen Abdeckung sowie einen schnellen, verteilten Algorithmus, Fast-Cover, der es uns ermöglicht, das allgemeinere Problem der Abdeckung mehrerer submodularer Funktionen in einem Ablauf des Algorithmus zu lösen.

Anschließend betrachten wir den Streaming Fall, wo der Algorithmus zu jedem Zeitpunkt nur auf einen kleinen Anteil der im Hauptspeicher gespeicherten Daten Zugriff hat. Wir präsentieren ein Single-Pass-Streaming-Verfahren, namens Streaming Local Search, zur Maximierung einer (nicht unbedingt monotonen) submodularen Funktion, die einer Sammlung von Unabhängigkeitssystemen und d Rucksackproblem Einschränkung unterworfen ist. Darüber hinaus, stellen wir das dynamische Löschungen-toleriende submodulare Maximierungsproblem vor und schlagen einen widerstandsfähigen Streaming-Algorithmus namens Robust-Streaming vor, der eine präzise Echtzeit-Zusammenfassung unter Berücksichtigung von Datenlöschungsanfragen der Nutzer erzeugen kann.

Abschließend, als ergänzendes Ziel der oben erwähnten Ansätzen betrachten wir die Entwicklung schnellere zentralisierteren Algorithmen für die submodulare Maximierung, die in ein verteiltes Framework integriert werden können, um noch mehr skalierbare Algorithmen zu liefern.

Insbesondere, entwickeln wir einen randomisierten Linearzeitalgorithmus namens Stochastic-Greedy zur Maximierung einer monotonen submodularen Funktion, die einer Kardinalitäsbeschränkung unterliegt. Anschließend schlagen wir einen praktischen und schnellen Algorithmus namens Fantom zur Maximierung einer (nicht notwendigerweise monotonen) submodularen Funktion vor, die Überschneidungen eines p-Systems und d Rucksack-Einschränkungen unterworfen ist, und zeigen, wie man Fantom für eine personalisierte Verdichtung nutzen kann.

Neben der Bereitstellung von Algorithmen und theoretischen Analysen präsentieren wir eine umfangreiche empirische Evaluierung unserer Ansätze für mehrere großangelegte und real-world Verdichtungsprobleme. Dazu gehören, unter anderem, zusammenfassen von: 80 Millionen Tiny Images, mehr als 45 Millionen Nutzer Besuche aus dem Featured Tab der Today Modul auf Yahoo! Front-Seite, die Suche nach dem dominierenden Set in Friendster sozialen Netzwerk mit mehr als 65 Millionen Knoten und 1.8 Milliarden Kanten und Filmempfehlung basierend auf den 20 Millionen Nutzer Bewertungen von 138.493 Benutzern der MovieLens Datenbank.

Acknowledgments

First and foremost, I'd like to thank my advisor, Andreas Krause who continuously inspired, challenged and supported me. This thesis would have not been possible without his guidance and encouragement. Andreas, thank you for your trust, patience, and support. I am truly honored and fortunate to have you as my adviser.

I would also like to thank my thesis committee members—Andreas Krause, Jeff Bilmes, and Yaron Singer—for providing me valuable feedback and a lot of encouragement. Their input and comments have been absolutely invaluable.

I am thankful to my collaborators and co-authors– Andreas Krause, Amin Karbasi, Ashwinkumar Badanidiyuru, Stefanie Jegelka, Yaron Singer, Morteza Zadimoghaddam, Joachim M. Buhmann, and Rik Sarkar –It is very enjoyable to work with all of you. A special thanks goes to Amin, for long-lasting collaborations, and career advice. I am also thankful to Jeff Bilmes for interesting discussions about submodularity.

My thanks also goes to Stefanie and Yaron for hosting my visits to MIT and Harvard, and thanks to D. Sivakumar, Igor Bilogrevic, and Nina Taft for hosting me twice as an intern in Google Research.

I am also very grateful to Thorsten Joachims, Jeff Bilmes, Ravi Kumar, David Kempe, Thomas Hofmann, Yisong Yue, Corinna Cortes, Karl Aberer, Hamed Hassani, Abhimanyu Das, Sven Dickinson, Keren Censor-Hillel, and Masashi Sugiyama who kindly supported me in pursuing my academic endeavor.

Many thanks to our group admin, Rita Klute, for her untiring support, and her kind and caring personality. I also thank all my friends, and members of the Learning and Adaptive Systems group in ETH Zurich.

Finally, I am very grateful to my family for their infinite love and support. To my brilliant and passionate mother who has always been a source of inspiration; to my

patient and ambitious father who makes me be determined, and have big dreams; and to my talented sister who makes life more exciting and fun. This thesis is dedicated with love to them, and to the memory of my lovely grandmother.

Contents

I Background and Survey **1**

1 Introduction **3**

1.1 Thesis Statement and Main Contributions 4

1.1.1 Distributed Algorithms for Submodular Summarization 5

1.1.2 Streaming Algorithms for Submodular Summarization 6

1.1.3 Fast Centralized Algorithms for Submodular Summarization . . 8

1.1.4 Applications and Empirical Studies 9

1.2 Summary of Key Contributions 14

1.3 Organization of this Dissertation 15

1.4 Publications 16

2 Background and Related Work **19**

2.1 Data Summarization 19

2.1.1 Geometric Data Summarization 20

2.1.2 Submodular Data Summarization 20

2.2 Data Summarization by Submodular Optimization 21

2.2.1 Constrained Maximization vs. Coverage 22

2.2.2 (Non-monotone) Maximization with General Constraints 23

2.2.3 Deletion-Robust Submodular Maximization 28

2.3 Large Scale Submodular Maximization 29
2.3.1 Distributed Algorithms 29
2.3.2 Streaming Algorithms 32
2.3.3 Fast Centralized Algorithms 35

3 Applications of Large Scale Submodular Summarization 39
3.1 Nonparametric Learning 39
3.1.1 Active Set Selection in Sparse Gaussian Processes (GPs). 39
3.1.2 Inference for Determinantal Point Processes. 40
3.2 Exemplar Based Clustering 41
3.3 Dominating Sets in Social Networks 42
3.4 Sensor Placement 43
3.5 Summarizing Image Collections 43
3.6 Movie Recommendation 45
3.7 Diversified Image summarization 46
3.8 Revenue Maximization with Multiple Products 46
3.9 Other Examples 47

II Distributed Algorithms 49

4 Overview of part II 51

5 Distributed Submodular Maximization 55
5.1 Submodular Maximization 57
5.1.1 Greedy Submodular Maximization 57
5.1.2 Distributed Submodular Maximization 58
5.1.3 Naive Approaches to Distributed Submodular Maximization . . 59
5.2 The GreeDi Approach for Distributed Submodular Maximization . . 60
5.2.1 An Intractable, yet Communication Efficient Approach 60

5.2.2 Our GreeDi Approximation 61
5.2.3 Performance on Datasets with Geometric Structure 62
5.2.4 Performance Guarantees for Very Large Datasets 65
5.2.5 Handling Decomposable Functions 66
5.2.6 Performance of GreeDi on Random Partitions Without Geometric Structure 67
5.3 (Non-Monotone) Submodular Functions with General Constraints . . . 68
5.3.1 GreeDi Approximation Guarantee under More General Constraints 68
5.4 Experiments 71
5.4.1 Exemplar Based Clustering 72
5.4.2 Active Set Selection 74
5.4.3 Non-Monotone Submodular Function (Finding Maximum Cuts) 77
5.4.4 Comparision with Greedy Scaling. 78
5.5 Summary 79

6 Distributed Submodular Cover: Succinctly Summarizing Massive Data 81
6.1 The Distributed Submodular Cover Problem 82
6.1.1 Naive Approaches Towards Distributed Submodular Cover . . . 83
6.2 DisCover Algorithm for Distributed Submodular Cover 84
6.2.1 Estimating Size of the Optimal Solution 84
6.2.2 Handling Approximations for Submodular Maximization 85
6.2.3 Trading Off Communication Cost and Number of Rounds 86
6.2.4 DisCover 87
6.3 Experiments 89
6.3.1 Exemplar based Clustering 89
6.3.2 Active Set Selection 90
6.3.3 Large Scale Dominating Set with Spark 90
6.4 Summary 92

7 Fast Distributed Submodular Cover: Public-Private Data Summarization **93**
7.1 Problem Statement: Public-Private Summarization 95
7.2 Applications of Pubic-Private Data Summarization 96
7.2.1 Personalized Movie Recommendation 96
7.2.2 Personalized Location Recommendation 97
7.3 FastCover for Fast Distributed Submodular Cover 98
7.4 Experiments 101
7.4.1 Personalized Location Recommendation with Spark 102
7.4.2 Personalized Movie Recommendation with Spark 104
7.4.3 Large Scale Dominating Set with Spark 105
7.5 Summary 107

III Streaming Algorithms **109**

8 Overview of part III **111**

9 Constrained Streaming Submodular Maximization **115**
9.1 Streaming Submodular Maximization 117
9.2 Video Summarization with DPPs 118
9.3 Streaming algorithm for constrained submodular maximization 118
9.3.1 Streaming Local Search for a collection of independence systems 119
9.3.2 Streaming Local Search for independence systems and multiple knapsack constraints 121
9.4 Experiments 123
9.5 Related Work 129
9.5.1 Video Summarization 129
9.5.2 Local Search 130
9.6 Summary 130

10 Deletion-Robust Submodular Maximization 131

10.1 Deletion-Robust Model 133

10.1.1 Dynamic Data: Additions and Deletions 133

10.1.2 Dealing with Limited Time and Memory 134

10.2 Example Applications 134

10.2.1 Summarizing Click-stream and Geolocation Data 135

10.2.2 Summarizing Image Collections 135

10.3 Robust-Streaming Algorithm 136

10.3.1 Increasing the Solution Size Does Not Help 136

10.3.2 Building Multiple Solutions 137

10.3.3 Dealing with Deletions 138

10.4 Experiments 142

10.4.1 Image Collection Summarization 143

10.4.2 Summarizing a stream of geolocation data 144

10.4.3 Large scale click through prediction 144

10.5 Summary 147

IV Fast Centralized Algorithms **149**

11 Overview of part IV 151

12 Lazier than Lazy Greedy 155

12.1 Greedy Algorithm 155

12.1.1 Lazy-Greedy 156

12.2 Stochastic-Greedy Algorithm 157

12.2.1 Random Sampling 157

12.2.2 Random Sampling with Lazy Evaluation 158

12.3 Experimental Results 159

12.3.1 Nonparametric Learning . . . 160
12.3.2 Exemplar-based clustering . . . 160
12.3.3 Sensor Placement . . . 163
12.4 Summary . . . 164

13 Fast Constrained Submodular Maximization: Personalized Summarization 165
13.1 Constrained Submodular Maximization . . . 167
13.2 Applications of Personalized Data Summarization . . . 168
13.3 Our Algorithm: FANTOM . . . 169
13.3.1 Greedy with Density Threshold (GDT) . . . 169
13.3.2 Iterated Greedy with Density Threshold (IGDT) . . . 170
13.3.3 FANTOM . . . 170
13.4 Experiments . . . 173
13.4.1 Personalized movie recommendation . . . 173
13.4.2 Revenue maximization with multiple products . . . 176
13.4.3 Personalized image summarization . . . 177
13.5 Summary . . . 179

V Conclusion and Future Research Directions 181

14 Conclusions 183
14.1 Summary . . . 184
14.1.1 Distributed Algorithms . . . 184
14.1.2 Streaming Algorithms . . . 185
14.1.3 Fast Centralized Algorithms . . . 185
14.1.4 Applications . . . 186
14.2 Future Research Directions . . . 188

A Proofs **191**

A.1 Proofs from Chapter 5 192

A.2 Proofs from Chapter 6 207

A.3 Proofs from Chapter 7 214

A.4 Proofs from Chapter 9 217

A.5 Proofs from Chapter 10 224

A.6 Proofs from Chapter 12 226

A.7 Proofs from Chapter 13 227

Bibliography **245**

Part I

Background and Survey

1 Introduction

The unprecedented growth in modern datasets – coming from different sources and modalities such as images, videos, sensor data, social networks, etc. – demands novel techniques that extract useful information from massive data, while still remaining computationally tractable. One compelling approach that has gained a lot of interest in recent years is *data summarization*: selecting representative subsets of manageable size out of large datasets. Applications range from exemplar-based clustering [DF07a], to document [LB11a; DKR13a] and corpus summarization [Sip+12a], to recommender systems [EA+09; EAG11], just to name a few.

A systematic way for data summarization, used in all the aforementioned applications, is to turn the problem into selecting a subset of data elements maximizing a utility function that quantifies"representativeness" of the selected set. Similarly, one can aim for finding a representative subset, ideally as small as possible, that its corresponding utility is comparable to that of the whole dataset. Often-times, these objective functions satisfy *submodularity*, an intuitive notion of diminishing returns (*c.f.*, [NWF78a]), stating that selecting any given element earlier helps more than selecting it later. Thus, many problems in data summarization require optimizing submodular set functions subject to cardinality, or other feasibility constraints [GK10; KG13], and big data means we have to solve this problem at scale.

Submodularity is a property of set functions with deep theoretical and practical consequences. The seminal result of Nemhauser et al. [NWF78a], that has been of great importance in data mining, is that a simple greedy algorithm – adding one element at a time providing maximal benefit considering the elements picked so far – produces solutions competitive with the optimal (intractable) solution. This greedy algorithm starts with the empty set, and iteratively locates the element with maximal marginal benefit (increasing the utility the most over the elements picked so far). In fact, if assuming nothing but submodularity, no efficient algorithm produces better solutions in general [NW78; Fei98]. This greedy algorithm (and other standard algorithms for submodular optimization), however, unfortunately requires *random access* to the data. Hence, while it can easily be applied if the data fits in main memory, it is impractical for data residing on disk, or arriving over time at a fast pace.

In many domains, data volumes are increasing faster than the ability of individual computers to store them in main memory. In some cases, data may be produced so rapidly that it cannot even be stored. Thus, the following question becomes of crucial importance:

Is it possible to scale up submodular summarization techniques?

One possible approach to tackle this problem is to distribute data to several machines, and seek parallel computation methods. Another natural approach to scale up submodular optimization, is to use streaming algorithms. In fact, in applications where the data arrives at a pace that does not allow even storing it, streaming algorithms are the only viable option. Last but not least, techniques such as randomization can be employed to further speed up the centralized algorithms for submodular maximization. Such fast methods can be integrated into the distributed frameworks to provide even more efficient and scalable algorithms.

1.1 Thesis Statement and Main Contributions

In this Thesis, we present distributed, streaming, and fast centralized techniques for submodular maximization. We claim the following statement:

Sequential, centralized approach for submodular summarization is impractical for truly large-scale problems. By designing efficient distributed, streaming, and fast centralized methods, one can scale up submodular optimization techniques, and still achieve near-optimal solutions.

In order to substantiate this claim, we present the following contributions.

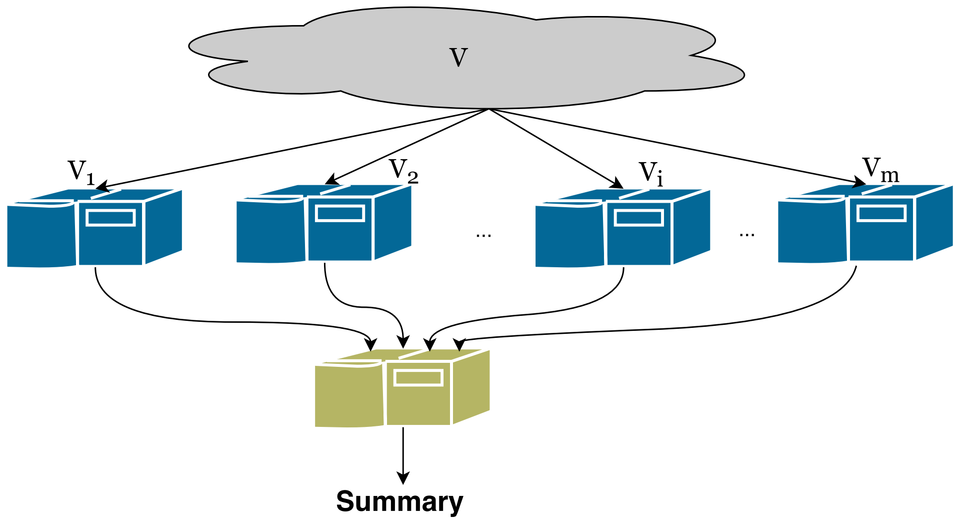


Figure 1.1: *Illustration of a distributed framework.*

1.1.1 Distributed Algorithms for Submodular Summarization

A recent direction in processing large-scale data is to make use of parallelism. MapReduce [DG08] is arguably one of the most successful programming models for reliable and efficient parallel computing. It works by distributing the data to independent machines: map tasks redistribute the data for appropriate parallel processing and the output then gets sorted and processed in parallel by reduce tasks. To perform submodular optimization in MapReduce, we need to design suitable parallel algorithms. Figure 1.1 shows an illustration of a distributed framework. In Part II of this Thesis, we develop distributed algorithms for submodular summarization.

Identifying representative elements in massive data. The greedy algorithms or their accelerated variants [Min78; BV14] that work well for centralized submodular optimization, however, are unfortunately sequential in nature; therefore they are poorly suited for parallel architectures. The question here is how to distribute the data among the machines, which algorithm should run on each machine, and how to merge the resulting solutions. We present a simple, parallel protocol, called GreeDi for distributed submodular maximization subject to cardinality constraints. It requires minimal communication, and can be easily implemented in MapReduce style parallel computation models. We show that under some natural conditions, for large datasets the quality of the obtained solution is provably competitive with the best centralized solution.

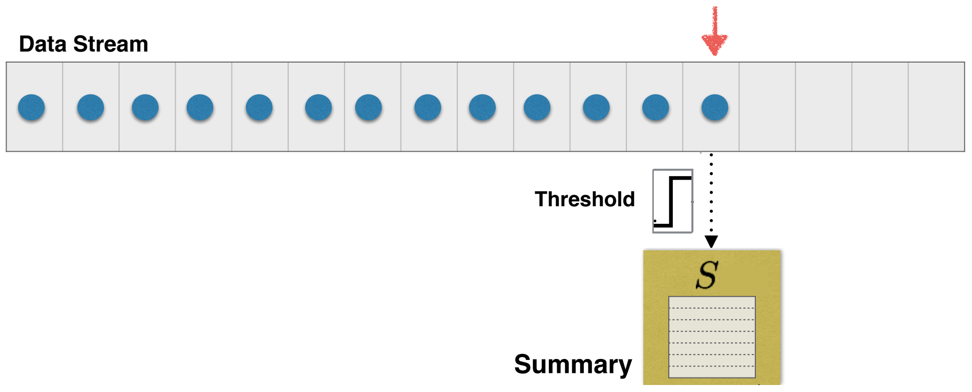


Figure 1.2: *Illustration of an streaming framework.*

We discuss extensions of our approach to obtain approximation algorithms for (not-necessarily monotone) submodular maximization subject to more general types of constraints, including matroid and knapsack constraints.

Succinctly summarizing massive data. In many cases in practice, we are interested to find a succinct summary of the data, i.e., a subset, ideally as small as possible, which achieves a desired (large) fraction of the utility provided by the full dataset. We formalize this problem as a submodular cover problem, and seek efficient algorithms for solving it in face of massive data. We develop the first distributed algorithm, DisCover, for solving the submodular cover problem. It can be easily implemented in MapReduce-style parallel computation models and provides a solution that is competitive with the (impractical) centralized solution.

We then propose a fast distributed algorithm, FastCover, that enables us to solve the more general problem of covering multiple submodular functions in one run of the algorithm. For both algorithms, we study the trade-off between the communication cost (for each round of MapReduce) and the number of rounds. The trade-off lets us choose between a small communication cost between machines while having more rounds to perform or a large communication cost with the benefit of running fewer rounds.

1.1.2 Streaming Algorithms for Submodular Summarization

In some cases, data may be produced so rapidly that it cannot even be stored. Thus, it becomes of crucial importance to process the data in a streaming fashion where at any point of time the algorithm has access only to a small fraction of data stored in primary memory. This approach not only avoids the need for vast amounts of random-access memory but also provides predictions in a timely manner based on the data seen so far, facilitating real-time analytics. Figure 1.2 shows an illustration of an streaming framework. In Part III of this Thesis, we focus on designing streaming algorithms for submodular maximization.

Constrained streaming submodular maximization. In a wide range of applications, such as video summarization, the underlying utility function is non-monotone, and there are often various constraints imposed on the optimization problem to consider privacy or personalization. These may range from a simple limit on the size of the summary to more complex restrictions such as focusing on particular individuals or objects, or excluding them from the summary. We develop the first efficient single pass streaming algorithm, Streaming Local Search, with constant factor approximation guarantee for maximizing a general submodular function under the intersection of a collection of independence systems and d knapsack constraints. The same framework can be applied more generally in many settings where we need to extract a small subset of data from a large stream to train or update a machine learning model.

Deletion-robust data summarization on the fly. An important requirement, which frequently arises in practice, is the ability to summarize a *dynamic* data stream when elements selected for the summary can be deleted at any time. In online services, the users generating the data may decide to exercise their right to restrict the service provider from using (part of) their data due to privacy concerns. Examples include traces of users' activities on social networks (posts, tweets, etc) or images/videos taken with wearables such as Google Glass. In such scenarios, the main problem is to find a representative subset (of practically the same quality) on the fly without having to store all the received data points and rerunning the summarization algorithm. Motivated by this challenge, we introduce the *dynamic deletion-robust submodular maximization* problem. We develop the first resilient streaming algorithm, called Robust-Streaming, with a constant factor approximation guarantee to the optimum solution.

1.1.3 Fast Centralized Algorithms for Submodular Summarization

A natural complementary goal to the aforementioned methods for scaling up submodular maximization techniques, is to develop faster centralized algorithms for submodular maximization. Such methods can be easily integrated into the existing distributed frameworks to provide even more efficient large-scale algorithmic frameworks. Moreover, they can be incorporated into the methods that decompose the submodular function into simpler functions for faster evaluation. In Part IV of this Thesis, we study fast centralized algorithms for submodular maximization.

Lazy stochastic data summarization. While submodularity can be exploited to implement an accelerated version of the classical greedy algorithm, usually called LAZY-GREEDY [Min78], as the size of the data increases, even for small summaries, running LAZY-GREEDY is infeasible. A natural question to ask is whether it is possible to further accelerate LAZY-GREEDY by a procedure with a weaker dependency on the size of the summary k. We propose the first linear time algorithm STOCHASTIC-GREEDY with no dependence on k for cardinality constrained submodular maximization, while simultaneously having the same approximation guarantee (in expectation). STOCHASTIC-GREEDY is substantially faster than LAZY-GREEDY, while being practically identical to it in terms of the utility. The properties of STOCHASTIC-GREEDY make it very appealing and necessary for solving very large scale problems.

Fast constrained data summarization. In general, utility functions designed to measure representativeness of subsets in terms of *conciseness* as well as *diversity* are naturally non-monotone. Furthermore, when summarizing multi-category data in a scalable manner, user preferences is a fundamental constraint. For instance, an individual interested in showing a summary of her recent trip photos may not intend to include more than a handful of them from each point of interest (i.e., matroid constraint). Or, a user interested in watching representative video clips (with different duration) from a particular category may not wish to spend more than a certain amount of time (i.e., knapsack constraint). We cast personalized data summarization as an instance of a general (not necessarily monotone) submodular maximization problem subject to multiple constraints. We develop the first practical and FAst coNsTrained submOdular Maximization algorithm, FANTOM, with strong theoretical guarantees. FANTOM maxi-

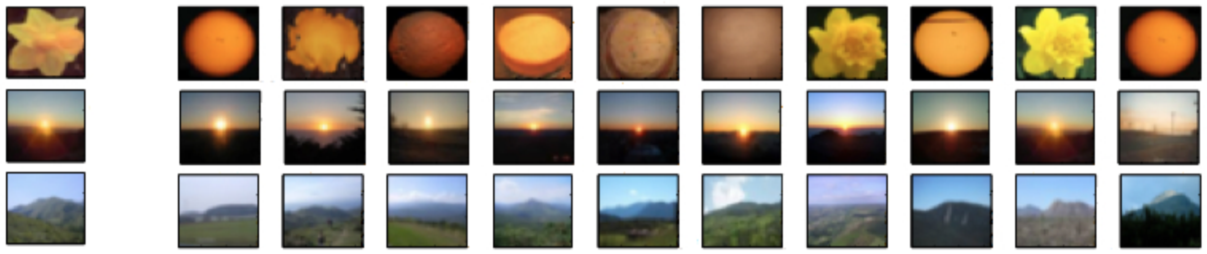


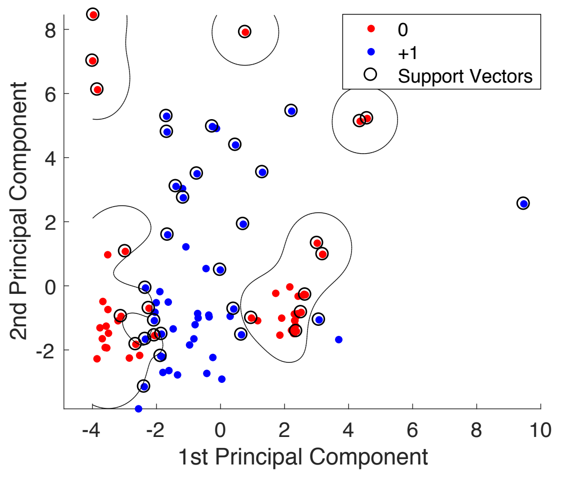
Figure 1.3: *Cluster exemplars (left column) discovered by our distributed algorithm* GreeDi *described in Chapter 5 applied to the Tiny Images dataset [TFF08], and a set of representatives from each cluster.*

mizes a submodular function (not necessarily monotone) subject to the intersection of a p-system and d knapsacks constrains. In particular, a p-system can model different aspects of data, such as categories or time stamps, from which the users choose. In addition, knapsacks encode users' constraints including budget or time.

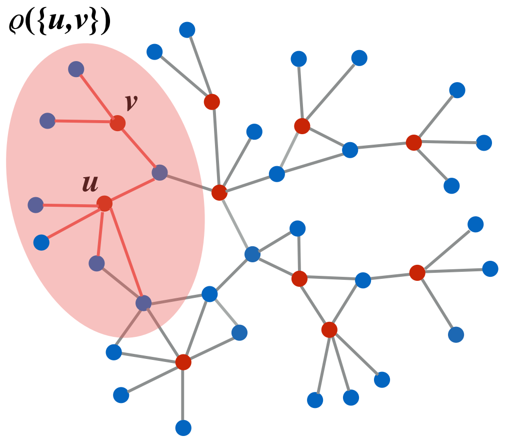
1.1.4 Applications and Empirical Studies

An important goal of this work is to verify the effectiveness of the proposed algorithms through extensive experiments on several large-scale real-world problems.

Large-scale image collection summarization. Given a collection of images, one might be interested in finding a subset that best summarizes and represents the collection. One approach for finding such exemplars is solving the k-medoid problem [KR09], which aims to minimize the sum of pairwise dissimilarities between exemplars and elements of the dataset. This problem can be transformed to maximizing a monotonic submodular utility function [KG10]. For medium-scale problems, the standard greedy algorithms provide good solutions. For massive data however, we need to resort to our large scale algorithms. We perform extensive experimentation on a large dataset containing 80,000,000 *Tiny Images* [TFF08], where each 32 by 32 RGB pixel image was represented by a 3,072 dimensional vector. Figure 1.3 shows a set of exemplars discovered by our large scale algorithms. Empirical studies showed that our algorithms obtained solutions competitive with the best centralized solution, and can scale well to very large datasets.



(a)

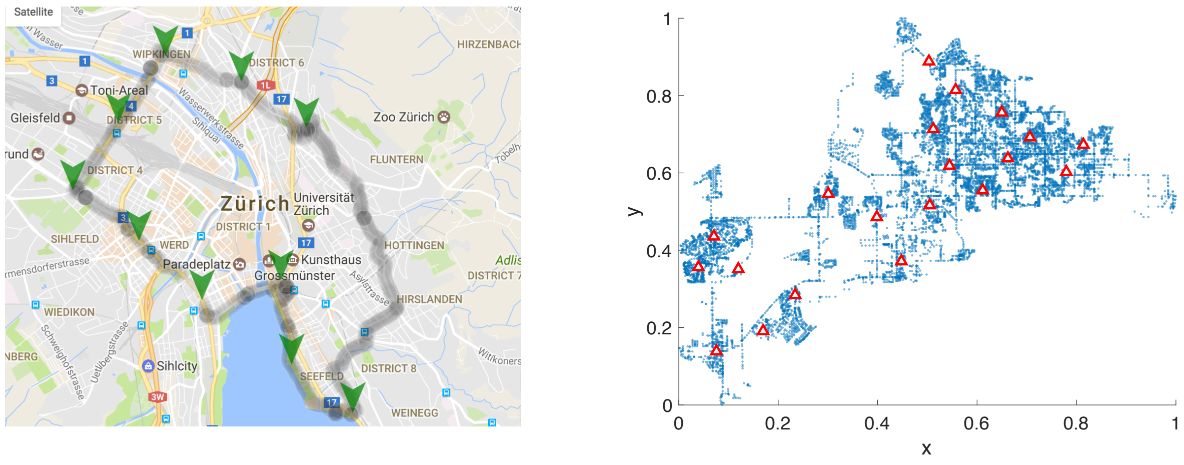


(b)

Figure 1.4: *(a) Example of an active set selected from a subset of the Yahoo! Webscope dataset [Yah12], and the corresponding decision boundary of the kernelized SVM classifier trained on the selected subset. For the sake of presentation, the data is projected onto its 2 largest principal components. (b) Example of a dominating set (marked as red) in a graph. The coverage of u, v is the set of their neighbors and is shown by $\varrho(\{u, v\})$.*

Large-scale dominating set in social networks. Probably the easiest way to define the influence of a subset of users on other members of a social network is by the dominating set problem. We define the coverage size of a set of users by the total number of their friends in the network. The goal is to find the smallest subset such that the coverage size is at least some fraction of the total number of users (*c.f.*, Figure 1.4b). We examine the performance of our algorithms on Friendster social network consists of 65,608,366 nodes and 1,806,067,135 edges [YL15], by obtaining covers for 50%, 40%, 30%, 20% and 10% of the whole graph. Interestingly, our algorithms can obtain a solution that is smaller than the centralized greedy algorithm. Note that running the centralized greedy is impractical if the dataset cannot fit into the memory of a single machine.

Large scale click through prediction. Besides extracting representative elements for sake of explorative data analysis, data summarization is a powerful technique for speeding up learning algorithms. As a concrete example, consider kernel machines (such as kernelized SVMs/logistic regression, Gaussian processes, etc.), which are powerful non-parametric learning techniques. A common approach to scale kernel methods to large datasets is to perform active set selection ([RW06; See04]), i.e., select a



(a) (b)

Figure 1.5: *(a) A summary obtained by our* Robust-Streaming *algorithm described in Chapter 10 from the geo-location trace of a bike route around Zurich [Fat15], (b) Example sensor placement obtained by our algorithm* Stochastic-Greedy *introduced in Chapter 12, in a realistic water network [Kra+08a].*

small, representative subset, and only work with the kernel matrix restricted to this subset. One prominent procedure that is often used in practice is the Informative Vector Machine (IVM) which aims to select a subset based on a monotone submodular function. Again for massive data, we need to use our scalable approaches. Having such a representative subset, we can aim at predicting users' behavior for each displayed article based on historical clicks (*c.f.*, Figure 1.4a).

The same problem can be considered in the streaming setting, where we want to extract the summary, while data is being received. In light of privacy concerns, it is natural to consider participatory models that empower users to decide what portion of their data could be made available, and the algorithm should be able to update the summary to comply with users' preferences.

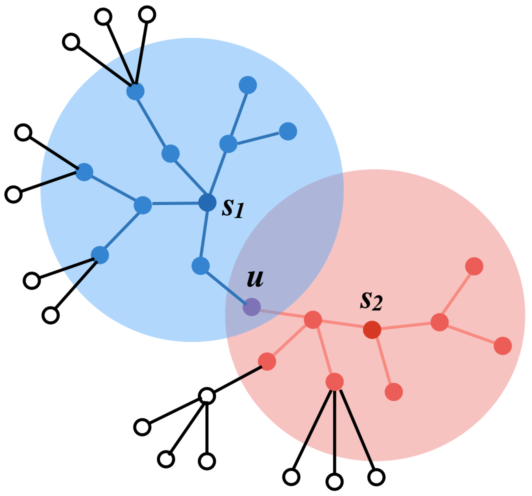
We used our large scale algorithms to summarize the Yahoo! Webscope dataset containing 45,811,883 user click logs for news articles displayed in the Featured Tab of the Today Module on Yahoo! Front Page during the first ten days in May 2009 [Yah12]. Our results showed that, the classifier trained on the summary returned by our algorithms can recover the performance of the classifier trained on the full training data.

Summarizing geo-location sensor data. There exists a tremendous opportunity of harnessing prevalent activity logs and sensing resources. For instance, GPS traces of mobile phones can be used by road traffic information systems (such as Google traffic, TrafficSense, Navigon) to monitor travel times and incidents in real time. This can be done by collecting data generated by active mobile phones along with their GPS coordinates. Continuously sharing all collected data is problematic for several reasons. First, memory and communication constraints may limit the amount of data that can be stored on the mobile devices and transmitted across the network. Second, reasonable privacy concerns may prohibit continuous tracking of users.

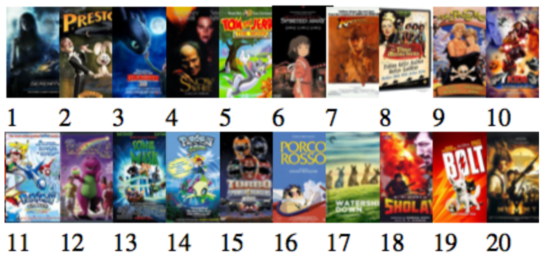
Extracting and communicating the most informative locations (for traffic monitoring purposes) can significantly reduce the power consumption for end users and the processing time needed for the base station. At the same time, we want to allow users to not share or to revoke information about parts of their activity. Therefore, the goal is to select a small subset of elements with a certain diversity in comply with users' preferences while data is being produced, and only communicate this subset to the base station. In order to find such a summary, we apply the algorithms developed in this Thesis on a geo-location dataset collected during a one hour bike ride around Zurich [Fat15], shown in Figure 1.5a.

Sensor placement. When monitoring spatial phenomena, we want to deploy a limited number of sensors in an area in order to quickly detect contaminants. Since the number of sensors is limited, it is important to place them at the most informative locations in order to detect the malicious introduction of contaminants. It has been shown that a large class of important sensor placement objectives satisfy submodularity [Kra+08a], and for which the greedy algorithm gives us a good solution. But, for large datasets we need to resort to our scalable algorithms for submodular maximization. In our experiments, we used the 12,527 node distribution network provided as part of the Battle of Water Sensor Networks (BWSN) challenge [Ost+08], shown in Figure 1.5b. Our experiments showed that our algorithms achieve near-maximal utility at much lower cost compared to the other benchmarks.

Movie recommendation. Consider a movie recommender system, where a user specifies the genres she is interested in, as well as any other constraints she may have –e.g. in terms of money, time, or accessibility–, and the recommender system has to provide



(a)



FANTOM		
Movie	Average rate	Genres
1	3.99	**1**,**2**,15
2	3.98	3,4,5,**9**
3	4.00	**2**,3,4,**9**,19
4	3.08	**9**,11
5	2.53	3,4,5

(b)

Figure 1.6: *(a) Revenue maximization with 2 products. The shaded blue and red regions indicate the influence spread from nodes s_1 and s_2 to buy product 1, 2 respectively. Here, node u has some value to buy both of the products. (b) Personalized movies recommendation using our algorithm* FANTOM *described in Chapter 13 from MovieLens dataset [Mov]. Movies are from 19 genres: Action(1), Adventure(2), Animation (3), Children (4), Comedy (5), Crime (6), Documentary (7), Drama (8), Fantasy (9), Film-Noir (10), Horror (11), Musical (12), Mystery (13), Romance (14), Sci-Fi (15), Thriller (16), War (17), Western (18), IMAX (19). The user is interested in adventure, animation, and fantasy movies (genres 1,2,9).*

a short list of representative movies accordingly. We formulate this problem as a non-negative and non-monotone submodular maximization problem. In our experiments we used a set of 10,437 movies from 19 genres, and 20,000,263 users' ratings from 138,493 users of the MovieLens database [Mov]. Each movie is associated with a 25 dimensional feature vector calculated from user ratings. An example recommendation is shown in Figure 1.6b. Our experiments showed that our algorithm is able to provide a good performance in scenarios where other baselines perform arbitrary poorly.

Revenue maximization with multiple products. In this application, we consider revenue maximization on a social network when multiple products from a basket that can be offered to each user. The goal is to offer for free or advertise some of the products to a set of users such that through their influence on others, the revenue increases (*c.f.*, Figure 1.6a). At the same time, users in a social network may want to see only a small number of advertisements, and we have a limited budget for advertisement. Following

[HMS08] we model user's value for a product based on the set of other users that own the product. The total revenue, that we try to maximize, is a non-monotone submodular function. We performed our experiment on the top 5000 largest communities of the YouTube social network consists of 39,841 nodes and 224,235 edges [YL15]. We consider the settings where we are to advertise up to 100 different types of product across all communities of the same social network. We noted again that our algorithm significantly outperforms the other benchmarks.

1.2 Summary of Key Contributions

The following table summarizes the key contributions of this Thesis, and their organization in the three main parts of this Thesis.

Table 1.1: *Summary of key contributions. n is the size of the dataset, m is the number of muchines, k is the size of the summary, r is the size of the largest feasible solution, Q is the desired utility, α is the approximation guarantee for streaming monotone submodular maximization algorithms under a collection of independence systems, d is the number of knapsack constraints, p is the ratio of the size of the largest vs. smallest independent sets in a p-system.*

Part	Algorithms	Theoretical guarantees	Applications
Distributed Algorithms	GreeDi	$O(1/\sqrt{\min(m,k)})$ rounds: 2	Image summarization Click prediction Active-set selection Finding dominating sets Public-private recommendation
	DisCover	$O(\ln(Q)k/\sqrt{\min(m,k)})$ rounds: $O(\ln Q\sqrt{\min(m,k)})$	
	FastCover	$\ln(Q)/(1-\epsilon)$ rounds: $O(\ln(\frac{n}{km})\ln(Q)/\epsilon)$	
Streaming Algorithms	Streaming-Local Search	$(1-\epsilon)/(1+2/\sqrt{\alpha}+1/\alpha+2d(1+\sqrt{\alpha}))$ passes: 1	Video summarization Location summarization Image summarization Click prediction
	Robust-Streaming	robust to any m deletions passes: 1	
Centralized Algorithms	Stochastic-Greedy	$1-1/e-\epsilon$ complexity: $O(n\ln(1/\epsilon))$	Image summarization Sensor placement
	Fantom	$p(1-\epsilon)/(p+1)(2p+2d+1)$ complexity: $O(nrp\ln(n)/\epsilon)$	Movie recommendation Revenue maximization

1.3 Organization of this Dissertation

In Part I of this Thesis, we will review the concept of submodularity, and introduce relevant terminology and concepts (Chapter 2). We will then review existing works on maximizing submodular set functions in distributed and streaming settings. More specific discussions of related work are presented in the subsequent chapters. In Chapter 3, we discuss concrete applications of large scale submodular maximization with their corresponding submodular objective functions.

Part II presents novel distributed algorithms for submodular maximization and submodular cover problems. In Chapter 5, we develop GreeDi, a distributed algorithm for maximizing a submodular function in MapReduce computational framework. In Chapter 6 and Chapter 7, we consider the problem of submodular cover and propose DisCover and FastCover, two efficient distributed algorithms for submodular cover in MapReduce parallel computation model.

Part III presents our results on streaming submoduar maximization, where data elements are being received at a fast pace, and we need to summarize a massive dataset “on the fly”. In Chapter 9, we introduce Streaming Local Search, our algorithm for maximizing a general submodular function under a collection of independence systems and d knapsack constraints in the streaming setting. In Chapter 10, we propose Robust-Streaming, a robust streaming algorithm able to update the summary in real-time, in case of arbitrary deletions.

In part IV we present our results on fast centralized algorithms, including Stochastic-Greedy, a fast stochastic algorithm for submodular maximization with running time independent of the size of the summary (Chapter12), and Fantom, a fast algorithm for maximizing a (not-necessarily) monotone submodular function subject to a p-system and d knapsack constraints (Chapter13).

Lastly, in Part V we present our conclusions, and discuss interesting open problems for future work (Chapter 14).

Appendix A presents the proofs of all theoretical results described in this Thesis.

1.4 Publications

Publications Covered in this Dissertation

This dissertation covers material primarily from the following publications.

- Baharan Mirzasoleiman, Amin Karbasi, Rik Sarkar, Andreas Krause, ”Distributed Submodular Maximization”, *Journal of Machine Learning Research*, 2016. 17(238):1-44. [Mir+16]
- Baharan Mirzasoleiman, Stefanie Jegelka, Andreas Krause, ”Streaming Non-monotone Submodular Maximization: Personalized Video Summarization on the Fly,” In: *Proc. Conference on Artificial Intelligence (AAAI)*. 2018. [MJK18]
- Baharan Mirzasoleiman, Amin Karbasi, Andreas Krause, ”Deletion Robust Submodular Maximization: Data Summarization with the Right to be Forgotten,” In: *Proc. International Conference on Machine Learning (ICML)*. 2017. [MKK17]
- Baharan Mirzasoleiman, Morteza Zadimoghaddam, Amin Karbasi, ”Fast Distributed Submodular Cover: Public Private Data Summarization,” In: *Proc. Advances in Neural Information Processing Systems (NIPS)*. 2016. [MZK16]
- Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, ”Fast Constrained Submodular Maximization: Personalized Data Summarization,” In: *Proc. International Conference on Machine Learning (ICML)*. 2016. [MBK16]
- Baharan Mirzasoleiman, Amin Karbasi, A. Badanidiyuru, Andreas Krause, ”Distributed Submodular Cover: Succinctly Summarizying Massive Data,” In: *Proc. Advances in Neural Information Processing Systems (NIPS)*. 2015. [Mir+15a]
- Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrak, Andreas Krause, “Lazier than Lazy Greedy”, In: *Proc. Conference on Artificial Intelligence (AAAI)*. 2015. [Mir+15b]
- Baharan Mirzasoleiman, Amin Karbasi, Rik Sarkar, Andreas Krause, ”Distributed Submodular Maximization: Identifying Representative Elements in Massive Data,” In: *Proc. Advances in Neural Information Processing Systems (NIPS)*. 2013. [Mir+13]

Publications Not Covered in this Dissertation

The following publications are relevant to the topic of this dissertation, but are not covered in this Thesis.

- Andrew Bian, Baharan Mirzasoleiman, Juachim Buhmann, Andreas Krause, ”Guaranteed Non-Convex Optimization: Submodular Maximization over Continuous Domain”, In: *Proc. International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2017. [Bia+17]
- Eric Balkanski, Andreas Krause, Baharan Mirzasoleiman, Yaron Singer, ”Learning Sparse Combinatorial Representations via Two-stage Submodular Maximization,” In: *Proc. International Conference on Machine Learning (ICML)*. 2016. [Bal+16]
- Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, Andreas Krause, “Streaming Submodular Maximization: Massive Data Summarization on the Fly”, In: *Proc. ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*. 2014. [Bad+14]

2 Background and Related Work

In this chapter, we provide background on submodular maximization, as well as the terminology used in this Thesis. We will also review related work that is relevant to most chapters of this thesis. In the next chapter, we will discuss concrete applications of large scale submodular maximization with their corresponding submodular objective functions.

2.1 Data Summarization

The large volume of modern datasets presents new computational challenges as the diverse, feature-rich, unstructured and usually high-resolution data does not allow for effective data-intensive inference. In this regard, data summarization is a compelling (and sometimes the only) approach that aims at both exploiting the richness of large-scale data and being computationally tractable. Instead of operating on complex and large data directly, carefully constructed summaries not only enable the execution of various data analytics tasks but also improve their efficiency and scalability. Geometric methods and submodular optimization techniques are the two most important classes of techniques for extracting such informative summaries.

2.1.1 Geometric Data Summarization

Originating from computational geometry, geometric summarization techniques try to compress a large dataset in linear or near-linear time, such that the result of running a more intricate algorithm on the summariy approximates those of the full dataset. *Coresets* and *sketches* are the two most important classes of geometric data summarization techniques.

Coreset. A coreset is a weighted subset of the data such that the quality of any clustering evaluated on the coreset closely approximates the quality on the full dataset. They have been successfully used to scale up many machine learning problems, including SVMs, and clustering models such as K-Means and Gaussian mixture models to massive datasets [HPM04; TKC05; LBK16].

Sketch. A sketch is a compressed mapping of the full dataset onto a data structure while preserving the structural properties of the original data. They approximate certain queries on the full dataset, and are easy to update with new or changed data. In particular, a linear sketch is one where the mapping is a linear function of each data point, thus making it easy for data to be added, subtracted, or modified [GK04; Aga+13; BEM16].

Geometric summarization techniques allow for efficient approximate inference with strong theoretical guarantees. However, the algorithms and the corresponding theory are usually specific to each specific problem. This limits their applicability to mostly clustering, and a few other machine learning problems. In this Thesis, we discuss data summarization by submodular optimization.

2.1.2 Submodular Data Summarization

Over the recent years, submodular optimization has been identified as a powerful tool for numerous data mining and machine learning applications. Submodular functions contain a large class of functions that naturally arises in data mining and machine learning applications, and hence there has been a recent surge of interest in applying submodular optimization methods to many problems, including viral marketing [KKT03], network monitoring[Les+07] , sensor placement and information gathering

[KG11], news article recommendation [EA+09], nonparametric learning [KG10; RG13], document and corpus summarization [LB11a; Du+13; Sip+12a], crowd teaching [Sin+14], and MAP inference of Determinental Point Process [GKT12b]. A key reason for such a wide range of applications is the existence of efficient algorithms (with near-optimal solutions) for a divers set of constraints.

2.2 Data Summarization by Submodular Optimization

Suppose that we have a large dataset V of size n, and we are interested in finding a subset of data points that are most representative according to some objective function, $f : 2^V \to \mathbb{R}_+$. For each $S \subseteq V$, $f(S)$ quantifies the utility of set S, capturing, e.g., how well S represents V according to some objective. We will discuss concrete instances of functions f in Chapter 3. A set function f is naturally associated with a *discrete derivative*, also called the *marginal gain*,

$$\triangle_f (e|S) \doteq f(S \cup \{e\}) - f(S), \tag{2.2.1}$$

which quantifies the increase in utility obtained when adding $e \in V$ to set $S \subseteq V$. Submodular functions are set functions which satisfy the following natural diminishing returns property.

Definition 1 (*c.f.*, **Nemhauser, Wolsey, and Fisher [NWF78a]).** A set function $f : 2^V \to \mathbb{R}$ is *submodular*, if for every $A \subseteq B \subseteq V$ and $e \in V \setminus B$

$$f(A \cup \{e\}) - f(A) \geq f(B \cup \{e\}) - f(B).$$

Equivalently, f is *submodular* iff for all $A \subseteq B \subseteq V$ and $e \in V \setminus B$ the following condition holds

$$\triangle_f (e|A) \geq \triangle_f(e|B). \tag{2.2.2}$$

That means, adding an element e in context of a set A helps at least as much as adding e in context of a superset B of A. Furthermore, f is called *monotone* iff for all $A \subseteq B \subseteq V$ it holds that $f(A) \leq f(B)$, and *non-monotone* otherwise. Throughout this Thesis, we adopt the common assumption that f is given in terms of a value oracle (a black box) that computes $f(S)$ for any $S \subseteq V$.

2.2.1 Constrained Maximization vs. Coverage

Our general goal in data summarization is to select a small subset $A \subseteq V$ out of a large dataset such that A is the most representative according to the objective function f.

A natural optimization problem here is to find a summary A^* of size at most k that maximizes the utility, i.e.,

$$A^* = \arg\max_{A:|A|\leq k} f(A). \tag{2.2.3}$$

Here, we try to find the most representative subset of data points subject to a cardinality constraint. We call Problem 2.2.3 a *submodular maximization* problem.

Similarly, we can aim at finding the smallest set A^* such that it achieves a desired utility $Q = (1-\epsilon)f(V)$ for some $0 \leq \epsilon \leq 1$. More precisely,

$$A^* = \arg\min_{S\subseteq V} |A| \quad \text{s.t.} \quad f(A) \geq Q. \tag{2.2.4}$$

Problem 2.2.4 is called a *submodular cover* problem.

Optimization problems 2.2.3 and 2.2.4 are NP-hard for many classes of submodular functions [Fei98]. However, a simple greedy algorithm that starts with the empty set and iteratively augments the current solution with an element of maximum incremental value

$$A_i = A_{i-1} \cup \{\arg\max_{e\in V} \triangle_f(e|A_{i-1})\}. \tag{2.2.5}$$

is known to be very effective.

Theorem 2 (submodular maximization [NWF78a]). *For any non-negative and monotone submodular function f, the greedy heuristic always produces a solution $A^g[k]$ of size k that achieves at least a constant factor $(1-1/e)$ of the optimal solution.*

$$f(A^g[k]) \geq (1-1/e) \max_{|A|\leq k} f(A).$$

This result can be easily extended to $f(A^g[l]) \geq (1-e^{-l/k}) \max_{|A|\leq k} f(A)$, where l and k are two positive integers [see, KG13].

Theorem 3 (submodular cover [Wol82]). *For any integral submodular function f (i.e., $f: 2^V \rightarrow N$), the size of the solution returned by the greedy algorithm $|A^g|$ is at most*

$H(\max_e f(e))|A^*|$, where A^* is the optimal solution, and $H(z)$ is the z-th harmonic number and is bounded by $H(z) \leq 1 + \ln(z)$.

$$|A^g| \leq (1 + \ln(\max_e f(e)))|A^*|.$$

For several classes of monotone submodular functions, it is known that the above approximation guarantees are the best that one can hope for [NW78; Fei98; KG05b]. Moreover, the greedy algorithm can be accelerated using lazy evaluations [Min78; Les+07]. We will discuss greedy with lazy evaluations later in Chapter 12 (Section 12.1.1).

2.2.2 (Non-monotone) Maximization with General Constraints

Instead of specifying feasibility of observation sets A by requiring that $|A| \leq k$ (Eq. 2.2.3) or $f(A) \geq Q$ Eq. (2.2.4), we can consider more general classes of constraints, by defining a collection of feasible sets $A \subseteq 2^V$, and requiring that the chosen sets A satisfy $A \in \zeta$. In this case, we would like to solve

$$\text{Maximize } f(S)$$
$$\text{Subject to } S \in \zeta.$$

Here, we assume that the feasible solutions should be members of the constraint set $\zeta \subseteq 2^V$. The function $f(\cdot)$ is submodular but may not be monotone. Throughout this section we assume that the constraint set ζ is hereditary, meaning that if $A \in \zeta$ then for any $B \subseteq A$ we also require that $B \in \zeta$. Cardinality constraints are obviously hereditary, so are all the examples we mention below.

Independence Systems

An independence system is a pair $\mathcal{M}^I = (V, \mathcal{I})$ where V is a finite (ground) set, and $\mathcal{I} \subseteq 2^V$ is a family of independent subsets of V satisfying the following two properties:

- The empty set is independent, i.e., $\emptyset \in \mathcal{I}$, and
- Every subset of an independent set is independent, i.e., for any $A \subseteq B \subseteq V, B \in \mathcal{I}$ implies that $A \in \mathcal{I}$ (hereditary property).

Matroid: A matroid $\mathcal{M} = (V, \mathcal{I})$ is an independence system with the additional *augmentation property* or the independent set *exchange property*:

- If $A, B \in \mathcal{I}$ and $|B| > |A|$, there is an element $e \in B \setminus A$ such that $A \cup \{e\} \in \mathcal{I}$.

A *uniform* matroid is the family of all subsets of size at most l. In a *partition* matroid, we have a collection of disjoint sets B_i and integers $0 \leq l_i \leq |B_i|$ where a set A is independent if for every index i, we have $|A \cap B_i| \leq l_i$. Maximizing a submodular function subject to matroid constraints has found several applications in machine learning and data mining, ranging from content aggregation on the web [AMT13] to viral marketing [NN12] and online advertising [SGK09].

One way to approximately maximize a monotone submodular function $f(S)$ subject to the constraint that each S is independent, i.e., $S \in \mathcal{I}$, is to use a generalization of the greedy algorithm. This algorithm, which starts with an empty set and in each iteration picks the feasible element with maximum benefit until there is no more element e such that $S \cup \{e\} \in \mathcal{I}$, is guaranteed to provide a $\frac{1}{2}$-approximation of the optimal solution [FNW78]. Recently, this bound has been improved to $(1 - 1/e)$ using the continuous greedy algorithm [Cal+11]. For non-negative and non-monotone submodular functions under a cardinality constraint k, a $1/e + 0.004$ approximation can be achieved by iteratively selecting a random element from the top k elements with the highest marginal gains, until we find a solution of size k [Buc+14]. The best known result for the same problem under a matroid constraint is a $1/e - \epsilon$ approximation based on continuous greedy [FNS11a].

Curvature: For a submodular function f, the total curvature of f with respect to a set S is defined as:

$$c = 1 - \min_{j \in V} \frac{f(j|S \setminus j)}{f(j)}.$$

Intuitively, the notion of curvature determines how far away f is from being modular. In other words, it measures how much the marginal gain of an element w.r.t. set S can decrease as a function of S. In general, $c \in [0, 1]$, and for additive (modular) functions, $c = 0$, i.e., the marginal values are independent of S. In this case, the greedy algorithm returns the optimal solution to $\max\{f(S) : S \in \mathcal{I}\}$. In general, the greedy algorithm gives a $\frac{1}{1+c}$-approximation to maximizing a non-decreasing submodular function with curvature c subject to a matroid constraint [CC84]. In case of the uniform matroid $\mathcal{I} = \{S : |S| \leq k\}$, the approximation factor is $(1 - e^{-c})/c$.

Intersection of Matroids: A more general case is when we have p matroids $\mathcal{M}_1 = (V, \mathcal{I}_1), \mathcal{M}_2 = (V, \mathcal{I}_2), ..., \mathcal{M}_p = (V, \mathcal{I}_p)$ on the same ground set V, and we want to maximize the submodular function f on the intersection of p matroids. That is, $\mathcal{I} = \bigcap_i \mathcal{I}_i$ consists of all subsets of V that are independent in all p matroids. This constraint arises, e.g., when optimizing over rankings (which can be modeled as intersections of two partition matroids). Another recent application considered is finding the influential set of users in viral marketing when multiple products need to be advertised and each user can tolerate only a small number of recommendations [Du+13]. For p matroid constraints, the $\frac{1}{p+1}$-approximation provided by the greedy algorithm [FNW78] has been improved to a $(\frac{1}{p} - \epsilon)$-approximation for $p \geq 2$ by [LSV09]. For the non-monotone case, a $1/(p + 1 + 1/p + \epsilon)$-approximation based on local search is also given by [LSV09].

***p*-matchoid:** A *p-matchoid* generalizes matchings and intersection of matroids. For q matroids $\mathcal{M}_\ell = (V_\ell, \mathcal{I}_\ell)$, $\ell \in [q]$, defined over overlapping ground sets V_ℓ, and for $V = \cup_{\ell=1}^{q} V_\ell$, $\mathcal{I} = \{S \subseteq V : S \cap V_\ell \in \mathcal{I}_\ell \ \ \forall \ell\}$, we have that $\mathcal{M}^p = (V, \mathcal{I})$ is a p-matchoid if every element $e \in V$ is a member of V_ℓ for at most p indices. For monotone submodular functions, the greedy algorithm provides a $\frac{1}{p+1}$-approximation guarantee subject to a p-matchoid constraint [FNW78]. For non-monotons submodular functions, the randomized algorithms of [FNS11a; Cal+11] provide a $(1 - \epsilon)(2 - o(1))/ep$ approximation guarantee in expectation.

***p*-systems:** p-independence systems generalize constraints given by the intersection of p matroids. Given an independence family $\mathcal{I}$ and a set $V' \subseteq V$, let $S(V')$ denote the set of maximal independent sets of $\mathcal{I}$ included in V', i.e., $S(V') = \{A \in \mathcal{I} \mid \forall e \in V' \setminus A : A \cup \{e\} \notin \mathcal{I}\}$. Then we call $(V, \mathcal{I})$ a p-system if for all nonempty $V' \subseteq V$ we have

$$\max_{A \in S(V')} |A| \leq p \cdot \min_{A \in S(V')} |A|.$$

Similar to p matroid and p-matchoid constraints, the greedy algorithm provides a $\frac{1}{p+1}$-approximation guarantee for maximizing a monotone submodular function subject to a p-systems constraint [FNW78]. For the non-monotone case, a $2/(3(p + 2 + 1/p))$-

approximation can be achieved by combining an algorithm of [Gup+10a] with the result for unconstrained submodular maximization of [Buc+12].

Knapsack Constraints

In many applications, including feature and variable selection in probabilistic models [KG05b] and document summarization [LB11b], elements $e \in V$ have non-uniform costs $c(e) > 0$, and we wish to find a collection of elements S that maximize f subject to the constraint that the total cost of elements in S does not exceed a given budget $\mathcal{R}$, i.e.

$$\max_{S} f(S) \;\; \text{s.t.} \; \sum_{v \in S} c(v) \leq \mathcal{R}.$$

Since the simple greedy algorithm ignores cost while iteratively adding elements with maximum marginal gains according to Eq. 2.2.5 until $|S| \leq \mathcal{R}$, it can perform arbitrary poorly. However, it has been shown that taking the maximum over the solution returned by the greedy algorithm that works according to Eq. 2.2.5 and the solution returned by the modified greedy algorithm that optimizes the cost-benefit ratio

$$v^* = \text{arg max} \underset{c(v) \leq \mathcal{R} - c(S)}{\underset{e \in V \setminus S}{}} \frac{f(S \cup \{e\}) - f(S)}{c(v)},$$

provides a $(1 - 1/\sqrt{e})$-approximation of the optimal solution [KG05a]. Furthermore, a more computationally expensive algorithm which starts with all feasible solutions of cardinality 3 and augments them using the cost-benefit greedy algorithm to find the set with maximum value of the objective function provides a $(1 - 1/e)$-approximation [Svi04]. For maximizing non-monotone submodular functions subject to knapsack constraints, a $(1/5 - \epsilon)$-approximation algorithm based on local search was given by [Lee+09].

Multiple Knapsack Constraints: In some applications such as procurement auctions [GKP01], video-on-demand systems and e-commerce [KST09], we have a d-dimensional budget vector $\mathcal{R}$ and a set of element $e \in V$ where each element is associated with a d-dimensional cost vector. In this setting, we seek a subset of elements $S \subseteq V$ with a total cost of at most $\mathcal{R}$ that maximizes a non-decreasing submodular function f. Kulik, Shachnai, and Tamir [KST09] proposed a two-phase algorithm that provides a

Table 2.1: *Approximation guarantees (τ) for monotone and non-monotone submodular maximization under different constraints.*

Constraints	Approximation (τ)			
	monotone submodular functions		non-monotone submodular functions	
Cardinality	$1 - 1/e$	[FNW78]	$1/e + 0.004$	[Buc+14]
1 matroid	$1 - 1/e$	[Cal+11]	$1/e - \epsilon$	[FNS11a]
p matroid	$1/p - \epsilon$	[LSV09]	$1/(p + 1 + 1/p + \epsilon)$	[LSV09]
1 knapsack	$1 - 1/e$	[Svi04]	$1/5$ - ϵ	[Lee+09]
d knapsack	$1 - 1/e - \epsilon$	[KST09]	$1/5$ - ϵ	[Lee+09]
p-system	$1/(p + 1)$	[FNW78]	$2/(3(p + 2 + 1/p))$	[Gup+10a]
p-system + d knapsacks	$1/(p + 2d + 1)$	[BV14]	$(1-\epsilon)p/(p+1)(2p+2d+1)$	[Ch. 13]

$(1 - 1/e - \epsilon)$-approximation for the problem by first guessing a constant number of elements of highest value, and then taking the value residual problem with respect to the guessed subset. For the non-monotone case, [Lee+09] provided a $(1/5 - \epsilon)$-approximation based on local search.

***p*-system and *d* knapsack constraints:** A more general type of constraint that has recently found interesting applications in viral marketing [Du+13] can be constructed by combining a p-system with d knapsack constraints which comprises the intersection of p matroids or d knapsacks as special cases. Badanidiyuru et al. [BV14] proposed a modified version of the greedy algorithm that guarantees a $1/(p + 2d + 1)$-approximation for maximizing a monotone submodular function subject to a p-system and d knapsack constraints.

For general (not-necessarily monotone) submodular functions, we will provide the first practical algorithm for constrained submodular maximization in Chapter 13. Our proposed method, FANTOM, achieves a $(1 - \epsilon)p/(p + 1)(2p + 2d + 1)$-approximation guarantee for maximizing a general submodular function subject to the intersection of a p-system and d knapsack constrains.

Table 2.1 summarizes the approximation guarantees for monotone and non-monotone submodular maximization under different constraints.

2.2.3 Deletion-Robust Submodular Maximization

An important requirement, which frequently arises in practice, is robustness. For example, when summarizing user activities, some users may decide not to share, or to revoke information about parts of their activity, resulting in an unrepresentative summary. In such scenarios, the goal is to find a summary that is still a good representatives of the data after removal of some of the elements. This problem can be formulated as a deletion-robust submodular maximization problem. There has been recent efforts to construct solutions that are robust against deletions in a batch mode way.

Krause et al. [Kra+08b] proposed the problem of robust submodular observation selection, where there are several submodular objective functions which we want to simultaneously optimize. The goal is to select observations which are robust against a worst-case objective function. More formally, for a collection of normalized monotonic submodular functions $F_1, \cdots, F_m$, we want to solve $\max_{A \in V} \min_i F_i(A)$, subject to $|A| \leq k$. Submodular maximization of f robust against m deletions can be cast as an instance of the above problem: $\max_{|A| \leq k} \min_{|B| \leq m} f_B(A)$, where $f_B(A) = f(A \setminus B)$. However, the running time will be exponential in m as there are exponentially many functions ℓ to consider (one for each set of size m).

Recently, Orlin, Schulz, and Udwani [OSU16] considered the deletion-robust submodular maximization and developed an algorithm with an asymptotic guarantee 0.285 for deleting up to $m = o(\sqrt{k})$ elements. The algorithm uses the greedy solution as a subroutine m times, where each time it performs m steps of the greedy solution starting from $\emptyset$. It then greedily augments the solution set until it finds k elements. Better theoretical results can be obtained for only 1 or 2 deletions. In a very recent work, Bogunovic et al. [Bog+17] proposed a new Partitioned Robust (PRO) submodular maximization algorithm that achieves the same guarantee for more general $m = o(k)$. The idea is to arrange the constructed set into partitions consisting of buckets whose sizes increase exponentially with the partition index. The guarantee is then obtained based on a recursive relationship between the objective values of buckets appearing in adjacent partitions.

2.3 Large Scale Submodular Maximization

Centralized approaches for constrained submodular maximization require random access to the data. Once the size of the dataset increases beyond the memory capacity (typical in many modern datasets) or the data is arriving incrementally over time, these methods can no longer be used. In such cases, either we seek parallel computation methods by distributing data to independent machines or we devise streaming algorithms where at each point in time, only a tiny portion of data is processed. Both approaches has recently received considerable attentions as prominent foundation for large scale machine learning and data mining algorithms [Chu+07; EPF08].

2.3.1 Distributed Algorithms

One possible approach to scale up submodular optimization is to distribute the data to several machines, and seek parallel computation methods.

Distributed Data Analysis and MapReduce Due to the rapid increase in dataset sizes, and the relatively slow advances in sequential processing capabilities of modern CPUs, parallel computing paradigms have received much interest. Inhabiting a sweet spot of resiliency, expressivity and programming ease, the MapReduce style computing model [DG08] has emerged as prominent foundation for large scale machine learning and data mining algorithms [Chu+07; EPF08]. A MapReduce job takes the input data as a set of $< key; value >$ pairs. Each job consists of three stages: the *map* stage, the *shuffle* stage, and the *reduce* stage. The map stage, partitions the data randomly across a number of machines by associating each element with a *key* and produce a set of $< key; value >$ pairs. Then, in the shuffle stage, the value associated with all of the elements with the same key gets merged and sent to the same machine. Each reducer then processes the values associated with the same key and outputs a set of new $< key; value >$ pairs with the same key. The reducers' output could be input to another MapReduce job and a program in MapReduce paradigm can consist of multiple rounds of map and reduce stages [KSV10].

Distributed submodular maximization. Kumar et al. [Kum+13] gave a distributed algorithms for maximizing a monotone submodular function. The high level idea is to simulate the ϵ-greedy algorithm [Gup+10a] in a distributed setting, by iteratively sampling a small subset of the elements that fits on a single machine, and runs the ϵ-greedy algorithm on the sample in order to obtain a candidate solution. This intermediate solution is then used to prune some of the elements in the dataset and reduce the size of the ground set. For a cardinality constraint, the number of rounds is a constant but for more general constraints such as a matroid, or a p-system constraint, the number of rounds is $\Theta(\log(\Delta))$ where Δ is the maximum marginal gain of a single element. The maximum marginal gain can be even larger than the size of the ground set. This dependency makes the approach infeasible for massive datasets.

In Chapter 5, we will provide the first efficient two-rounds distributed algorithm for maximizing a non-negative submodular function with a worst-case $1/\min(m,k)$-approximation guarantee to the centralized solution (m is the number of machines and k is the cardinality of the desired solution). This algorithm is proposed in parallel to [Kum+13], and can handle general types of constraints. We will further show that, under some additional assumptions on the objective function, i.e. Lipschitz continuity, performance close to the centralized greedy algorithm can be achieved.

Barbosa et al. [Bar+15] improved the worst-case guarantee for the distributed framework proposed in Chapter 5 to $1/\Theta(\min(\sqrt{k}, m))$. In Chapter 6, we will show that this bound can be further improved to $1/\Theta(\min(\sqrt{k}, \sqrt{m}))$. Barbosa et al. [Bar+15] further used a randomized analysis to provide a 0.31 approximation guarantees in expectation for the above distributed framework.

In parallel, Mirrokni and Zadimoghaddam [MZ15] showed that a 0.27-approximation in expectation can be achieved under cardinality constraint, using notion of randomized composable core-sets. They further observed that one cannot achieve a better than the $1/2$ factor to the centralized solution via core-sets of size k (cardinality constraint) using greedy or any algorithm in a family of local search algorithms. Table 2.2 summarizes the approximation guarantees and the number of rounds required by each of the aforementioned distributed algorithms.

Distributed submodular cover. Despite it's importance, there has been relatively little study of how to find covers efficiently until quite recently, where distributed solutions for some special cases of the submodular cover problem have been proposed.

In particular, for the set cover problem (i.e., find the smallest subcollection of sets

Table 2.2: *Approximation guarantees for distributed monotone and non-monotone submodular maximization under different constraints. $\rho[\zeta]$ is the maximum size of the desired solution, and bounds for randomized algorithms that hold in expectation are marked (R).*

Constraints	monotone submodular functions		non-monotone submodular functions	
	Approximation	# rounds	Approximation	# rounds
Cardinality	$O\left(\frac{0.63}{\sqrt{\min(m,k)}}\right)$	2	$O\left(\frac{0.37}{\sqrt{\min(m,k)}}\right)$	2 [Ch. 5, 6]
	0.316 (R)	2	0.12 (R)	2 [Bar+15]
	0.27 (R)	2	0.14 (R)	2 [MZ15]
	$\frac{1}{2} - \epsilon$	$O(\frac{1}{\delta})$	–	– [Kum+13]
d knapsack	$\Omega(\frac{1}{d})$			
p-system	$\frac{1}{p+1}\lceil\frac{1}{\delta}\rceil^{-1}$			
	$\frac{1}{p+1+\epsilon}$	$O(\frac{1}{\epsilon\delta}\log(\Delta))$		
	$\frac{1}{2(p+1)}$ (R)	2	$\frac{1}{2+4(p+1)}$ (R)	2 [Bar+15]
p-system + d knapsacks			–	
	$O\left(\frac{1/(p+2d+1)}{\sqrt{\min(m,\rho[\zeta])}}\right)$	2	$O\left(\frac{(p-\epsilon)/(p+1)(2p+2d+1)}{\sqrt{\min(m,\rho[\zeta])}}\right)$	2 [Ch. 5, 6]

that covers all the data points), Berger et al. [BRS89] provided the first distributed solution with an approximation guarantee similar to that of the greedy procedure. Their proposed method closely approximate the greedy algorithm by bucketing utility values by factors of $(1+\epsilon)$, and employing randomized techniques to select appropriate subsets in each bucket in parallel. It requires $O(\log^5 M)$ rounds of MapReduce, where $M \geq n$ is the sum of the sizes of the sets. Blelloch et al. [BPT11] improved their result in terms of the number of rounds required by a MapReduce-based implementation to $O(\log^3 M)$, by utilizing the same bucketing idea, and then selecting maximal nearly independent collection of sets in each bucket.

Very recently, Stergiou et al. [ST15] introduced an efficient distributed algorithm for set cover instances of massive size in $O(\log \Delta)$ rounds of MapReduce, where Δ is the cardinality of the largest set. They adopted a variation of the bucketing approach in order to split the input into chunks. In each bucket, the sets containing a specific number of undiscovered elements are filtered in parallel on each machine. This elements are then merged on the central machine, and the solution is augmented sequentially from the filtered elements with considerable marginal gain.

Another variant of the set cover problem that has received some attention is maximum k-cover (i.e., cover as many elements as possible from the ground set by choosing at most k subsets) for which Chierichetti et al. [CKT10] introduced a distributed solution with a $(1 - 1/e - \epsilon)$ approximation guarantee in $O(poly(\epsilon) \log^3 mn)$. Here m is the number of sets, and n is the size of the ground set. They use similar bucketing idea of [BRS89], but benefit from a randomized method for selection of sets in each bucket. However, the required number of passes over the data makes it impractical for large datasets.

In Chapter 6 of this thesis, we will present the first efficient algorithms to address the general distributed submodular cover problem and propose an algorithm DisCover for approximately solving it. Subsequently, in Chapter 7, we will propose a fast distributed algorithm, FastCover, that truly scales to massive data and produces a solution that is competitive with that of the greedy algorithm.

2.3.2 Streaming Algorithms

Another natural approach to scale up submodular optimization, is to use streaming algorithms. In fact, in applications where data arrives at a pace that does not allow even storing it, this is the only viable option. This approach not only avoids the need for vast amounts of random-access memory but also provides predictions in a timely manner based on the data seen so far, facilitating real-time analytics.

Streaming submodular maximization. The first algorithm, Stream-Greedy, for submodular maximization on data streams is presented by Gomes and Krause [GK10]. Their multi-pass algorithm provides a $1/2 - \epsilon$ approximation guarantee with $O(k)$ memory under strong assumptions on the way the data is generated. However, their approach makes strong assumptions about the way the data stream is generated, and unless their assumptions are met, it is fairly easy to construct examples where the performance of their algorithm degrades quickly when compared to the optimum solution. Furthermore, the update time (computational cost to process one data point) of their approach is $\Omega(k)$, which is prohibitive for large k.

The work of Kumar et al. [Kum+13] claims a multi-pass and a single-pass streaming algorithm. The claimed guarantees for the single pass algorithm depend on the maximum increase in the objective any element can offer (Thm. 27, [Kum+13]), while the multi-pass algorithm has a memory requirement depending on the data size n.

Badanidiyuru et al. [Bad+14] proposed the first efficient streaming algorithm with constant factor $1/2 - \epsilon$ approximation guarantee to the optimum solution, requiring only a single pass through the data, and memory $k \log(k)/\epsilon$ independent of data size. The idea is to update the estimation of the optimal solution while receiving new elements, and construct multiple solutions in parallel. New elements are added to each solution if they marginal value is above the corresponding threshold for that solution.

Chakrabarti el al. [CK14] developed a single pass algorithm, still for maximizing a monotone submodular function, with $1/4p$ approximation guarantee for handling more general constraints such as intersections of p matroids. To deal with the constraints, they consider deleting elements with smaller marginal gains from the solution when adding a new element is infeasible. The required memory is unbounded and increases polylogarithmically with the size of the data. They further provided a better approximation guarantee, namely, $1/(p+1+\epsilon)$, at the cost of $O(\epsilon^{-3} \log p)$ passes over the stream.

Finally, Chekuri et al. [CGQ15] presented deterministic and randomized algorithms for maximizing monotone and non-monotone submodular functions subject to a broader range of constraints, namely p-matchoids. They combine the idea of estimating the value of the optimal solution on the fly [Bad+14], and switching the new element with a candidate set of elements in the solution by comparing their marginal gain, when adding a new element is infeasible [CK14]. For maximizing a monotone submodular function, their proposed method gives a $1/4p$ approximation using $O(k \log k/\epsilon^2)$ memory (k is the size of the largest feasible solution). For non-monotone functions, they provide a deterministic $1/(9p+1)$ approximation using the $1/(p+1)$ offline approximation of [NWF78b] under a p-matchoid constraint. Their randomized algorithm uses $O(k \log k/\epsilon^2)$ memory and $O(pk^2 \log k/\epsilon^2)$ update time, and provides a $1/(4p+1/\tau_p)$ approximation in expectation, where $\tau_p = (1-\epsilon)(2-o(1))/(ep)$ [FNS11a] is the approximation guarantee for maximizing a non-negative submodular function in the offline setting. Table 2.3 summarizes the approximation guarantees, memory and update time of the single pass streaming algorithms.

Very recently, submodular optimization over sliding window model –where we are interested in a solution that considers only the last W items– has been studies. In particular, for monotone submodular maximization under cardinality constraint, Epasto

Table 2.3: *Approximation guarantees, memory and update time for single-pass monotone and non-monotone streaming submodular maximization under different constraints. Bounds for randomized algorithms that hold in expectation are marked (R). α, M, T is the approximation guarantee, memory, and update time of the streaming monotone submodular maximization under a collection of independent systems and d knapsack constraints.*

	Constraint	Approximation	Memory	Update time	
Monotone	Cardinality	$1/2-\epsilon$	$k/\epsilon \log(n\delta)$	?	[Kum+13]
		$1/2-\epsilon$	$O(k\log(k)/\epsilon)$	$O(\log(k)/\epsilon))$	[Bad+14]
	p matroid	$1/4p$	$O((n\log(n))$	$\Omega(np)$	[CK14]
	p matchoid	$1/4p$	$O(\frac{k\log(k)}{\epsilon^2})$	$O(\frac{pk\log(k)}{\epsilon^2})$	[CGQ15]
Non-negative	cardinality	$\frac{(1-\epsilon)}{(2+e)}$ (R)	$O(\frac{k\log(k)}{\epsilon^2})$	$O(\frac{pk^2\log(k)}{\epsilon^2})$	
	1 matroid	$\frac{(1-\epsilon)}{(4+e)}$ (R)			
	p matroid	$\frac{(1-\epsilon)(p-1)}{5p^2-4p+\epsilon}$ (R)			
	p matchoid	$\frac{(1-\epsilon)(2-o(1))}{(8+e)p}$ (R)			
	Independence systems + d knapsack	$\frac{(1-\epsilon)}{1+\frac{2}{\sqrt{\alpha}}+\frac{1}{\alpha}+2d(1+\sqrt{\alpha})}$	$O(\frac{M\log(k)}{\epsilon\sqrt{\alpha}})$	$O(\frac{T\log(k)}{\epsilon\sqrt{\alpha}})$	[Ch. 9]

et al. [Epa+16] proposed a $1/3-\epsilon$ approximation algorithm with $O(k\log^2(k\Phi)/\epsilon^2)$ memory, where Φ is the ratio between maximum and minimum values of the submodular function. In parallel, [CNZ16] proposed a $1/(4p+(1+\epsilon)16p^2)$-approximation algorithm using $O(k/\epsilon \log M)$ space for monotone submodular maximization subject to p-matroid constraints, where M is an upperbound on the value of the optimum solution.

In Chapter 9, we will develop the first single pass streaming algorithm, STREAMING LOCAL SEARCH, for maximizing a general submodular function subject to intersection of a collection of independence systems and d knapsack constrains. Subsequently, in Chapter 10, we will introduce the dynamic deletion-robust streaming submodular maximization problem, when elements selected for the summary can be deleted at any time. We will then develop the first resilient streaming algorithm, called ROBUST-STREAMING, with a constant factor approximation guarantee for this problem. Unlike the centralized deletion-robust approaches introduced in Section 2.2.3 that aim to construct solutions that are still good representatives after deletions, ROBUST-STREAMING is able to *update* the summary immediately after each deletion in the streaming setting.

Submodular secretary. There is further related work on the *submodular secretary problem* [Gup+10a; BHZ10]. While also processing elements in a stream, these approaches are different in two important ways: (i) they work in the stronger model where they must either commit to or permanently discard newly arriving elements; (ii) they require *random* arrival of elements, and have a worse approximation ratio (≤ 0.1 vs. $1/2 - \epsilon$). If elements arrive in arbitrary order, performance can degrade arbitrarily. Some approaches also require large (i.e., $O(n)$) memory [Gup+10a].

2.3.3 Fast Centralized Algorithms

Previously, we have seen that centralized algorithms for submodular maximization are computationally intractable for very large datasets. Moreover, even smaller sized data can be prohibitive, particularly when evaluating the function itself is computationally costly [LB11a; Wei+13]. Hence, developing fast centralized algorithms for submodular maximization has attracted increasing attention.

Constrained monotone submodular maximization. Motivated by extremely large-scale machine learning problems, there have been recent efforts to further accelerate monotone submodular maximization algorithms.

In particular, Wei et al. [WIB14] proposed a multistage algorithm, Multi-Greedy, that tries to decrease the running time of Lazy-Greedy by approximating the underlying submodular function with a set of (sub)modular functions that can be potentially evaluated less expensively. This approach is effective only for those submodular functions that can be easily decomposed and approximated. Badanidiyuru and Vondrak [BV14] proposed a variant of the continuous greedy algorithm, that provides an $O(n^2/\epsilon^4 \log^2(n/\epsilon))$-time $(1 - 1/e - \epsilon)$-approximation for a matroid constraint, and a $O(n^2(1/\epsilon \log n)^{poly(1/\epsilon)})$-time $(1 - 1/e - \epsilon)$-approximation for a knapsack constraint. Previous variants and alternative techniques were known to use at least $\tilde{O}(n^4)$ oracle calls.

In Chapter 12, we will provide a linear-time algorithm for maximizing a general monotone submodular function subject to a cardinality constraint. We show that our

Table 2.4: *Comparison of running times and approximation ratios for non-monotone submodular maximization under different constraints.*

Constraint	Previous		new [Ch. 13]	
	Approximation	Complexity	Approximation	Complexity
p-system + d knapsacks	-	-	$\frac{(1+\epsilon)(p+1)(2p+2d+1)}{p}$	$O(\frac{nrp\log(n)}{\epsilon})$
1-matroid + d knapsacks	e+small const.$+\epsilon$ [Buc+; CVZ]	$\text{poly}(n)\cdot\exp(d,\epsilon)$		
p-matroid + d knapsacks	$p/0.19+\epsilon$ [CVZ]	$\text{poly}(n)\cdot\exp(p,d,\epsilon)$		
p-system	$(p+1)(3p+3)/p$ [Gup+10b]	$O(nrp)$	$\frac{(p+1)(2p+1)}{p}$	$O(nrp)$
p-matroid	$p+1+\frac{1}{(p-1)}+\epsilon$ [LSV10]	$\text{poly}(n)\cdot\exp(p,\epsilon)$		

randomized algorithm, Stochastic-Greedy, can achieve a $(1-1/e-\epsilon)$ approximation guarantee, in expectation, to the optimum solution in time linear in the size of the data and independent of the cardinality constraint. Stochastic-Greedy can be easily integrated into existing distributed methods, e.g. for solving the sub-problems in each stage of Multi-Greedy to develop a faster multistage method, or in our distributed framework that will be described in Chapter 5. In general, any (distributed) algorithm that uses Lazy-Greedy as a sub-routine, can directly benefit from our method and provide even more efficient large-scale algorithmic frameworks.

Constrained non-monotone submodular maximization. While maximizing monotone submodular functions has been applied to many machine learning applications, the problem of maximizing non-monotone submodular functions has not found as many applications. Part of the reason is that the existing algorithms for handling generic constraints such as both matroid and knapsack constraints are very slow. A body of work [FNS11b; CVZ; Gup+10b; LSV10; FMV11; Buc+15] has found algorithms with good approximation ratios, albeit with running times of very high polynomial. A comparison can be found in Table 2.4. In particular, for 1-matroid and d knapsack constraints, a $(e+\epsilon)$-approximation follows from applying results from [FNS11b] to contention resolution techniques from [CVZ]. Feldman, Naor, and Schwartz [FNS11b] don't formalize this as a theorem but state it in the introduction. Gupta et al. [Gup+10b] and Gupta, Nagarajan, and Ravi [GNR15] further showed a $O(p)$ approximation for p-

matroid and d knapsack constraints. This result is summarized in Table 1 of [CVZ] as a $(p/0.19+\epsilon)$-approximation. For a p-system constraint without any knapsack constraint, the approximation ratio of $(p+1)(3p+3)/p$ can be obtained by substituting $\alpha = 1/2$ approximation for unconstrained maximization into Theorem 3.3 of [Gup+10b]. Finally, p-matroid approximation ratio of $p+1+1/(p-1)+\epsilon$ is provided by Theorem 4 of [LSV10].

It is worth mentioning that in addition to the above results, recently Chekuri et al. [CJV15] developed a continuous-time framework that provides a $(1-1/e-\epsilon)$ approximation for maximizing a submodular function under packing constraints. Although their algorithm gives a $O(n^2)$ runtime for the fractional solution to the multi-linear extension, converting the fractional solution to an intergral solution requires enumerating sets of size poly$(1/\epsilon)$. This results in a run time of $n^{\text{poly}(1/\epsilon)}$, which is impractical for most real-world scenarios. In Chapter 13, we will develop the first practical algorithm, Fantom, for maximizing non-monotone submodular functions with very generic p-system and d knapsack constraints.

3 Applications of Large Scale Submodular Summarization

In this chapter, we discuss concrete problem instances, with their corresponding submodular objective functions f, where the size of the datasets often requires a large-scale solution for the underlying submodular maximization.

3.1 Nonparametric Learning

Nonparametric learning (i.e., learning of models whose complexity may depend on the dataset size n) is notoriously hard to scale to large datasets. A concrete instance of this problem arises from training Gaussian processes or performing MAP inference in Determinantal Point Processes, as considered below. Similar challenges arise in many related learning methods, such as training kernel machines, when attempting to scale them to large datasets.

3.1.1 Active Set Selection in Sparse Gaussian Processes (GPs).

Formally a GP is a joint probability distribution over a (possibly infinite) set of random variables $\mathbf{X}_V$, indexed by the ground set V, such that every (finite) subset $\mathbf{X}_S$ for

$S = \{e_1, \dots, e_s\}$ is distributed according to a multivariate normal distribution. More precisely, we have

$$P(\mathbf{X}_S = \mathbf{x}_S) = \mathcal{N}(\mathbf{X}_S; \mu_S, \Sigma_{S,S}),$$

where $\mu = (\mu_{e_1}, \dots, \mu_{e_s})$ and $\Sigma_{S,S} = [\mathcal{K}_{e_i,e_j}]$ are prior mean and covariance matrix, respectively. The covariance matrix is parametrized via a positive definite kernel $\mathcal{K}(\cdot, \cdot)$. As a concrete example, when elements of the ground set V are embedded in a Euclidean space, a commonly used kernel in practice is the squared exponential kernel defined as follows:

$$\mathcal{K}(e_i, e_j) = \exp(-||e_i - e_j||_2^2 / h^2).$$

Gaussian processes are commonly used as priors for nonparametric regression. In GP regression, each data point $e \in V$ is considered a random variable. Upon observations $\mathbf{y}_A = \mathbf{x}_A + \mathbf{n}_A$ (where $\mathbf{n}_A$ is a vector of independent Gaussian noise with variance σ^2), the predictive distribution of a new data point $e \in V$ is a normal distribution $P(\mathbf{X}_e \mid \mathbf{y}_A) = \mathcal{N}(\mu_{e|A}, \Sigma^2_{e|A})$, where mean $\mu_{e|A}$ and variance $\sigma^2_{e|A}$ are given by

$$\mu_{e|A} = \mu_e + \Sigma_{e,A}(\Sigma_{A,A} + \sigma^2 \mathbf{I})^{-1}(\mathbf{x}_A - \mu_A), \tag{3.1.1}$$
$$\sigma^2_{e|A} = \sigma^2_e - \Sigma_{e,A}(\Sigma_{A,A} + \sigma^2 \mathbf{I})^{-1}\Sigma_{A,e}. \tag{3.1.2}$$

Evaluating (3.1.1) and (3.1.2) is computationally expensive as it requires solving a linear system of $|A|$ variables. Instead, most efficient approaches for making predictions in GPs rely on choosing a small–so called *active*–set of data points. For instance, in the Informative Vector Machine (IVM) one seeks a set S such that the *information gain*, defined as

$$f(S) = I(\mathbf{Y}_S; \mathbf{X}_V) = H(\mathbf{X}_V) - H(\mathbf{X}_V|\mathbf{Y}_S) = \frac{1}{2}\log\det(\mathbf{I} + \sigma^{-2}\Sigma_{S,S}) \tag{3.1.3}$$

is maximized. It can be shown that this choice of f is monotone submodular [KG05b].

3.1.2 Inference for Determinantal Point Processes.

Determinantal Point Processes (DPPs) [Mac75] are distributions over subsets with a preference for diversity, i.e., there is a higher probability associated with sets containing dissimilar elements. They have been successfully applied to video summarization

[Gon+14], as well as problems like document summarization [KT+12] and information retrieval [GKT12a].

Formally, a point process $\mathcal{P}$ on a set of items $V = \{1, 2, ..., N\}$ is a probability measure on 2^V (the set of all subsets of V). $\mathcal{P}$ is called *determinantal point process* if for every $S \subseteq V$ we have:

$$\mathcal{P}(S) \propto \det(L_S), \tag{3.1.4}$$

where L is a positive semidefinite kernel matrix, and $L_S \equiv [L_{ij}]_{i,j \in S}$, is the restriction of L to the entries indexed by elements of S (we adopt that $\det(L_\emptyset) = 1$). The normalization constant can be computed explicitly from the following equation

$$\sum_S \det(L_S) = \det(\mathbf{I} + L), \tag{3.1.5}$$

where I is the $N \times N$ identity matrix. Intuitively, the kernel matrix determines which items are similar and therefore less likely to appear together.

To find the most diverse and informative subset, we need to find $\arg\max_{S \in \mathcal{I}} \det(L_S)$, where $\mathcal{I} \subset 2^V$ is a given family of feasible solutions. This problem is NP-hard, as the total number of possible subsets is exponential [KLQ95]. However, the objective function is log-submodular, i.e. $f(S) = \log\det(L_S)$ is a (non-monotone) submodular function [KT+12]. Hence, MAP inference in large DPPs is another application of submodular maximization.

3.2 Exemplar Based Clustering

Suppose we wish to select a set of exemplars, that best represent a massive dataset. One approach for finding such exemplars is solving the k-medoid problem [KR09], which aims to minimize the sum of pairwise dissimilarities between exemplars and elements of the dataset. More precisely, let us assume that for the dataset V we are given a nonnegative function $l : V \times V \to \mathbb{R}$ (not necessarily assumed symmetric, nor obeying the triangle inequality) such that $l(\cdot, \cdot)$ encodes dissimilarity between elements of the underlying set V. Then, the cost function for the k-medoid problem is:

$$L(S) = \frac{1}{|V|} \sum_{v \in V} \min_{e \in S} l(e, v). \tag{3.2.1}$$

Finding the subset

$$S^* = \arg\min_{S \in \mathcal{I}} L(S)$$

that minimizes the cost function (3.2.1) is NP-hard. However, by introducing an auxiliary element e_0, a.k.a. phantom exemplar, we can turn L into a monotone submodular function [KG10]

$$f(S) = L(\{e_0\}) - L(S \cup \{e_0\}). \tag{3.2.2}$$

In words, f measures the decrease in the loss associated with the set S versus the loss associated with just the auxiliary element. We begin with a phantom exemplar and try to find the active set that together with the phantom exemplar reduces the value of our loss function more than any other set. Technically, any point e_0 that satisfies the following condition can be used as a phantom exemplar:

$$\max_{v' \in V} l(v, v') \leq l(v, e_0), \quad \forall v \in V \setminus S.$$

This condition ensures that once the distance between any $v \in V \setminus S$ and e_0 is greater than the maximum distance between elements in the dataset, then $L(S \cup \{e_0\}) = L(S)$. As a result, maximizing f (a monotone submodular function) is equivalent to minimizing the cost function L.

3.3 Dominating Sets in Social Networks

Probably the easiest way to define the influence of a subset of users on other members of a social network is by the dominating set problem. Here, we assume that there is a graph $G = (V, E)$ where V and E indicate the set of nodes and edges, respectively. Let $\mathcal{N}(S)$ denote the neighbors of S. Then, we define the coverage size of S by

$$f(S) = |\mathcal{N}(S) \cup S|, \tag{3.3.1}$$

It is easy to see that f is a monotone submodular function. The dominating set is the problem of choosing a small subset of nodes S such that it covers a desired fraction of $|V|$ [Lat+11].

3.4 Sensor Placement

When monitoring spatial phenomena, we want to deploy a limited number of sensors in an area in order to quickly detect contaminants. Thus, the problem would be to select a subset of all possible locations $S \subseteq V$ to place sensors. Consider a set of intrusion scenarios I where each scenario $i \in I$ defines the introduction of a contaminant at a specified point in time. For each sensor $s \in S$ and scenario i, the detection time, $T(s,i)$, is defined as the time it takes for s to detect i. If s never detects i, we set $T(s,i) = \infty$. For a set of sensors S, detection time for scenario i could be defined as

$$T(S,i) = \min_{s \in S} T(s,i). \tag{3.4.1}$$

Depending on the time of detection, we incur penalty $\pi_i(t)$ for detecting scenario i at time t. Let $\pi_i(\infty)$ be the maximum penalty incurred if the scenario i is not detected at all. Then, the penalty reduction for scenario i can be defined as

$$R(S,i) = \pi_i(\infty) - \pi_i(T(S,i)). \tag{3.4.2}$$

Having a probability distribution over possible scenarios, we can calculate the expected penalty reduction for a sensor placement S as

$$f(S) = \sum_{i \in I} P(i) R(S,i). \tag{3.4.3}$$

This function is montone submodular [Kra+08a].

3.5 Summarizing Image Collections

Given a collection of images, one might be interested in finding a subset that best summarizes and represents various aspects of the collections with minimal redundancy. This problem has recently been addressed via submodular maximization. Fidelity and diversity are two general properties that characterize good image collection summarizes, and can be naturally characterized by submodularity. Tschiatschek et al. [Tsc+14] designed several submodular objectives $F_1, \ldots, F_l$, which quantify different characteristics that good summaries should have. These functions include:

Facility Location that quantifies the coverage of a subset of elements $S \subseteq V$ by the sum of similarities $s(.,.)$ between elements of the dataset $i \in V$ and their closest element in the summary $j \in S$.

$$F(S) = \sum_{i \in V} \max_{j \in S} s(i,j). \tag{3.5.1}$$

Sum Coverage that finds a representative subset that is most similar to the whole data, by maximizing the sum between pairwise similarities $s(.,.)$ between all elements of the summary and the data elements.

$$F(S) = \sum_{i \in V} \sum_{j \in S} s(i,j). \tag{3.5.2}$$

Thresholded Sum Coverage that thresholds the inner sum to keep any element from being overly covered by S

$$F(S) = \sum_{i \in V} \min \left(\sum_{j \in S} s(i,j), \alpha \sum_{i \in V} s(i,j) \right). \tag{3.5.3}$$

Feature Functions that use the notion of visual words to measure the coverage of visual features by the images in summary S:

$$F(S) = \sum_{w \in W} g(\sum_{i \in V} b_{i,w}), \tag{3.5.4}$$

where $b_{i,w}$ is a bag-of-words representation of image i indexed by the set of visual words W, and $g(\cdot)$ is a monotone non-decreasing concave function. A visual bag-of-words vocabulary can be constructed based on e.g. SIFT descriptors [Low99], color descriptors [WDJ11], and raw image patches [FPZ03].

Clustered Diversity that rewards selecting a diverse summary from different clusters $P_1, P_2, \cdots, P_k$ obtained by some clustering algorithm.

$$F(S) = \sum_{j \in k} g(S \cap P_i), \tag{3.5.5}$$

where $g(.)$ is a monotone submodular function.

Penalty based diversity that penalizes the sum of similarities between elements of the summary S.

$$F(S) = -\sum_{i \in S} \sum_{j \in S, j \geq i} s(i,j). \tag{3.5.6}$$

Then, they optimize a weighted combination of such functions

$$f(S) = \sum_{i=1}^{l} w_i F_i(S), \tag{3.5.7}$$

where weights are non-negative, i.e., $w_i \geq 0$, and learned via a large-margin structured prediction. Here, computing the weighted combination of such large number of functions makes the function evaluation considerably expensive.

3.6 Movie Recommendation

Consider a movie recommender system that has to provide a short list of representative movies according to the users' interests. To this end, we represent each movie by a vector consist of users' ratings. Such a representation can be easily obtained by using existing low-rank matrix completion techniques [CR12] that provide a complete rating matrix $M_{k \times n}$ based on few ratings of k users for n movies in the ground set V. By forming M, we can measure the similarity $s_{i,j}$ between movies i and j through the inner product between the corresponding columns. Note that a movie can be a member of different categories (e.g., a movie can be both drama and comedy). We denote by $G(i)$ the genres of movie $i \in V$. We also let V_g denote the set of movies from genre g. Clearly, two genres g, g' may overlap, i.e., $V_g \cap V_{g'} \neq \emptyset$. A sensible submodular utility function that we can use in order to score the quality of the selected movies is

$$f(S) = \sum_{i \in V} \sum_{j \in S} s_{i,j} - \lambda \sum_{i \in S} \sum_{j \in S} s_{i,j}, \tag{3.6.1}$$

for some $0 \leq \lambda \leq 1$. Note that for $\lambda = 1$ the above function is the cut-function. This utility function is non-negative and non-monotone. The first term is the sum-coverage function (to capture coverage) and the second term penalizes similarity within S (to capture diversity). Such functions have been previously used in document [LB11b] and scene summarization [SSS07]. Another possible way to compute the similarity between a movie i and the dataset V is to consider only movies which have a common genre with i as follows

$$f(S) = \sum_{j \in S} \sum_{g \in G(j)} \sum_{i \in V_g} s_{i,j} - \lambda \sum_{j \in S} \sum_{g \in G(j)} \sum_{i \in V_g \cap S} s_{i,j}, \tag{3.6.2}$$

which is again a non-negative and non-monotone submodular function.

3.7 Diversified Image summarization

Here, we have a collection of images V, and we want to find a concise and diverse summary of all the images. For the utility function, we can use

$$f(S) = \sum_{i \in V} \max_{j \in S} d_{i,j} - \frac{1}{|V|} \sum_{i \in S} \sum_{j \in S} d_{i,j}, \tag{3.7.1}$$

where $d_{i,j}$ determines the similarity of image i to image j. There are many ways to determine the similarity between images such as cosine similarity or a distance metric. The first term is the facility location objective function (for coverage) and the second term is a dispersion function (for diversity). Facility location (see Eq. 3.5.1) has been extensively used for image summarization [DF07b; GK10]. The above submodular function is non-negative and non-monotone.

3.8 Revenue Maximization with Multiple Products

In this application, we consider the revenue maximization on a social network $G = (V, W)$ when multiple products from a basket Q that can be offered to each user $i \in V$. Here, we assume that $W = [w_{ij}]$ represents the weight of edges. The goal is to offer for free or advertise some of the products to a set of users $S \subseteq V$ such that through their influence on others the revenue increases. Following [HMS08] model, a user's value v_i^q for a product q is determined by the set of other users that own the product, i.e., $v_i^q : 2^V \rightarrow \mathbb{R}^+$. The function $v_i^q(S)$ usually takes a concave graph model [HMS08; Bab+13], i.e., for all $i \in V$ and $S \subseteq V \setminus \{i\}$, we have

$$v_i^q(S) = g_i^q\left(\sum_{j \in S \cup \{i\}} w_{ij}\right) \tag{3.8.1}$$

where $g_i^q : \mathbb{R}^+ \to \mathbb{R}^+$ is a concave function (depending on the product $q \in Q$) and w_{ij} are chosen independently from a distribution μ. The revenue of a set S for a product $q \in Q$ is defined as

$$f^q(S) = \sum_{i \in V \setminus S} v_i^q(S) = \sum_{i \in V \setminus S} g_i^q\Big(\sum_{j \in S \cup \{i\}} w_{ij}\Big). \tag{3.8.2}$$

Note that f^q is a non-monotone submodular function. Each product $q \in Q$ can be advertised to a potentially different subset $S^q \subseteq V$. The *total revenue*, that we try to maximize, is $\sum_{q \in Q} f^q(S^q)$ which is again a non monotone submodular function.

3.9 Other Examples

Numerous other real world problems in machine learning can be modeled as maximizing a monotone submodular function subject to appropriate constraints (e.g., cardinality, matroid, knapsack). To name a few, specific applications that have been considered range from efficient content discovery for web crawlers and multi topic blog-watch [CKT10], over document summarization [LB11b] and speech data subset selection [Wei+13], to outbreak detection in social networks [Les+07], online advertising and network routing [DVV03], and inferring network of influence [GRLK10]. In all such examples, the size of the dataset (e.g., number of webpages, size of the corpus, number of blogs in the blogosphere, number of nodes in social networks) is massive, thus our developed methods offers a scalable approach, in contrast to the standard centralized algorithms, for such problems.

Part II

Distributed Algorithms

4 Overview of part II

In Part II of this Thesis, we consider distributed methods for submodular summarization. We start by discussing distributed submodular maximization for identifying representative elements in large datasets, and then we turn our attention to the problem of distributed submodular cover for succinctly summarizing massive data.

Distributed submodular maximization. In section 5, we consider the problem of submodular function maximization in a distributed fashion. We develop a simple, two-stage protocol GreeDi, that is easily implemented using MapReduce style computations. We theoretically analyze our approach, and show that under certain natural conditions, performance close to the centralized approach can be achieved. We begin with monotone submodular maximization subject to a cardinality constraint, and then extend this approach to obtain approximation guarantees for (not necessarily monotone) submodular maximization subject to more general constraints including matroid or knapsack constraints. In our extensive experiments, we demonstrate the effectiveness of our approach on several applications, including sparse Gaussian process inference and exemplar based clustering on tens of millions of examples using Hadoop.

Distributed submodular cover. So far, we have considered the problem of identifying representative elements from large datasets. However, in many applications we need to find a subset, ideally as small as possible, that well represents a massive dataset. I.e., its corresponding utility, measured according to a suitable utility function, should be comparable to that of the whole dataset. In Chapter 6, we formalize this challenge as a submodular cover problem, and develop the first distributed algorithm – DisCover – for submodular set cover that is easily implementable using MapReduce-style computations. We theoretically analyze our approach, and present approximation guarantees for the solutions returned by DisCover. We also study a natural trade-off between the communication cost and the number of rounds required to obtain such a solution. In our extensive experiments, we demonstrate the effectiveness of our approach on several applications, including active set selection, exemplar based clustering, and vertex cover on tens of millions of data points using Spark.

Fast distributed submodular cover for public-private summarization. In chapter 7, we introduce the public-private framework of data summarization motivated by privacy concerns in personalized recommender systems and online social services. Such systems have usually access to massive data generated by a large pool of users. A major fraction of the data is public and is visible to (and can be used for) all users. However, each user can also contribute some private data that should not be shared with other users to ensure her privacy. The goal is to provide a succinct summary of massive dataset, ideally as small as possible, from which customized summaries can be built for each user, i.e. it can contain elements from the public data (for diversity) and users' private data (for personalization).

To formalize the above challenge, we model the data summarization targeted to each user as an instance of a submodular cover problem. However, for a large pool of users, it is too time consuming to find such summaries separately. Instead, we develop a fast distributed algorithm for submodular cover, FastCover, that provides a succinct summary in *one shot* and for *all users*. We show that the solution provided by FastCover is competitive with that of the centralized algorithm with the number of rounds that is exponentially smaller than state of the art results. Moreover, we have implemented FastCover with Spark to demonstrate its practical performance on a number of concrete applications, including personalized location recommendation, personalized movie recommendation, and dominating set on tens of millions of data points and varying number of users.

Summary of contributions. The key contributions of this part of the Thesis are:

1. We consider submodular summarization in a distributed setting, where the size of the data doesn't allow for centralized solution. We develop novel, efficient approximation algorithms:
    - GreeDi, a two-stage distributed submodular maximization framework for identifying representative elements from massive data,
    - DisCover, a distributed submodular cover algorithm for succinctly summarizing massive data, and
    - FastCover, a faster distributed submodular cover algorithm for public-private data summarization.
2. We theoretically analyze our approaches, and provide approximation guarantees for the quality of the distributed solutions.
3. For distributed submodular cover algorithms, we also study the natural trade-off between the communication cost and the number of rounds required to provide the theoretical guarantees.
4. We implement our distributed algorithms with Hadoop and Spark, and demonstrate the performance of our algorithms on several real-world problems, including
    - summarizing 80,000,000 Tiny Images,
    - summarizing 45,811,883 user visits from the Featured Tab of the Today Module on Yahoo! Front Page,
    - finding dominating set in Friendster social network with 65,608,366 nodes and 1,806,067,135 edges, and
    - movie recommendation based on 20,000,263 users' ratings from 138,493 users of the MovieLens database,

    and show that performance close to the centralized approach can be achieved.

5 Distributed Submodular Maximization

As we discussed in Part I, the simple greedy algorithm produces solutions competitive with the optimal (intractable) solution [NWF78a], for maximizing a submodular function. However, such greedy algorithms or their accelerated variants [e.g., lazy evaluations; Min78; Les+07, and stochastic methods; Chapter 12] do not scale well when the dataset is massive. As data volumes in modern applications increase faster than the ability of individual computers to process them, we need to look at ways to adapt our computations using parallelism.

MapReduce [DG08] is arguably one of the most successful programming models for reliable and efficient parallel computing. It works by distributing the data to independent machines: *map* tasks redistribute the data for appropriate parallel processing and the output then gets sorted and processed in parallel by *reduce* tasks.

To perform submodular optimization in MapReduce, we need to design suitable parallel algorithms. The greedy algorithms that work well for centralized submodular optimization do not translate easily to parallel environments. The algorithms are inherently sequential in nature, since the marginal gain from adding each element is dependent on the elements picked in previous iterations. This mismatch makes it inefficient to apply classical algorithms directly to parallel setups.

In this chapter, we develop a distributed procedure for maximizing submodular functions, that can be easily implemented in MapReduce. Our strategy is to partition the data (e.g., randomly) and process it in parallel. In particular:

- We present a simple, parallel protocol, called GreeDi for distributed submodular maximization subject to cardinality constraints. It requires minimal communication, and can be easily implemented in MapReduce style parallel computation models.
- We show that under some natural conditions, for large datasets the quality of the obtained solution is provably competitive with the best centralized solution.
- We discuss extensions of our approach to obtain approximation algorithms for (not-necessarily monotone) submodular maximization subject to more general types of constraints, including matroid and knapsack constraints.
- We implement our approach for exemplar based clustering and active set selection in Hadoop, and show how our approach allows to scale exemplar based clustering and sparse Gaussian process inference to datasets containing tens of millions of points.
- We extensively evaluate our algorithm on several machine learning problems, including exemplar based clustering, active set selection and finding cuts in graphs, and show that our approach leads to parallel solutions that are very competitive with those obtained via centralized methods (98% in exemplar based clustering, 97% in active set selection, 90% in finding cuts).

This chapter is organized as follows. In Section 5.1, we formalize the distributed submodular maximization problem under cardinality constraints, and introduce naive approaches toward solving the problem. We subsequently present our GreeDi algorithm in Section 5.2, and prove its approximation guarantees. We then consider maximizing a submodular function subject to more general constraints in Section 5.3. We also present computational experiments on very large datasets in Section 5.4, showing that in addition to its provable approximation guarantees, our algorithm provides results close to the centralized greedy algorithm. We conclude in Section 5.5.

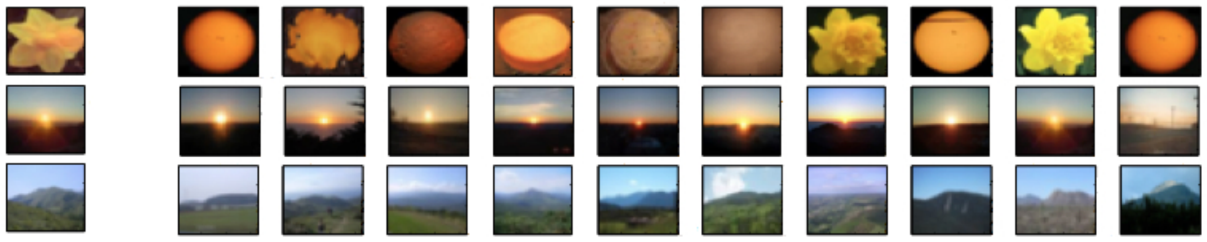


Figure 5.1: *Cluster exemplars (left column) discovered by our distributed algorithm GreeDi described in Section 5.2 applied to the Tiny Images dataset [TFF08], and a set of representatives from each cluster.*

5.1 Submodular Maximization

In this section, we first review how to greedily maximize submodular functions. We then describe the *distributed submodular maximization* problem, the focus of this chapter. Finally, we discuss two naive approaches towards solving this problem.

5.1.1 Greedy Submodular Maximization

Suppose that we have a large dataset of images, e.g. the set of all images on the Web or an online image hosting website such as Flickr, and we wish to retrieve a subset of images that best represents the visual appearance of the dataset. Collectively, these images can be considered as *exemplars* that *summarize* the visual categories of the dataset as shown in Figure 5.1.

One way to approach this problem is to formalize it as the *k-medoid* problem. Given a set $V = \{e_1, e_2, \dots, e_n\}$ of images (called ground set) associated with a (not necessarily symmetric) dissimilarity function, we seek to select a subset $S \subseteq V$ of at most k exemplars or cluster centers, and then assign each image in the dataset to its least dissimilar exemplar. If an element $e \in V$ is assigned to exemplar $v \in S$, then the cost associated with e is the dissimilarity between e and v. The goal of the k-medoid problem is to choose exemplars that minimize the sum of dissimilarities between every data point $e \in V$ and its assigned cluster center.

Solving the k-medoid problem optimally is NP-hard, however we can transform this problem, and many other summarization tasks, to the problem of maximizing a monotone submodular function subject to a cardinality constraint (*c.f.* Section 3)

$$\max_{S \subseteq V} f(S) \quad \text{s.t.} \quad |S| \leq k. \tag{5.1.1}$$

For any non-negative and monotone submodular function f, the greedy heuristic that iteratively augments the current solution with an element of maximum incremental value

$$v^* = \arg\max_{v \in V \setminus A} f(A \cup \{v\}), \tag{5.1.2}$$

always produces a solution $A^{\text{gc}}[k]$ of size k that achieves at least a constant factor $(1 - 1/e)$ of the optimal solution.

$$f(A^{\text{gc}}[k]) \geq (1 - 1/e) \max_{|A| \leq k} f(A).$$

This result can be easily extended to $f(A^{\text{gc}}[l]) \geq (1 - e^{-l/k}) \max_{|A| \leq k} f(A)$, where l and k are two positive integers [see, KG13].

5.1.2 Distributed Submodular Maximization

In many today's applications where the size of the ground set $|V| = n$ is very large and cannot be stored on a single computer, running the standard greedy algorithm or its variants [e.g., lazy evaluations, Min78; Les+07, and stochastic methods, Chapter 12] in a centralized manner is infeasible. Hence, we seek a solution that is suitable for large-scale parallel computation. The greedy method described above is in general difficult to parallelize, since it is inherently sequential: at each step, only the object with the highest marginal gain is chosen and every subsequent step depends on the preceding ones.

Concretely, we consider the setting where the ground set V is very large and cannot be handled on a single machine, thus must be distributed among a set of m machines. While there are several approaches towards parallel computation, in this chapter we consider the following model that can be naturally implemented in MapReduce. The computation proceeds in a sequence of rounds. In each round, the dataset is distributed to m machines. Each machine i carries out computations independently in parallel on its local data. After all machines finish, they synchronize by exchanging a limited amount of data (of size polynomial in k and m, but independent of n). Hence, any distributed algorithm in this model must specify: 1) how to distribute V among the m machines, 2) which algorithm should run on each machine, and 3) how to communicate and merge the resulting solutions.

In particular, the distributed submodular maximization problem requires the specification of the above steps in order to implement an approach for submodular maximization. More precisely, given a monotone submodular function f, a cardinality constraint k, and a number of machines m, we wish to produce a solution $A^{\mathrm{d}}[m,k]$ of size k such that $f(A^{\mathrm{d}}[m,k])$ is competitive with the optimal *centralized* solution $\max_{|A|\leq k, A\subseteq V} f(A)$.

While in principle multiple rounds of computation can be realized, in practice, expensive synchronization is required after each round. Hence, we are interested in distributed algorithms that require few rounds of computation.

5.1.3 Naive Approaches to Distributed Submodular Maximization

One way to solve problem 5.1.1 in a distributed fashion is as follows. The dataset is first partitioned (randomly, or using some other strategy) onto the m machines, with V_i representing the data allocated to machine i. We then proceed in k rounds. In each round, all machines–in parallel–compute the marginal gains of all elements in their sets V_i. Next, they communicate their candidate to a central processor, who identifies the globally best element, which is in turn communicated to the m machines. This element is then taken into account for computing the marginal gains and selecting the next elements. This algorithm (up to decisions on how break ties) implements exactly the centralized greedy algorithm, and hence provides the same approximation guarantees on the quality of the solution. Unfortunately, this approach requires synchronization after each of the k rounds. In many applications, k is quite large (e.g., tens of thousands), rendering this approach impractical for MapReduce style computations.

An alternative approach for large k would be to greedily select k/m elements independently on each machine (without synchronization), and then merge them to obtain a solution of size k. This approach that requires only two rounds (as opposed to k), is much more communication efficient, and can be easily implemented using a single MapReduce stage. Unfortunately, many machines may select redundant elements, and thus the merged solution may suffer from diminishing returns. It is not hard to construct examples for which this approach produces solutions that are a factor $\Omega(m)$ worse than the centralized solution.

In Section 5.2, we introduce an alternative protocol GreeDi, which requires little communication, while at the same time yielding a solution competitive with the centralized one, under certain natural additional assumptions.

5.2 The GreeDi Approach for Distributed Submodular Maximization

In this section we present our main results. We first provide our distributed solution GreeDi for maximizing submodular functions under cardinality constraints. We then show how we can make use of the geometry of data inherent in many practical settings in order to obtain strong data-dependent bounds on the performance of our distributed algorithm.

5.2.1 An Intractable, yet Communication Efficient Approach

Before we introduce GreeDi, we first consider an intractable, but communication–efficient two-round parallel protocol to illustrate the ideas. This approach, shown in Algorithm 1, first distributes the ground set V to m machines. Each machine then finds the *optimal* solution, i.e., a set of cardinality at most k, that maximizes the value of f in each partition. These solutions are then merged, and the optimal subset of cardinality k is found in the combined set. We denote this distributed solution by $f(A^{\mathrm{d}}[m,k])$.

As the optimum centralized solution $A^{\mathrm{c}}[k]$ achieves the maximum value of the submodular function, it is clear that $f(A^{\mathrm{c}}[k]) \geq f(A^{\mathrm{d}}[m,k])$. For the special case of selecting a single element $k = 1$, we have $f(A^{\mathrm{c}}[1]) = f(A^{\mathrm{d}}[m,1])$. Furthermore, for *modular* functions f (i.e., those for which f and $-f$ are both submodular), it is easy to see that the distributed scheme in fact returns the optimal centralized solution as well. In general, however, there can be a gap between the distributed and the centralized solution. Nonetheless, as the following theorem shows, this gap cannot be more than $1/\min(m,k)$. Furthermore, this result is tight.

Algorithm 1: Inefficient Distributed Submodular Maximization

```
Input: Set $V$, #of partitions $m$, constraints $k$.
Output: Set $A^{\mathrm{d}}[m,k]$.
1: Partition $V$ into $m$ sets $V_1, V_2, \dots, V_m$.
2: In each partition $V_i$ find the optimum set $A_i^c[k]$ of cardinality $k$.
3: Merge the resulting sets: $B = \cup_{i=1}^m A_i^c[k]$.
4: Find the optimum set of cardinality $k$ in $B$. Output this solution $A^{\mathrm{d}}[m,k]$.
```

Theorem 4. *Let f be a monotone submodular function and let $k > 0$. Then, $f(A^d[m,k])) \geq \frac{1}{\min(m,k)} f(A^c[k])$. In contrast, for any value of m and k, there is a monotone submodular function f and a particular partitioning such that $f(A^c[k]) = \min(m,k) \cdot f(A^d[m,k])$.*

The proofs of all the theorems in this chapter can be found in Appendix A.1. The above theorem fully characterizes the performance of Algorithm 1 in terms of the best centralized solution. In practice, we cannot run Algorithm 1, since there is no efficient way to identify the optimum subset $A_i^c[k]$ in set V_i, unless P=NP. In the following, we introduce an efficient distributed approximation – GreeDi. We will further show, that under some additional assumptions, much stronger guarantees can be obtained.

5.2.2 Our GreeDi Approximation

Our efficient distributed method GreeDi is shown in Algorithm 2. It parallels the intractable Algorithm 1, but replaces the selection of optimal subsets, i.e., $A_i^c[k]$, by greedy solutions $A_i^{\mathrm{gc}}[k]$. Due to the approximate nature of the greedy algorithm, we allow it to pick sets slightly larger than k. More precisely, GreeDi is a two-round algorithm that takes the ground set V, the number of partitions m, and the cardinality constraint κ. It first distributes the ground set over m machines. Then each machine separately runs the standard greedy algorithm by sequentially finding an element $e \in V_i$ that maximizes the discrete derivative (5.1.2). Each machine i–in parallel–continues adding elements to the set $A_i^{\mathrm{gc}}[\cdot]$ until it reaches κ elements. We define $A_{\max}^{\mathrm{gc}}[\kappa]$ to be the set with the maximum value among $\{A_1^{\mathrm{gc}}[\kappa], A_2^{\mathrm{gc}}[\kappa], \dots, A_m^{\mathrm{gc}}[\kappa]\}$. Then the solutions are merged, i.e., $B = \cup_{i=1}^m A_i^{\mathrm{gc}}[\kappa]$, and another round of greedy selection is performed over B until κ elements are selected. We denote this solution by $A_B^{\mathrm{gc}}[\kappa]$. The final distributed solution with parameters m and κ, denoted by $A^{\mathrm{gd}}[m,\kappa]$, is the set with a

Algorithm 2: Greedy Distributed Submodular Maximization (GreeDi)

```
Input: Set $V$, #of partitions $m$, constraints $\kappa$.
Output: Set $A^{\mathrm{gd}}[m,\kappa]$.
1: Partition $V$ into $m$ sets $V_1, V_2, \ldots, V_m$ (arbitrarily or at random).
2: Run the standard greedy algorithm on each set $V_i$ to find a solution $A_i^{\mathrm{gc}}[\kappa]$.
3: Find $A_{\max}^{\mathrm{gc}}[\kappa] = \arg\max_A\{F(A) : A \in \{A_1^{\mathrm{gc}}[\kappa], \ldots, A_m^{\mathrm{gc}}[\kappa]\}\}$
4: Merge the resulting sets: $B = \cup_{i=1}^m A_i^{\mathrm{gc}}[\kappa]$.
5: Run the standard greedy algorithm on $B$ to find a solution $A_B^{\mathrm{gc}}[\kappa]$.
6: Return $A^{\mathrm{gd}}[m,\kappa] = \arg\max_A\{F(A) : A \in \{A_{\max}^{\mathrm{gc}}[\kappa], A_B^{\mathrm{gc}}[\kappa]\}\}$.
```

higher value between $A_{\max}^{\mathrm{gc}}[\kappa]$ and $A_B^{\mathrm{gc}}[\kappa]$ (*c.f.*, Figure 5.2 shows GreeDi schematically). The following result parallels Theorem 4.

Theorem 5. *Let f be a monotone submodular function and $\kappa \geq k$. Then*

$$f(A^{gd}[m,\kappa]) \geq \frac{(1-e^{-\kappa/k})}{\min(m,k)} f(A^c[k]).$$

For the special case of $\kappa = k$ the result of Theorem 5 simplifies to $f(A^{\mathrm{gd}}[m,\kappa]) \geq \frac{(1-1/e)}{\min(m,k)} f(A^c[k])$. Moreover, it is straightforward to generalize GreeDi to multiple rounds (i.e., more than two) for very large datasets.

In light of Theorem 4, one can expect that in general it is impossible to eliminate the dependency of the distributed solution on $\min(k,m)$ [1]. However, as we show in the rest of this section, in many practical settings, the ground set V exhibits rich geometrical structure that can be used to obtain stronger guarantees.

5.2.3 Performance on Datasets with Geometric Structure

In practice, we can hope to do much better than the worst case bounds shown previously by exploiting underlying structure often presents in real data and important set functions. In this part, we assume that a metric $d : V \times V \to \mathbb{R}$ exists on the data elements, and analyze performance of the algorithm on functions that vary slowly with changes in the input. We refer to these as *Lipschitz functions*:

Definition 6. Let $\lambda > 0$. A set function $f : 2^V \to \mathbb{R}$ is λ-*Lipschitz* w.r.t. metric d on V, if for any integer k, any equal sized sets $S = \{e_1, e_2, \ldots, e_k\} \subseteq V$ and $S' =$

[1]We'll show in Chapter 6 that the tightest dependency is $\Theta(\sqrt{\min(m,k)})$.

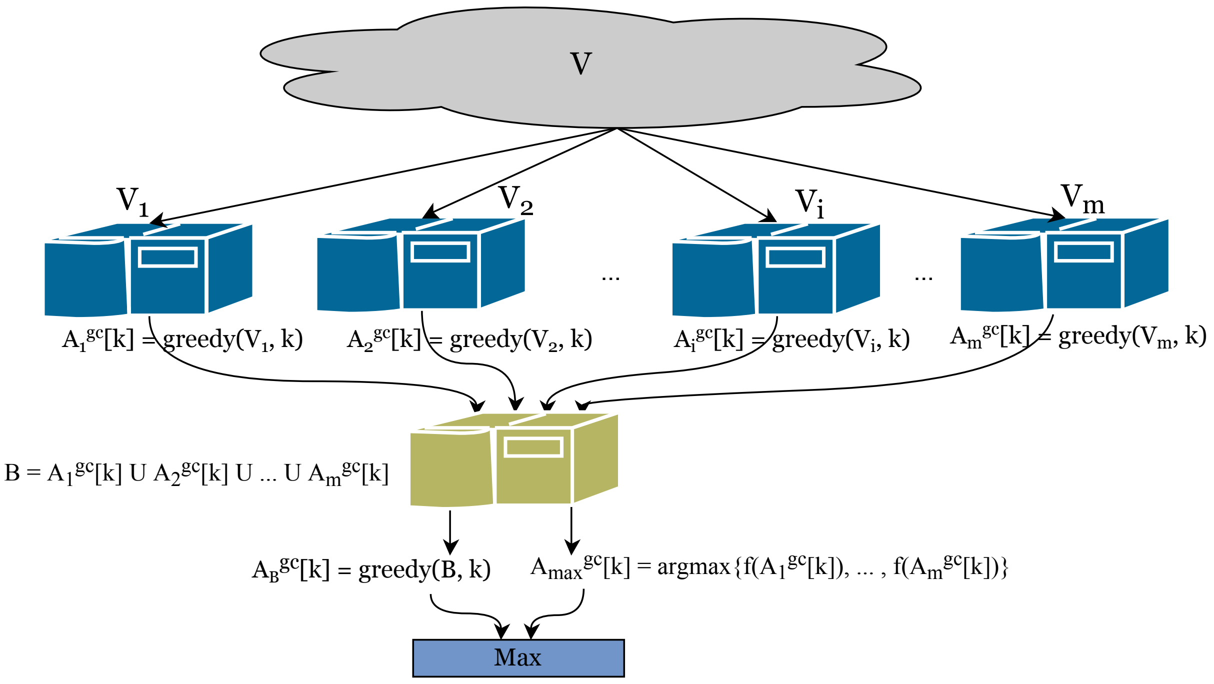


Figure 5.2: *Illustration of our two-round algorithm* GreeDi

$\{e'_1, e'_2, \ldots, e'_k\} \subseteq V$ and any matching of elements: $M = \{(e_1, e'_1), (e_2, e'_2) \ldots, (e_k, e'_k)\}$, the difference between $f(S)$ and $f(S')$ is bounded by:

$$\left|f(S) - f(S')\right| \leq \lambda \sum_i d(e_i, e'_i). \tag{5.2.1}$$

We can show that the objective functions from both applications considered in this chapter, namely, exemplar-based clustering (*c.f.* Section 3.2), and active set selection (*c.f.* Section 3.1.1) are λ-Lipschitz for suitable kernels/distance functions:

Proposition 7. *Suppose that the covariance matrix of a Gaussian process is parametrized via a positive definite kernel $\mathcal{K} : V \times V \to \mathbb{R}$ which is Lipschitz continuous with respect to metric $d : V \times V \to \mathbb{R}$ with constant $\mathcal{L}$, i.e., for any triple of points $x_1, x_2, x_3 \in V$, we have $|\mathcal{K}(x_1, x_3) - \mathcal{K}(x_2, x_3)| \leq \mathcal{L}d(x_1, x_2)$. Then, the mutual information $I(\mathbf{Y}_S; \mathbf{X}_V) = \frac{1}{2}\log\det(\mathbf{I} + K_S)$ for the Gaussian process is λ-Lipschitz with $\lambda = \mathcal{L}k^3$, where k is the number of elements in the selected subset S.*

Proposition 8. *Let $d : V \times V \to \mathbb{R}$ be a metric on the elements of the dataset. Furthermore, let $l : V \times V \to \mathbb{R}$ encode the dissimilarity between elements of the underlying set V. Then* *for $l = d^{\alpha}$, $\alpha \geq 1$ the loss function $L(S) = \frac{1}{|V|}\sum_{v\in V}\min_{e\in S} l(e,v)$ (and hence also the corresponding submodular utility function f) is λ-Lipschitz with $\lambda = \alpha R^{\alpha-1}$, where R is the diameter of the ball encompassing elements of the dataset in the metric space. In particular, for the k-medoid problem, which minimizes the loss function over all clusters with respect to $l = d$, we have $\lambda = 1$, and for the k-means problem, which minimizes the loss function over all clusters with respect to $l = d^2$, we have $\lambda = 2R$.*

Beyond Lipschitz-continuity, many practical instances of submodular maximization can be expected to satisfy a natural *density* condition. Concretely, whenever we consider a representative set (i.e., optimal solution to the submodular maximization problem), we expect that any of its constituent elements has potential candidates for replacement in the ground set. For example, in our exemplar-based clustering application, we expect that cluster centers are not isolated points, but have many almost equally representative points close by. Formally, for any element $v \in V$, we define its *α-neighborhood* as the set of elements in V within distance α from v (i.e., α-close to v):

$$N_{\alpha}(v) = \{w : d(v,w) \leq \alpha\}.$$

By λ-Lipschitz-continuity, it must hold that if we replace element v in set S by an α-close element v' (i.e., $v' \in N_{\alpha}(v)$) to get a new set S' of equal size, it must hold that $|f(S) - f(S')| \leq \alpha\lambda$.

As described earlier, our algorithm GreeDi partitions V into sets $V_1, V_2, \dots V_m$ for parallel processing. If in addition we assume that elements are assigned uniformly at random to different machines, α-neighborhoods are sufficiently dense, and the submodular function is Lipschitz continuous, then GreeDi is guaranteed to produce a solution close to the centralized one. More formally, we have the following theorem.

Theorem 9. *Under the conditions that 1) elements are assigned uniformly at random to m machines, 2) for each $e_i \in A^c[k]$ we have $|N_{\alpha}(e_i)| \geq km\log(k/\delta^{1/m})$, and 3) f is λ-Lipschitz continuous, then with probability at least $(1-\delta)$ the following holds:*

$$f(A^{gd}[m,\kappa]) \geq (1 - e^{-\kappa/k})(f(A^c[k]) - \lambda\alpha k).$$

Note that once the above conditions are satisfied for small values of α (meaning that there is a high density of data points within a small distance from each element of the optimal solution) then the distributed solution will be close to the optimal centralized one. In particular if we let $\alpha \to 0$, the distributed solution is guaranteed to be within a $1 - e^{\kappa/k}$ factor from the optimal centralized solution. This situation naturally corresponds to very large datasets. In the following, we discuss more thoroughly this important scenario.

5.2.4 Performance Guarantees for Very Large Datasets

Suppose that our dataset is a finite sample V drawn i.i.d. from an underlying *infinite* set $\mathcal{V}$, according to some (unknown) probability distribution. Let $A^c[k]$ be an optimal solution in the infinite set, i.e., $A^c[k] = \arg\max_{S \subseteq \mathcal{V}} f(S)$, such that around each $e_i \in A^c[k]$, there is a neighborhood of radius at least α^* where the probability density is at least β at all points (for some constants α^* and β). This implies that the solution consists of elements coming from reasonably dense and therefore representative regions of the dataset.

Let us suppose $g : \mathbb{R} \to \mathbb{R}$ is the *growth function of the metric*: $g(\alpha)$ is defined to be the volume of a ball of radius α centered at a point in the metric space. This means, for $e_i \in A^c[k]$ the probability of a random element being in $N_\alpha(e_i)$ is at least $\beta g(\alpha)$ and the expected number of α neighbors of e_i is at least $E[|N_\alpha(e_i)|] = n\beta g(\alpha)$. As a concrete example, Euclidean metrics of dimension D have $g(\alpha) = O(\alpha^D)$. Note that for simplicity we are assuming the metric to be homogeneous, so that the growth function is the same at every point. For heterogeneous spaces, we require g to have a uniform lower bound on the growth function at every point.

In these circumstances, the following theorem guarantees that if the dataset V is sufficiently large and f is λ-Lipschitz, then GreeDi produces a solution close to the centralized one.

Theorem 10. *For $n \geq \dfrac{8km\log(k/\delta^{1/m})}{\beta g(\frac{\varepsilon}{\lambda k})}$, where $\frac{\varepsilon}{\lambda k} \leq \alpha^*$, if the algorithm* GreeDi *assigns elements uniformly at random to m processors , then with probability at least* $(1 - \delta)$,

$$f(A^{gd}[m,\kappa]) \geq (1 - e^{-\kappa/k})(f(A^c[k]) - \varepsilon).$$

The above theorem shows that for very large datasets, GreeDi provides a solution that is within a $1 - e^{\kappa/k}$ factor of the optimal centralized solution. This result is based on the fact that for sufficiently large datasets, there is a suitably dense neighborhood around each member of the optimal solution. Thus, if the elements of the dataset are partitioned uniformly at random to m processors, at least one partition contains a set $A_i^c[k]$ such that its elements are very close to the elements of the optimal centralized solution and provides a constant factor approximation of the optimal centralized solution.

5.2.5 Handling Decomposable Functions

So far, we have assumed that the objective function f is given to us as a black box, which we can evaluate for any given set S *independently* of the dataset V. In many settings, however, the objective f depends itself on the entire dataset. In such a setting, we cannot use GreeDi as presented above, since we cannot evaluate f on the individual machines without access to the full set V. Fortunately, many such functions have a simple structure which we call *decomposable*. More precisely, we call a submodular function f *decomposable* if it can be written as a sum of submodular functions as follows [KG10]:

$$f(S) = \frac{1}{|V|} \sum_{i \in V} f_i(S)$$

In other words, there is separate submodular function associated with every data point $i \in V$. We require that each f_i can be evaluated without access to the full set V. Note that the exemplar based clustering application we discussed in Section 3.2 is an instance of this framework, among many others. Let us define the evaluation of f restricted to $D \subseteq V$ as follows:

$$f_D(S) = \frac{1}{|D|} \sum_{i \in D} f_i(S)$$

In the remaining of this section, we show that assigning each element of the dataset randomly to a machine and running GreeDi will provide a solution that is with high probability close to the optimum solution. For this, let us assume that f_i's are bounded, and without loss of generality $0 \leq f_i(S) \leq 1$ for $1 \leq i \leq |V|, S \subseteq V$. Similar to Section 5.2.3 we assume that GreeDi performs the partition by assigning elements uniformly at random to the machines. These machines then each greedily optimize f_{V_i}. The second stage of GreeDi optimizes f_U, where $U \subseteq V$ is chosen uniformly at random with size $\lceil n/m \rceil$.

Then, we can show the following result. First, for any fixed ϵ, m, k, let us define n_0 to be the smallest integer such that for $n \geq n_0$ we have $\ln(n)/n \leq \epsilon^2/(mk)$.

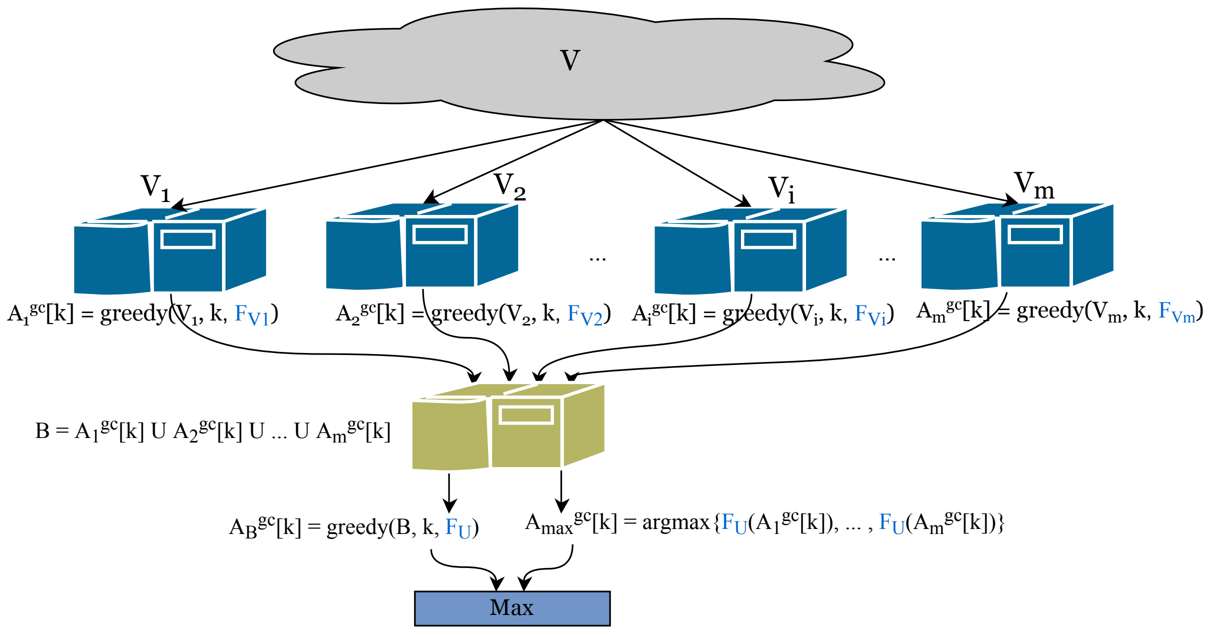

V
V1
V2
Vi
Vm
...
...
A1gc[k] = greedy(V1, k, FV1)
A2gc[k] = greedy(V2, k, FV2)
Aigc[k] = greedy(Vi, k, FVi)
Amgc[k] = greedy(Vm, k, FVm)
B = A1gc[k] U A2gc[k] U ... U Amgc[k]
ABgc[k] = greedy(B, k, FU)
Amaxgc[k] = argmax{FU(A1gc[k]), ... , FU(Amgc[k])}
Max

Figure 5.3: *Illustration of our two-round algorithm* GreeDi *for decomposable functions*

Theorem 11. *For $n \geq \max(n_0, m\log(\delta/4m)/\epsilon^2)$, $\epsilon < 1/4$, and under the assumptions of Theorem 10, we have, with probability at least $1-\delta$,*

$$f(A^{gd}[m,\kappa]) \geq (1-e^{-\kappa/k})(f(A^c[k]) - 2\varepsilon).$$

The above result demonstrates why GreeDi performs well on decomposable submodular functions with massive data even when they are evaluated locally on each machine. We will report our experimental results on exemplar-based clustering in the next section.

5.2.6 Performance of GreeDi on Random Partitions Without Geometric Structure

Very recently, a constant $(1-e^{-1})/2$-approximation guarantee was proven for GreeDi for the case of random partitioning of the data among the m machines.

Theorem 12 (Barbosa et al. [Bar+15] and Mirrokni and Zadimoghaddam [MZ15]). *If elements are assigned uniformly at random to the machines, and $\kappa = k$,* GreeDi *gives a*

$(1 - 1/e)/2$ *approximation guarantee (in the average case) to the optimum centralized solution.*

$$\mathbb{E}[f(A^{gd}[m,k])] \geq \frac{1-1/e}{2} f(A^c[k]).$$

These results show that random partitioning of the data is sufficient to guarantee that GreeDi provides a constant factor approximation, irrespective of m and k, and without the requirement of any geometric structure. On the other hand, if geometric structure is present, the bounds from the previous sections can provide sharper approximation guarantees.

5.3 (Non-Monotone) Submodular Functions with General Constraints

In this section we show how GreeDi can be extended to handle 1) more general constraints, and 2) non-monotone submodular functions. More precisely, we consider the following optimization setting

$$\begin{aligned}&\text{Maximize } f(S)\\&\text{Subject to } S \in \zeta.\end{aligned}$$

Here, we assume that the feasible solutions should be members of the constraint set $\zeta \subseteq 2^V$. The function $f(\cdot)$ is submodular but may not be monotone. By overloading the notation we denote the set that achieves the above constrained optimization problem by $A^c[\zeta]$. Throughout this section we assume that the constraint set ζ is hereditary, meaning that if $A \in \zeta$ then for any $B \subseteq A$ we also require that $B \in \zeta$. Cardinality constraints are obviously hereditary, so are all the examples we mention below.

5.3.1 GreeDi Approximation Guarantee under More General Constraints

Assume that we have a set of constraints $\zeta \subseteq 2^V$ that is hereditary. Further assume we have access to a "black box" algorithm X that gives us a constant factor approximation guarantee for maximizing a non-negative (but not necessarily monotone) submodular function f subject to ζ, i.e.

$$X : (f, \zeta) \mapsto A^X[\zeta] \in \zeta \;\text{ s.t. }\; f(A^X[\zeta]) \geq \tau \max_{A \in \zeta} f(A). \tag{5.3.1}$$

We can modify GreeDi to use any such approximation algorithm as a black box, and provide theoretical guarantees about the solution. In order to process a large dataset, it first distributes the ground set over m machines. Then instead of greedily selecting elements, each machine i–in parallel–separately runs the black box algorithm X on its local data in order to produce a feasible set $A_i^X[\zeta]$ meeting the constraints ζ. We denote by $A^{\text{gc}}_{\max}[\zeta]$ the set with maximum value among $A_i^X[\zeta]$. Next, the solutions are merged: $B = \cup_{i=1}^m A_i^X[\zeta]$, and the black box algorithm is applied one more time to set B to produce a solution $A_B^{\text{gc}}[\zeta]$. Then, the distributed solution for parameter m and constraints ζ, $A^{Xd}[m, \zeta]$, is the best among $A^{\text{gc}}_{\max}[\zeta]$ and $A_B^{\text{gc}}[\zeta]$. This procedure is given in more detail in Algorithm 3.

Algorithm 3: GreeDi under General Constraints

```
Input: Set $V$, #of partitions $m$, constraints $\zeta$, submodular function $f$.
Output: Set $A^{Xd}[m, \zeta]$.
1: Partition $V$ into $m$ sets $V_1, V_2, \ldots, V_m$.
2: In parallel:
   Run the approximation algorithm X on each set $V_i$ to find a solution $A_i^X[\zeta]$.
3: Find $A^{\text{gc}}_{\max}[\zeta] = \arg\max_A \{F(A) | A \in \{A_1^X[\zeta], \ldots, A_m^X[\zeta]\}\}$.
4: Merge the resulting sets: $B = \cup_{i=1}^m A_i^X[\zeta]$.
5: Run the approximation algorithm $X$ on $B$ to find a solution $A_B^{\text{gc}}[\zeta]$.
6: Return $A^{Xd}[m, \zeta] = \arg\max\{A^{\text{gc}}_{\max}[\zeta], A_B^{\text{gc}}[\zeta]\}$.
```

The following result generalizes Theorem 5 for maximizing a submodular function subject to more general constraints.

Theorem 13. *Let f be a non-negative submodular function and X be a black box algorithm that provides a τ-approximation guarantee for submodular maximization subject to a set of hereditary constraints ζ. Then*

$$f(A^{Xd}[m, \zeta])) \geq \frac{\tau}{\min\left(m, \rho([\zeta])\right)} f(A^c[\zeta]),$$

where $f(A^c[\zeta])$ is the optimum centralized solution, and $\rho([\zeta]) = \max_{A \in \zeta} |A|$.

Specifically, for submodular maximization subject to the matroid constraint $\mathcal{M}$, we have $\rho([A \in \mathcal{I}]) = r_{\mathcal{M}}$ where $r_{\mathcal{M}}$ is the rank of the matroid (i.e., the maximum size of any independent set in the system). For submodular maximization subject to the knapsack constraint $\mathcal{R}$, we can bound $\rho([c(A) \leq \mathcal{R}])$ by $\lceil \mathcal{R}/ \min_v c(v) \rceil$ (i.e. the capacity of the knapsack divided by the smallest weight of any element).

Performance on Datasets with Geometric Structure. When the submodular function $f(\cdot)$ and the constraint set ζ have more structure, then we can provide much better approximation guarantees. Assuming the elements of V are embedded in metric space with distance $d : V \times V \to \mathbb{R}^+$, we say that ζ is *locally replaceable* with respect to a set $S \subseteq V$ with parameter $\alpha > 0$ if

$$\forall S' \subseteq V \text{ s.t. } |S'| = |S| \text{ and } d_\infty(S, S') \leq \alpha \Rightarrow S' \in \zeta.$$

Here, we define the distance d_∞ between two sets S and S' of the same size k as follows. Let M be the set of all possible matchings between S and S', i.e.,

$$M = \{((e_1, e'_1), \ldots, (e_k, e'_k)) \text{ s.t } e_i \in S \text{ and } e'_i \in S' \text{ for } 1 \leq i \leq k\}.$$

Then $d_\infty(S, S') = \min_M \max_i d(e_i, e'_i)$. We require locality only with respect to $A^c[\zeta]$ to ensure that the optimum solution can be well approximated. What the locally replaceable property requires is that as elements of $A^c[\zeta]$ get replaced by nearby elements, the resulting set is also a feasible solution. Combining this property with λ-Lipschitzness will provide us with the following theorem.

Theorem 14. *Under the conditions that 1) elements are assigned uniformly at random to m machines, 2) for each $e_i \in A^c[\zeta]$ we have $|N_\alpha(e_i)| \geq \rho([\zeta]) m \log(\rho([\zeta])/\delta^{1/m})$, 3) $f(\cdot)$ is λ-Lipschitz, and 4) ζ is locally replaceable with respect to $A^c[\zeta]$ with parameter α, then with probability at least $(1-\delta)$,*

$$f(A^{Xd}[m, \zeta])) \geq \tau(f(A^c[\zeta]) - \lambda\alpha\rho([\zeta])).$$

The above result generalizes Theorem 9 for maximizing non-negative submodular functions subject to different constraints.

Performance Guarantee for Very Large datasets. Similarly, we can generalize Theorem 10 for maximizing non-negative submodular functions subject to more general constraints. Suppose that our dataset is a finite sample V drawn i.i.d. from an underlying *infinite* set $\mathcal{V}$, according to some (unknown) probability distribution. Let $A^c[\zeta]$ be an optimal solution in the infinite set, i.e., $A^c[\zeta] = \arg\max_{S\subseteq\mathcal{V}} f(S)$, such that around each $e_i \in A^c[\zeta]$, there is a neighborhood of radius at least α^* where the probability density is at least β at all points (for some constants α^* and β). Recall that $g : \mathbb{R} \to \mathbb{R}$ is the growth function where $g(\alpha)$ measures the volume of a ball of radius α centered at a point in the metric space.

Theorem 15. *For* $n \geq \frac{8\rho([\zeta])m\log(\rho([\zeta])/\delta^{1/m})}{\beta g(\frac{\varepsilon}{\lambda\rho([\zeta])})}$, *where* $\frac{\varepsilon}{\lambda\rho([\zeta])} \leq \alpha^*$, *if* GreeDi *assigns elements uniformly at random to m processors and under the conditions that f is* λ*-Lipschitz, and* ζ *is locally replaceable with respect to* $A^c[\zeta]$ *with parameter* α^*, *then with probability at least* $(1-\delta)$, *we have*

$$f(A^{Xd}[m,\zeta])) \geq \tau(f(A^c[\zeta]) - \varepsilon).$$

Performance Guarantee for Decomposable Functions. For the case of decomposable functions described in Section 5.2.5, the following generalization of Theorem 11 holds for maximizing a non-negative submodular function subject to more general constraints. Let us define n_0 to be the smallest integer such that for $n \geq n_0$ we have $\ln(n)/n \leq \epsilon^2/(m\cdot\rho([\zeta]))$.

Theorem 16. *For* $n \geq \max(n_0, m\log(\delta/4m)/\epsilon^2)$, $\epsilon < 1/4$, *and under the assumptions of Theorem 15, we have, with probability at least* $1-\delta$,

$$f(A^{Xd}[m,\zeta])) \geq \tau(f(A^c[\zeta]) - 2\varepsilon).$$

5.4 Experiments

In our experimental evaluation we wish to address the following questions: 1) how well does GreeDi perform compared to the centralized solution, 2) how good is the performance of GreeDi when using decomposable objective functions (see Section 5.2.5), and finally 3) how well does GreeDi scale in the context of massive datasets. To this end, we run GreeDi on three scenarios: exemplar based clustering, active set selection in GPs and finding the maximum cuts in graphs.

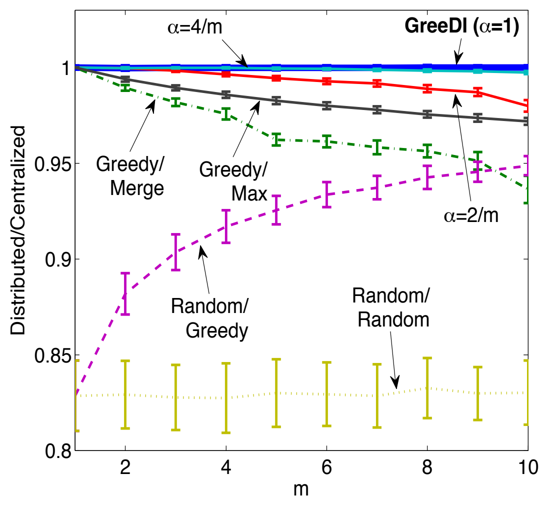
We compare the performance of our GreeDi method to the following naive approaches:

- *random/random*: in the first round each machine simply outputs k randomly chosen elements from its local data points and in the second round k out of the merged mk elements, are again randomly chosen as the final output.
- *random/greedy*: each machine outputs k randomly chosen elements from its local data points, then the standard greedy algorithm is run over mk elements to find a solution of size k.
- *greedy/merge*: in the first round k/m elements are chosen greedily from each machine and in the second round they are merged to output a solution of size k.
- *greedy/max*: in the first round each machine greedily finds a solution of size k and in the second round the solution with the maximum value is reported.

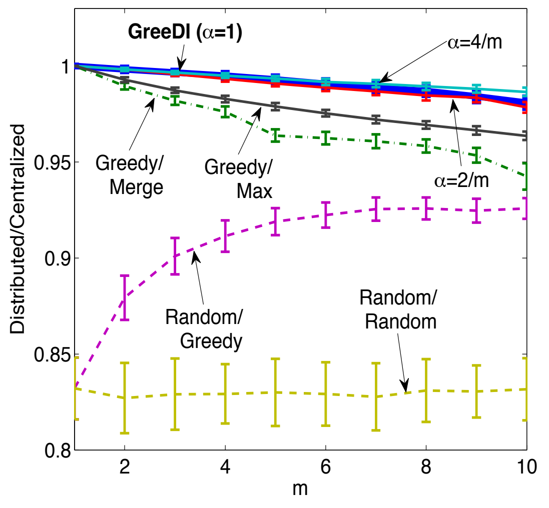
For GreeDi, we let each of the m machines select a set of size αk, and select a final solution of size k among the union of the m solutions (i.e., among αkm elements). We present the performance of GreeDi for different parameters $\alpha > 0$. For datasets where we are able to find the centralized solution, we report the ratio of $f(A_{\text{dist}}[k])/f(A^{\text{gc}}[k])$, where $A_{\text{dist}}[k]$ is the distributed solution (in particular $A^{\text{gd}}[m, \alpha k, k] = A_{\text{dist}}[k]$ for GreeDi).

5.4.1 Exemplar Based Clustering

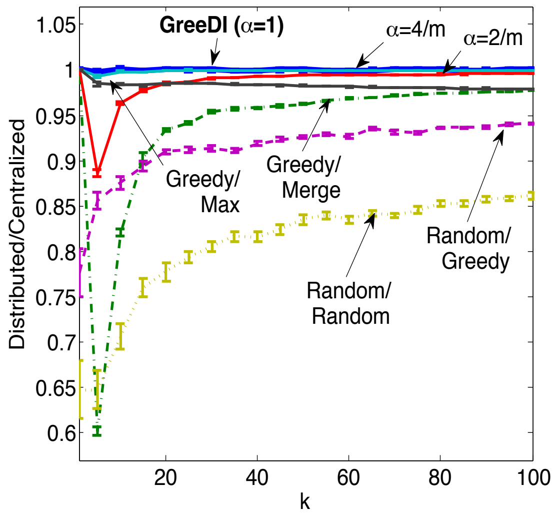
Our exemplar based clustering experiment involves GreeDi applied to the clustering utility $f(S)$ (see Sec. 3.2) with $d(x, x') = \|x - x'\|^2$. We performed our experiments on a set of 10,000 *Tiny Images* [TFF08]. Each 32 by 32 RGB pixel image was represented by a 3,072 dimensional vector. We subtracted from each vector the mean value, normalized it to unit norm, and used the origin as the auxiliary exemplar. Figure 5.4a compares the performance of our approach to the benchmarks with the number of exemplars set to $k = 50$, and varying number of partitions m. It can be seen that GreeDi significantly outperforms the benchmarks and provides a solution that is very close to the centralized one. Interestingly, even for very small $\alpha = \kappa/k < 1$, GreeDi performs very well. Since the exemplar based clustering utility function is decomposable, we repeated the experiment for the more realistic case where the function evaluation in each machine was restricted to the local elements of the dataset in that particular machine (rather than the entire dataset). Figure 5.4b shows similar qualitative behavior for decomposable objective functions.



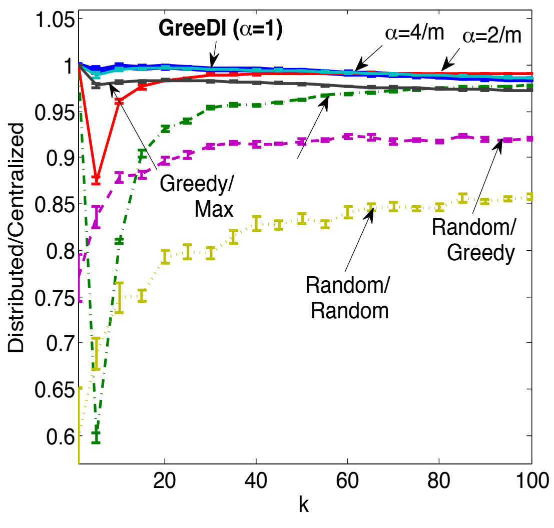
(a) Global objective function



(b) Local objective function



(c) Global objective function



(d) Local objective function

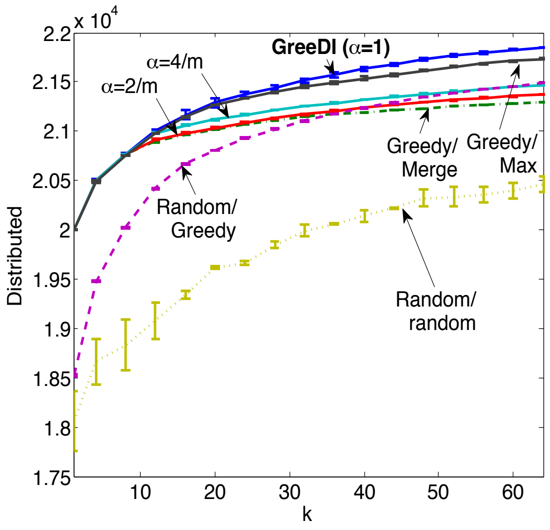
Figure 5.4: *Performance of* GreeDi *compared to the other benchmarks. a) and b) show the mean and standard deviation of the ratio of distributed vs. centralized solution for global and local objective functions with budget $k = 50$ and varying the number m of partitions. c) and d) show the same ratio for global and local objective functions for $m = 5$ partitions and varying budget k, for a set of 10,000* Tiny Images.

Large scale experiments with Hadoop. As our first large scale experiment, we applied GreeDi to the whole dataset of 80,000,000 *Tiny Images* [TFF08] in order to select a set of 64 exemplars. Our experimental infrastructure was a cluster of 10 quad-core machines running Hadoop with the number of reducers set to $m = 8000$. Hereby, each machine carried out a set of reduce tasks in sequence. We first partitioned the images uniformly at random to reducers. Each reducer separately performed the lazy greedy algorithm on its own set of 10,000 images ($\approx$123MB) to extract 64 images with the highest marginal gains w.r.t. the local elements of the dataset in that particular partition. We then merged the results and performed another round of lazy greedy selection on the merged results to extract the final 64 exemplars. Function evaluation in the second stage was performed w.r.t a randomly selected subset of 10,000 images from the entire dataset. The maximum running time per reduce task was 2.5 hours. As Figure 5.5a shows, GreeDi highly outperforms the other distributed benchmarks and can scale well to very large datasets. Figure 5.5b shows a set of cluster exemplars discovered by GreeDi where Figure 5.5c and Figure 5.5d show 100 nearest images to exemplars 26 and 63 (shown with red borders) in Figure 5.5b.

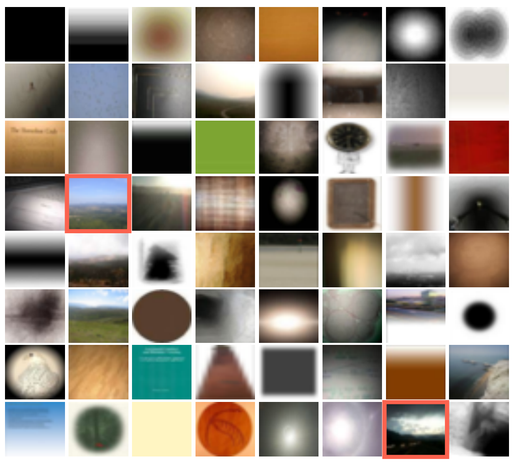
5.4.2 Active Set Selection

Our active set selection experiment involves GreeDi applied to the information gain $f(S)$ (see Sec. 3.1.1) with Gaussian kernel, $h = 0.75$ and $\sigma = 1$. We used the *Parkinsons Telemonitoring* dataset [Tsa+10] consisting of 5,875 bio-medical voice measurements with 22 attributes from people with early-stage Parkinson's disease. We normalized the vectors to zero mean and unit norm. Figure 5.6b compares the performance GreeDi to the benchmarks with fixed $k = 50$ and varying number of partitions m. Similarly, Figure 5.6a shows the results for fixed $m = 10$ and varying k. We find that GreeDi significantly outperforms the benchmarks.

Large scale experiments with Hadoop. Our second large scale experiment consists of 45,811,883 user visits from the Featured Tab of the Today Module on Yahoo! Front Page [Yah12]. For each visit, both the user and each of the candidate articles are associated with a feature vector of dimension 6. Here, we used the normalized user features. Our experimental setup was a cluster of 8 quad-core machines running Spark [Zah+10] with the number of reducers set to $m = 32$. Each reducer performed the lazy greedy algorithm on its own set of $\approx$1,431,621 vectors ($\approx$34MB) in order to extract 256 elements

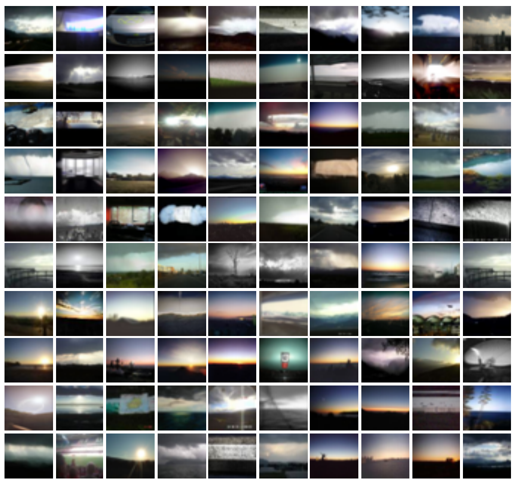


(a) Tiny Images 80M



(b)

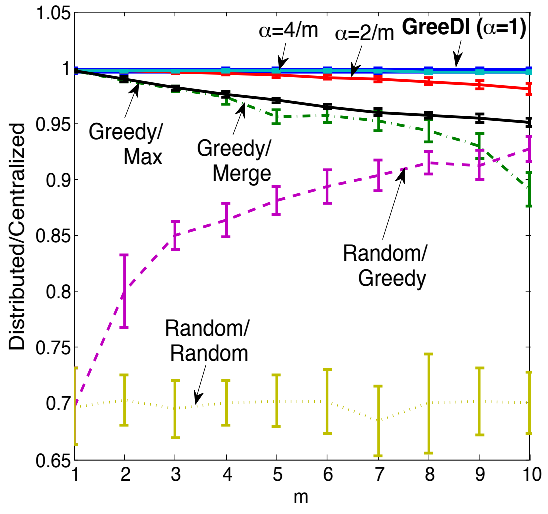
(c)



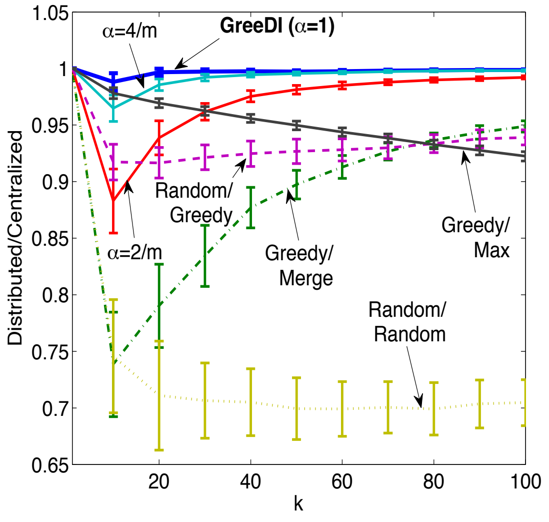
(d)

Figure 5.5: *Performance of* GreeDi *compared to the other benchmarks. a) shows the distributed solution with $m = 8000$ and varying k for local objective functions on the whole dataset of 80,000,000* Tiny Images. *b) shows a set of cluster exemplars discovered by* GreeDi, *and each column in c) shows 100 images nearest to exemplars 26 and d) shows 100 images nearest to exemplars 63 in b).*

with the highest marginal gains w.r.t the local elements of the dataset in that particular partition. We then merged the results and performed another round of lazy greedy selection on the merged results to extract the final active set of size 256. The maximum running time per reduce task was 12 minutes for selecting 128 elements and 48 minutes for selecting 256 elements. Figure 5.7 shows the performance of GreeDi compared



(a) Parkinsons Telemonitoring



(b) Parkinsons Telemonitoring

Figure 5.6: *Performance of* GreeDi *compared to the other benchmarks. a) shows the ratio of distributed vs. centralized solution with* $k = 50$ *and varying* m *for* Parkinsons Telemonitoring. *b) shows the same ratio with* $m = 10$ *and varying* k *on the same dataset.*

to the benchmarks. We note again that GreeDi significantly outperforms the other distributed benchmarks and can scale well to very large datasets.

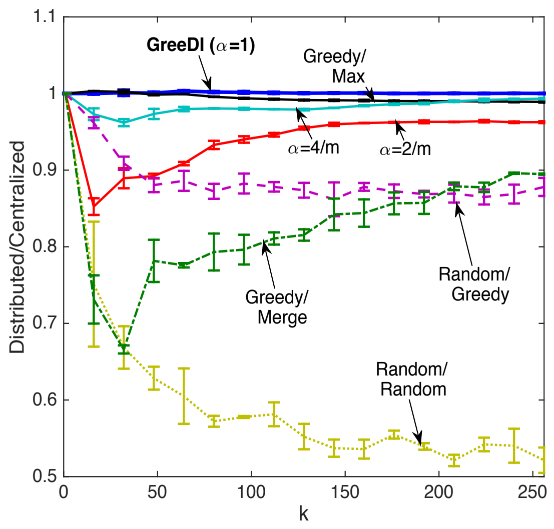
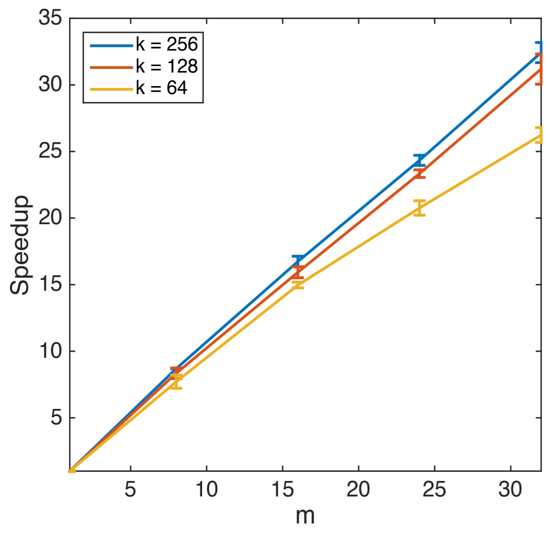


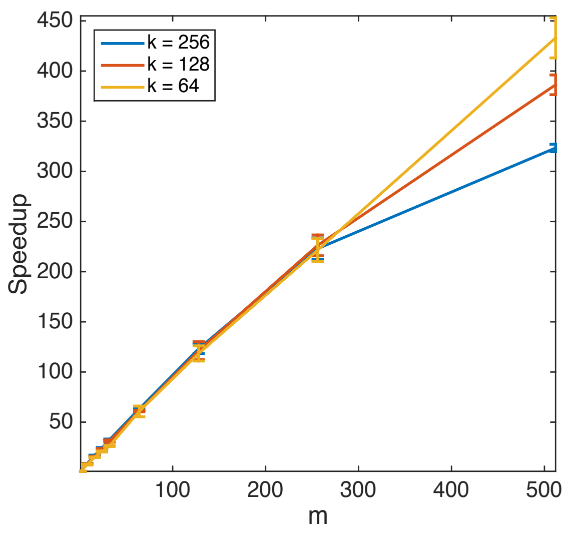
Figure 5.7: *Performance of* GreeDi *with* $m = 32$ *and varying budget* k *compared to the other benchmarks on* Yahoo! Webscope data.

Performance Comparison. Figure 5.8 shows the speedup of GreeDi compared to the centralized greedy benchmark for different values of k and varying number of partitions

m. As Figure 5.8a shows, for small values of m, the speedup is almost linear in the number of machines. However, for large values of m the running time of the second stage of GreeDi increases and ultimately dominates the whole running time. Hence, we do not observe a linear speedup anymore. This effect can be observed in Figure 5.8b. For larger values of k, the speedup is higher on fewer machines, but decreases more quickly by increasing m, as the second stage takes longer to complete.



(a) Yahoo! front page

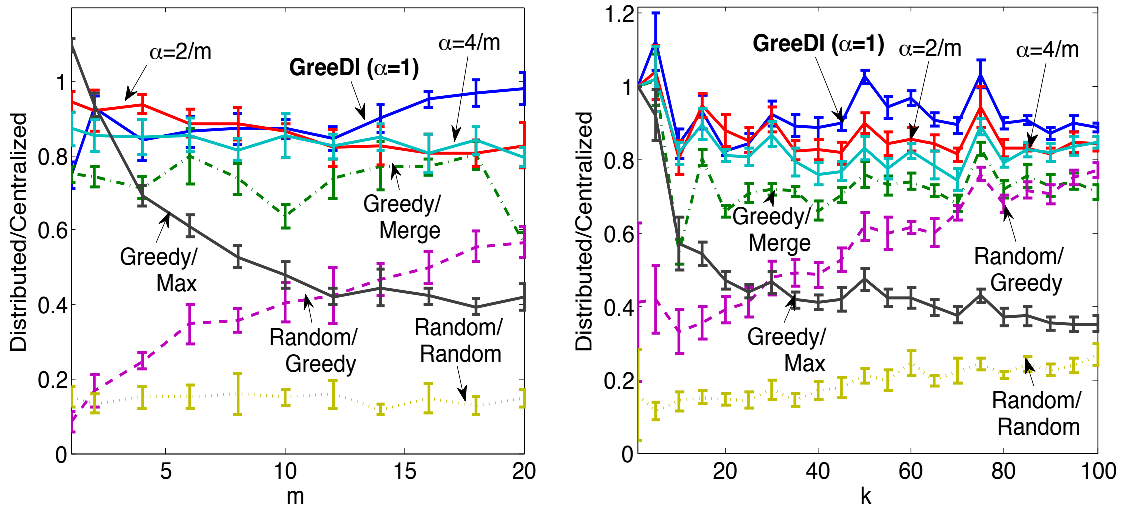


(b) Yahoo! front page

Figure 5.8: *Running time of* GreeDi *compared to the centralized greedy algorithm. a) shows the ratio of centralized vs. distributed solution with* $k = 64, 128, 256$ *and up to* $m = 32$ *machines for* Yahoo Webscope *data. b) shows the same ratio with* $k = 64, 128, 256$ *and up to* $m = 512$ *machines on the same dataset. Both experiments are performed on a cluster of 8 quad core machines.*

5.4.3 Non-Monotone Submodular Function (Finding Maximum Cuts)

We also applied GreeDi to the problem of finding maximum cuts in graphs. In our setting we used a *Facebook-like social network* [OP09]. This dataset includes the users that have sent or received at least one message in an online student community at University of California, Irvine and consists of 1,899 users and 20,296 directed ties. Figures 5.9a and 5.9b show the performance of GreeDi applied to the cut function on graphs. We evaluated the objective function locally on each partition. Thus, the links between the partitions are disconnected. Since the problem of finding the maximum cut in a graph



(a) Facebook-like social network (b) Facebook-like social network

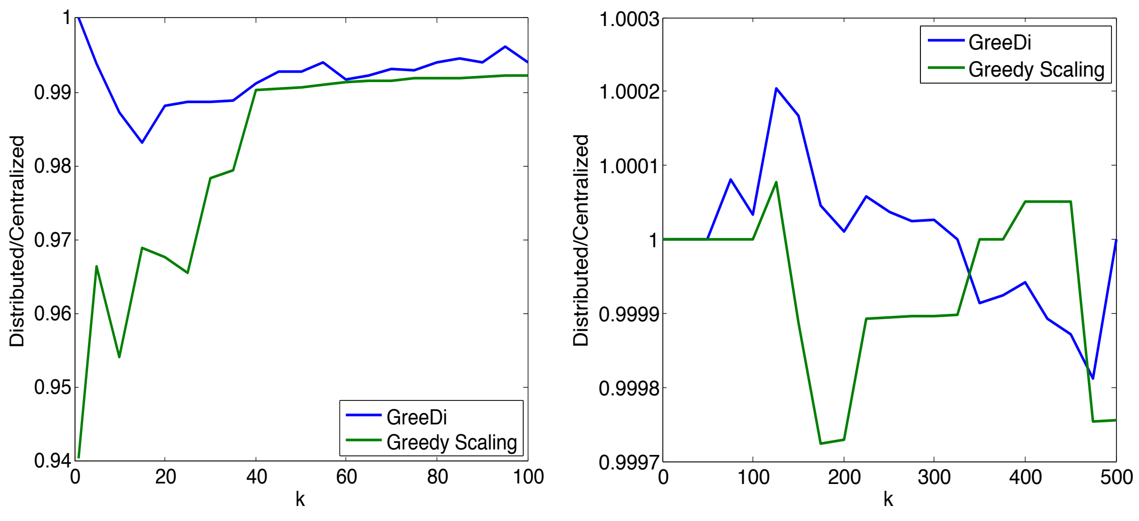
Figure 5.9: *Performance of* GreeDi *compared to the other benchmarks. a) shows the mean and standard deviation of the ratio of distributed to centralized solution for budget $k = 20$ with varying number of machines m and b) shows the same ratio for varying budget k with $m = 10$ on* Facebook-like social network.

is non-monotone submodular, we applied the RandomGreedy algorithm proposed by [Buc+14] to find the near optimal solution in each partition.

Although the cut function does not decompose additively over individual data points, perhaps surprisingly, GreeDi still performs very well, and significantly outperforms the benchmarks. This suggests that our approach is quite robust, and may be more generally applicable.

5.4.4 Comparision with Greedy Scaling.

[Kum+13] recently proposed an alternative approach–GreedyScaling–for parallel maximization of submodular functions. GreedyScaling is a randomized algorithm that carries out a number (typically less than k) rounds of MapReduce computations. We applied GreeDi to the submodular coverage problem in which given a collection V of sets, we would like to pick at most k sets from V in order to maximize the size of their union. We compared the performance of our GreeDi algorithm to the reported performance of GreedyScaling on the same datasets, namely *Accidents* [Geu+03] and *Kosarak* [Bod12]. As Figures 5.10a and 5.10b show, GreeDi outperforms



(a) Accidents (b) Kosarak

Figure 5.10: *Performance of* GreeDi *compared to the GreedyScaling algorithm of [Kum+13] (as reported in their paper). a) shows the ratio of distributed to centralized solution on Accidents dataset with 340,183 elements and b) shows the same ratio for Kosarak dataset with 990,002 elements. The results are reported for varying budget k and varying number of machines $m = n/\mu$ where $\mu = O(kn^{\delta} \log n)$ and n is the size of the dataset. The results are reported for $\delta = 1/2$. Note that the results presented by [Kum+13] indicate that GreedyScaling generally requires a substantially larger number of MapReduce rounds compared to* GreeDi.

GreedyScaling on the *Accidents* dataset and its performance is comparable to that of GreedyScaling in the *Kosarak* dataset.

5.5 Summary

We have developed an efficient distributed protocol GreeDi, for constrained submodular function maximization. We have theoretically analyzed the performance of our method and showed that under certain natural conditions it performs very close to the centralized (albeit impractical in massive datasets) solution. We have also demonstrated the effectiveness of our approach through extensive experiments, including active set selection in GPs on a dataset of 45 million examples, and exemplar based summarization of a collection of 80 million images using Hadoop. We believe our results provide an important step towards solving submodular optimization problems in very large scale, real applications.

6 Distributed Submodular Cover: Succinctly Summarizing Massive Data

In Chpater 5, we studied distributed methods for selecting a small subset of data points such that they maximize a particular quality criterion. However, in many applications we don't know the size of the smallest subset that well represents a massive dataset. For example, in the variable selection problem [DK11], one is interested in selecting enough features from a set of observations such that the resulting linear predictor achieves a certain accuracy; in the sensor placement problem [Kra+08a], the main objective is to install a small number of sensors such that they (approximately) cover the whole monitoring area; similarly, in viral marketing [KKT03], the goal is to identify a small set of individuals in a social network, such that by sending advertisement to those the message spreads to a substantial portion of the network.

Our focus in this chapter is to find a succinct summary of the data, i.e., a subset, ideally as small as possible, which achieves a desired (large) fraction of the utility provided by the full dataset. Hereby, utility is measured according to an appropriate submodular function. We formalize this problem as a submodular cover problem, and seek efficient algorithms for solving it in face of massive data. As discussed in Chapter

2, the celebrated result of Wolsey [Wol82] shows that a greedy approach that selects elements sequentially in order to maximize the gain over the items selected so far, yields a logarithmic factor approximation. It is also known that improving upon this approximation ratio is hard under natural complexity theoretic assumptions [Fei98]. Even though such a greedy algorithm produces near-optimal solutions, it is impractical for massive datasets, as sequential procedures that require centralized access to the full data are highly constrained in terms of speed and memory.

In this chapter, we develop the first distributed algorithm – DisCover – for solving the submodular cover problem. It can be easily implemented in MapReduce-style parallel computation models [DG08] and provides a solution that is competitive with the (impractical) centralized solution. We also study a natural trade-off between the communication cost (for each round of MapReduce) and the number of rounds. The trade-off lets us choose between a small communication cost between machines while having more rounds to perform or a large communication cost with the benefit of running fewer rounds. Our experimental results demonstrate the effectiveness of our approach on a variety of submodular cover instances: dominating set, exemplar-based clustering, and active set selection in non-parametric learning. We also implemented DisCover on Spark [Zah+10] and approximately solved dominating set on a social graph containing more than 65 million nodes and 1.8 billion edges.

6.1 The Distributed Submodular Cover Problem

The focus of this chapter is on the *submodular cover problem*, i.e., finding the smallest set A^c such that it achieves a certain quality. More precisely, we have a large dataset indexed by V (called the ground set) and we wish to select a small subset $A \subseteq V$ such that A achieves a utility $Q = (1-\epsilon)f(V)$ for some $0 \leq \epsilon \leq 1$. I.e., we are interested in the following optimization problem:

$$A^c = \arg\min_{A \subseteq V} |A|, \quad \text{such that} \quad f(A) \geq Q. \tag{6.1.1}$$

We call A^c the optimum *centralized* solution with size $k = |A^c|$. Throughout this chapter we assume that the utility function $f : 2^V \to \mathbb{R}_+$ that *measures* the representativeness of any subset $A \subseteq V$, according to some objective, is monotone submodular. In Section 3 we discussed concrete instances of functions f that we consider in our experiments.

Unfortunately, finding A^c is NP-hard, for many classes of submodular functions [Fei98]. However, as discussed in Section 2.2.1, the greedy algorithm that iteratively selects elements with maximum marginal utility $\triangle(e|A_{i-1}) \doteq f(A_{i-1} \cup \{e\}) - f(A_{i-1})$, is known to be very effective. Let us denote this (centralized) greedy solution by A^g. When f is *integral* (i.e., $f : 2^V \to \mathbb{N}$) it is known that the size of the solution returned by the greedy algorithm $|A^g|$ is at most $H(\max_e f(\{e\}))|A^c|$, where $H(z)$ is the z-th harmonic number and is bounded by $H(z) \leq 1 + \ln z$ [Wol82]. Thus, we have

$$|A^g| \leq (1 + \ln(\max_e f(\{e\})))|A^c|,$$

and obtaining a better solution is hard under natural complexity theoretic assumptions [Fei98]. As it is standard practice, for our theoretical analysis to hold, we assume that f is an integral, monotone submodular function.

In many data summarization applications where the ground set V is large, the sequential greedy algorithm is impractical: either the data cannot be stored on a single computer or the centralized solution is too expensive in terms of computation time. Hence, we seek an algorithm for solving the submodular cover problem in a distributed manner.

6.1.1 Naive Approaches Towards Distributed Submodular Cover

The two naive approaches for distributed submodular maximization, introduced in Section 5.1.3, can be also applied to solve the distributed submodular cover problem: 1) we start with an empty solution. In each round, all machines – in parallel – find and communicate their best local candidate (condition on the current solution) to a central processor, who then identifies the globally best element, and communicates it back to all the m machines. 2) each machine i selects greedily enough elements from its partition V_i until it reaches at least Q/m utility. Then, all machines merge their solution.

However, as we discussed, the first approach requires exactly $|A^g|$ many rounds of communication, and therefore is impractical for large k and hence $|A^g|$. In the second approach, many machines may select redundant elements, and the merged solution may suffer from diminishing returns and never reach Q. Instead of aiming for Q/m, one could aim for a larger fraction, but it is not clear how to select this target value.

In Section 6.2, we introduce our solution DisCover, which requires few rounds of communication, while at the same time yielding a solution competitive with the centralized one.

6.2 DisCover Algorithm for Distributed Submodular Cover

On a high level, our main approach is to reduce the submodular cover to a sequence of cardinality constrained submodular maximization problems[1], a problem for which good distributed algorithms (e.g., GreeDi introduced in Chapter 5, and further analyzed in [Bar+15; MZ15]) are known. Concretely, our reduction is based on a combination of the following three ideas.

To get an intuition, we will first assume that we have access to an optimum algorithm which can solve cardinality constrained submodular maximization exactly, i.e., solve, for some specified ℓ,

$$A^c[\ell] = \arg\max_{|S|\leq\ell} f(S). \tag{6.2.1}$$

We will then consider how to solve the problem when, instead of $A^c[\ell]$, we only have access to an approximation algorithm for cardinality constrained maximization. Lastly, we will illustrate how we can parametrize our algorithm to trade-off the number of rounds of the distributed algorithm versus communication cost per round.

6.2.1 Estimating Size of the Optimal Solution

Momentarily, assume that we have access to an optimum algorithm OptCard(V, ℓ) for computing $A^c[\ell]$ on the ground set V. Then one simple way to solve the submodular cover problem would be to incrementally check for each $\ell = \{1, 2, 3, \ldots\}$ if $f(A^c[\ell]) \geq Q$. But this is very inefficient since it will take $k = |A^c|$ rounds of running the distributed algorithm for computing $A^c[\ell]$. A simple fix that we will follow is to instead start with $\ell = 1$ and double it until we find an ℓ such that $f(A^c[\ell]) \geq Q$. This way we are guaranteed to find a solution of size at most $2k$ in at most $\lceil \log_2(k) \rceil$ rounds of running $A^c[\ell]$. The pseudocode is given in Algorithm 4. However, in practice, we cannot run Algorithm 4. In particular, there is no efficient way to identify the optimum subset $A^c[\ell]$ in set V, unless P=NP. Hence, we need to rely on approximation algorithms.

[1]Note that while reduction from submodular coverage to submodular maximization has been used (e.g., [IB13]), the straightforward application to the distributed setting incurs large communication cost.

Algorithm 4: Approximate Submodular Cover

```
Input: Set $V$, constraint $Q$.
Output: Set $A$.
1: $\ell = 1$.
2: $A^{\text{c}}[\ell] = \textsc{OptCard}(V, \ell)$.
3: while $f(A^{\text{c}}[\ell]) < Q$ do
4:    $\ell = \ell \times 2$.
5:    $A^{\text{c}}[l] = \textsc{OptCard}(V, \ell)$.
6: end while
7: $A = A^{\text{c}}[\ell]$.
8: Return $A$.
```

Algorithm 5: Approximate OptCard

```
Input: Set $V$, #of partitions $m$, constraint $Q$, $\ell$.
Output: Set $A^{\text{dc}}[m]$.
1: $r = 0,\ A^{\text{gd}}[m, \ell] = \emptyset,\ A^{\text{dc}}[m] = \emptyset$.
2: while $f(A^{\text{dc}}[m]) < Q$ do
3:    $A = A^{\text{gd}}[m, \ell]$.
4:    $r = r + 1$.
5:    $A^{\text{gd}}[m, \ell] = \textsc{DisCard}(V, m, \ell, A)$.
6:    if $f(\{A^{\text{gd}}[m, \ell] \cup A\}) - f(A) \geq \lambda(Q - f(A))$ then
7:       $A^{\text{dc}}[m] = \{A^{\text{gd}}[m, \ell] \cup A\}$.
8:    else
9:       break
10:   end if
11: end while
12: Return $A^{\text{dc}}[m]$.
```

6.2.2 Handling Approximations for Submodular Maximization

Assume that there is a distributed algorithm DisCard(V, m, ℓ), for cardinality constrained submodular maximization, that runs on the dataset V with m machines and provides a set $A^{\text{gd}}[m, \ell]$ with λ-approximation guarantee to the optimal solution $A^{\text{c}}[\ell]$, i.e., $f(A^{\text{gd}}[m, \ell]) \geq \lambda f(A^{\text{c}}[\ell])$. Let us assume that we could run DisCard with the unknown value $\ell = k$. Then the solution we get satisfies $f(A^{\text{gd}}[m, k]) \geq \lambda Q$, and we are not guaranteed to achieve Q any longer. Now, what we can do (still under the assumption that we know k) is to repeatedly run DisCard in order to augment our solution set until we get the desired value Q. Note that for each invocation of

DisCard, to find a set of size $\ell = k$, we have to take into account the solutions A that we have accumulated so far. So, by overloading the notation, DisCard(V, m, ℓ, A) returns a set of size ℓ given that A has already been selected in previous rounds (i.e., DisCard optimizes the conditional submodular function $f(\cdot|A) : 2^{V\setminus A} \to \mathbb{R}_+$ with $f(S|A) = f(S \cap A) - f(A)$). Note that at every invocation –thanks to submodularity– DisCard increases the value of the solution by at least $\lambda(Q - f(A))$. Therefore, by running DisCard at most $\lceil \log(Q)/\lambda \rceil$ times we get Q.

Unfortunately, we do not know the optimum value k. So, we can feed an estimate ℓ of the size of the optimum solution k to DisCard. Now, again thanks to submodularity, DisCard can check whether this ℓ is good enough or not: if the improvement in the value of the solution is not at least $\lambda(Q - f(A))$ during the augmentation process, we can infer that ℓ is a too small estimate of k and we cannot get the desired value Q by using ℓ – so we apply the doubling strategy again.

Theorem 17. *Let* DisCard *be a distributed algorithm for cardinality-constrained submodular maximization with λ approximation guarantee. Then, Algorithm 4 (where* OptCard *is replaced with Approximate* OptCard*, Algorithm 5) runs in at most* $\lceil \log(k) + \log(Q)/\lambda + 1 \rceil$ *rounds and produces a solution of size at most* $\lceil 2k + 2\log(Q)k/\lambda \rceil$.

The proofs of all the theorems in this Chapter can be found in Appendix A.2.

6.2.3 Trading Off Communication Cost and Number of Rounds

While Algorithm 4 successfully finds a distributed solution $A^{\text{dc}}[m]$ with $f(A^{\text{dc}}[m]) \geq Q$, (*c.f.*, Theorem 17), the intermediate problem instances (i.e., invocations of DisCard) are required to select sets of size up to twice the size of the optimal solution k, and these solutions are communicated between all machines. Oftentimes, k is quite large and we do not want to have such a large communication cost per round. Now, instead of finding an $\ell \geq k$ what we can do is to find a smaller $\ell \geq \alpha k$, for $0 < \alpha \leq 1$ and augment these smaller sets in each round of Algorithm 5. This way, the communication cost reduces to an α fraction (per round), while the improvement in the value of the solution is at least $\alpha\lambda(Q - f(A^{\text{gd}}[m, \ell]))$. Consequently, we can trade-off the communication cost per round with the total number of rounds. As a positive side effect, for $\alpha < 1$, since in each invocation of DisCard it returns smaller sets, the final solution set size can potentially get closer to the optimum solution size k. For instance, for the extreme case of $\alpha = 1/k$ we recover the solution of the sequential greedy algorithm (up to $O(1/\lambda)$). We see this effect in our experimental results.

6.2.4 DisCover

The DisCover algorithm is shown in Algorithm 6. The algorithm proceeds in rounds, with communication between machines taking place only between successive rounds. In particular, DisCover takes the ground set V, the number of partitions m, and the trade-off parameter α. It starts with $\ell = 1$, and $A^{\text{dc}}[m] = \emptyset$. It then augments the set $A^{\text{dc}}[m]$ with set $A^{\text{gd}}[m, \ell]$ of at most ℓ new elements using an arbitrary distributed algorithm for submodular maximization under cardinality constraint, DisCard. If the gain from adding $A^{\text{gd}}[m, \ell]$ to $A^{\text{dc}}[m]$ is at least $\alpha\lambda(Q - f(A^{\text{gd}}[m, \ell]))$, then we continue augmenting $A^{\text{gd}}[m, \ell]$ with another set of at most ℓ elements. Otherwise, we double ℓ and restart the process with 2ℓ. We repeat this process until we get Q.

Algorithm 6: DisCover

```
Input: Set $V$, #of partitions $m$, constraint $Q$, trade off parameter $\alpha$.
Output: Set $A^{\text{dc}}[m]$.
 1: $A^{\text{dc}}[m] = \emptyset,\ r = 0$.
 2: while $f(A^{\text{dc}}[m]) < Q$ do
 3:    $r = r + 1$.
 4:    $A^{\text{gd}}[m, \ell] = \text{DisCard}(V, m, \ell, A^{\text{dc}}[m])$.
 5:    if $f(A^{\text{dc}}[m] \cup A^{\text{gd}}[m, \ell]) - f(A^{\text{dc}}[m]) \geq \alpha\lambda(Q - f(A^{\text{dc}}[m]))$ then
 6:       $A^{\text{dc}}[m] = \{A^{\text{dc}}[m] \cup A^{\text{gd}}[m, \ell]\}$.
 7:    else
 8:       $\ell = \ell \times 2$.
 9:    end if
10: end while
11: Return $A^{\text{dc}}[m]$.
```

Theorem 18. *Let* DisCard *be a distributed algorithm for cardinality-constrained submodular maximization with λ approximation guarantee. Then,* DisCover *runs in at most $\lceil \log(\alpha k) + \log(Q)/(\lambda\alpha) + 1 \rceil$ rounds and produces a solution of size $\lceil 2\alpha k + \log(Q) 2k/\lambda \rceil$.*

GreeDi as Subroutine: So far, we have assumed that a distributed algorithm DisCard that runs on m machines is given to us as a black box, which can be used to find sets of cardinality ℓ and obtain a λ-factor of the optimal solution. More concretely, we can use GreeDi, the distributed algorithm for maximizing submodular functions under a cardinality constraint that we introduced in Chapter 5 (outlined in Algorithm 2). Remember that GreeDi first distributes the ground set V to m machines. Then each machine i separately runs the standard greedy algorithm to produce a set $A_i^{\text{gc}}[\ell]$ of size ℓ. Finally, the solutions are merged, and another round of greedy selection is performed (over the merged results) in order to return the solution $A^{\text{gd}}[m,\ell]$ of size ℓ. In Chapter 5 (*c.f.* Theorem 5), we proved that GreeDi provides a $(1-e^{-1})^2/\min(m,\ell)$-approximation to the optimal solution. Here, we prove a (tight) improved bound on the performance of GreeDi. More formally, we have the following theorem.

Theorem 19. *Let f be a monotone submodular function and let $\ell > 0$. Then,* GreeDi *produces a solution $A^{gd}[m,\ell]$ where $f(A^{gd}[m,\ell]) \geq \frac{1}{36\sqrt{\min(m,\ell)}} f(A^c[\ell])$.*

We illustrate the resulting algorithm DisCover using GreeDi as subroutine in Figure 6.1. By combining Theorems 18 and 19, we will have the following.

Corollary 20. *By using* GreeDi, *we get that* DisCover *produces a solution of size $\lceil 2\alpha k + 72\log(Q)k\sqrt{\min(m,\alpha k)})\rceil$ and runs in at most $\lceil \log(\alpha k) + 36\sqrt{\min(m,\alpha k)}\log(Q)/\alpha + 1\rceil$ rounds.*

Note that for a constant number of machines m, $\alpha = 1$ and a large solution size $\alpha k \geq m$, the above result simply implies that in at most $O(\log(kQ))$ rounds, DisCover produces a solution of size $O(k\log Q)$. In contrast, the greedy solution with $O(k\log Q)$ rounds (which is much larger than $O(\log(kQ))$) produces a solution of the same quality.

Very recently, a $(1-e^{-1})/2$-approximation guarantee was proven for the randomized version of GreeDi [MZ15; Bar+15]. This suggests that, if it is possible to reshuffle (i.e., randomly re-distribute V among the m machines) the ground set each time that we revoke GreeDi, we can benefit from these stronger approximation guarantees (which are independent of m and k). Furthermore, stochastic version of the greedy algorithm that we will introduce in Chapter 12 can be incorporated into GreeDi to provide faster distributed frameworks for submodular cover. Note that Theorem 18 does not directly apply here, since it requires a deterministic subroutine for constrained submodular maximization.

As a final technical remark, for our theoretical results to hold we have assumed that the utility function f is integral. In some applications (like active set selection) this

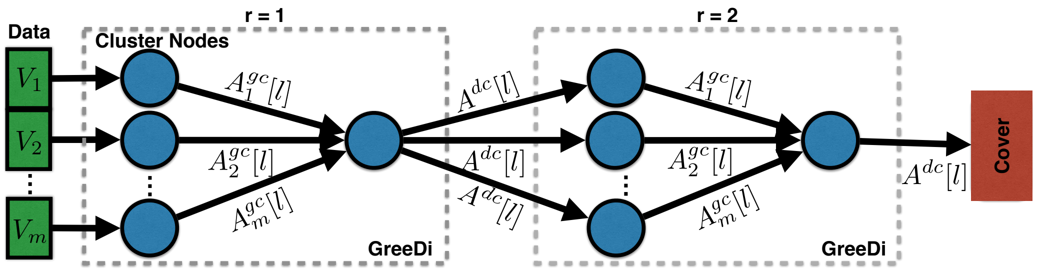


Figure 6.1: *Illustration of our multi-round algorithm* DisCover *, assuming it terminates in two rounds (without doubling search for ℓ).*

assumption may not hold. In these cases, either we can appropriately discretize and rescale the function, or instead of achieving the utility Q, try to reach $(1-\epsilon)Q$, for some $0 < \epsilon < 1$. In the latter case, we can simply replace Q with Q/ϵ in Theorem 18.

6.3 Experiments

In our experiments we wish to address the following questions: 1) How well does DisCover perform compare to the centralized greedy solution; 2) How is the trade-off between the solution size and the number of rounds affected by parameter α; and 3) How well does DisCover scale to massive datasets. To this end, we run DisCover on three scenarios: exemplar based clustering, active set selection in GPs, and dominating set problem. For dominating set, we report experiments on a large social graph with more than 65.6 million vertices and 1.8 billion edges. Since the constant in Theorem 19 is not optimized, we used $\lambda = 1/\sqrt{\min(m,k)}$ in all the experiments.

6.3.1 Exemplar based Clustering

Our exemplar based clustering experiments involve DisCover applied to the clustering utility $f(S)$ described in Section 3.2, with $d(x,x') = \|x - x'\|^2$. We perform our experiments on a set of 10,000 *Tiny Images* [TFF08]. Each 32 by 32 RGB pixel image is represented as a 3,072 dimentional vectors. We subtract from each vector the mean value, then normalize it to have unit norm. We use the origin as the auxiliary exemplar for this experiment. Figure 6.2a compares the performance of our approach to the centralized benchmark with the number of machines set to $m = 10$ and varying coverage percentage $Q = (1-\epsilon)f(V)$. Here, we have $\beta = (1-\epsilon)$. It can be seen that DisCover provides a solution which is very close to the centralized solution, with a number of rounds much smaller than the solution size. Varying α results in a tradeoff between solution size and number of rounds.

6.3.2 Active Set Selection

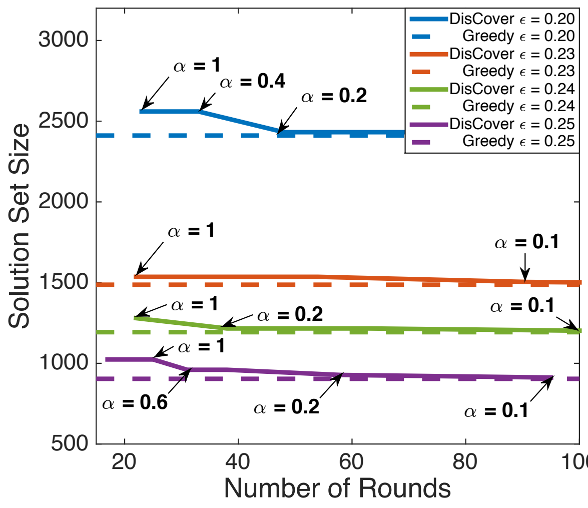
Our active set selection experiments involve DisCover applied to the log-determinant function $f(S)$ described in Section 3.1.1, using an exponential kernel $K(e_i, e_j) = \exp(-|e_i - e_j|^2/0.75)$. We use the *Parkinsons Telemonitoring* dataset [Tsa+10] comprised of 5,875 biomedical voice measurements with 22 attributes from people in early-stage Parkinson's disease. Figure 6.2b compares the performance of our approach to the benchmark with the number of machines set to $m = 6$ and varying coverage percentage $Q = (1 - \epsilon)f(V)$. Again, DisCover performs close to the centralized greedy solution, even with very few rounds. Again we see a tradeoff by varying α.

6.3.3 Large Scale Dominating Set with Spark

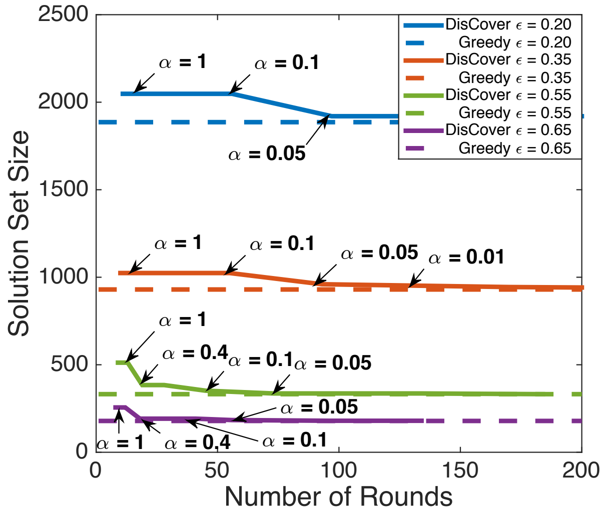
As our large scale experiment, we applied DisCover to the Friendster network consists of 65,608,366 nodes and 1,806,067,135 edges [YL15]. The average out-degree is 55.056 while the maximum out-degree is 5,214. The disk footprint of the graph is 30.7GB, stored in 246 part files on HDFS. Our experimental infrastructure was a cluster of 8 quad-core machines with 32GB of memory each, running Spark [Zah+10]. We set the number of reducers to $m = 64$.

Each machine carried out a set of map/reduce tasks in sequence, where each MapReduce stage corresponds to running GreeDi with a specific values of ℓ on the whole dataset. We first distributed the data uniformly at random to the machines, where each machine received ≈1,025,130 vertices (≈12.5GB RAM). Then we start with $\ell = 1$, perform a MapReduce task to extract one element. We then communicate back the results to each machine and based on the improvement in the value of the solution, we perform another round of map/reduce calculation with either the the same value for ℓ or $2 \times \ell$. We continue performing map/reduce tasks until we get the desired value Q.

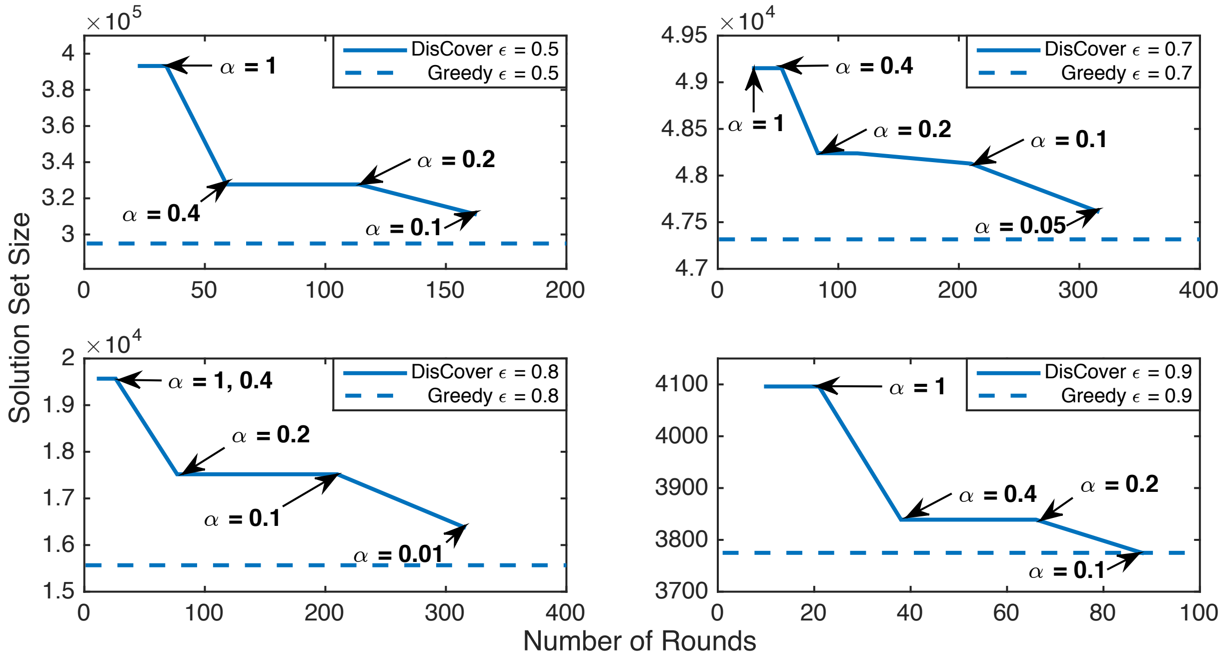
We examine the performance of DisCover by obtaining covers for 50%, 30%, 20% and 10% of the whole graph (*c.f.* Section 3.3). The total running time of the algorithm for the above coverage percentages with $\alpha = 1$ was about 5.5, 1.5, 0.6 and 0.1 hours



(a) Images 10K



(b) Parkinsons Telemonitoring



(c) Friendster

Figure 6.2: *Performance of* DisCover *compared to the centralized solution. a, b) show the solution set size vs. the number of rounds for various α, for a set of 10,000 Tiny Images and Parkinsons Telemonitoring. c) shows the same quantities for the Friendster network with 65,608,366 vertices.*

respectively. For comparison, we ran the centralized greedy on a computer of 24 cores and 256GB memory. Note that, loading the entire dataset into memory requires 200GB of RAM, and running the centralized greedy algorithm for 50% cover requires at least another 15GB of RAM. This highlights the challenges in applying the centralized greedy algorithm to larger scale datasets. Figure 6.2c shows the solution set size versus the number of rounds for various α and different coverage constraints. We find that by decreasing α, DisCover's solutions quickly converge (in size) to those obtained by the centralized solution.

6.4 Summary

We have developed the first efficient distributed algorithm –DisCover – for the submodular cover problem. We have theoretically analyzed its performance and showed that it can perform arbitrary close to the centralized (albeit impractical in context of large datasets) greedy solution. We also study a natural trade-off between the communication cost and the number of rounds required to obtain such a solution. We also demonstrated the effectiveness of our approach through extensive experiments, including dominating set on a graph with 65.6 million vertices using Spark. DisCover uses DisCard as a subroutine. More efficient distributed algorithms, e.g. GreeDi with Stream-Greedy [Chapter 12], or Fantom [Chapter 13] can be integrated into DisCover to improve its efficiency, and provide guarantees under more general constraints.

7 Fast Distributed Submodular Cover: Public-Private Data Summarization

In Chapter 6, we studies distributed algorithms for the submodular cover problem. In this chapter, we propose a fast distributed algorithm, FastCover, that enables us to solve the more general problem of covering multiple submodular functions in one run of the algorithm. It relies one three important ingredients:

1. a reduction from multiple submodular cover problems into a single instance of a submodular cover problem [Kra+08b; IB13],
2. randomized filtration mechanism to select elements with high utility, and
3. a set of carefully chosen threshold functions used for the filteration mechanism.

FastCover also provides a natural tarde-off between the number of MapReduce rounds and the size of the returned summary. It effectively lets us choose between compact summaries (i.e., smaller solution size) while running more MapReduce rounds or larger summaries while running fewer MapReduce rounds.

This setting is motivated by privacy concerns in many modern applications, including personalized recommender systems, online social services, and the data collected by apps on mobile platforms [Chi+15]. In such applications, users have some control over their own data and can mark some part of it private (in a slightly more general case, we can assume that users can make part of their data private to specific groups and public to others). As a result, the dataset consists of public data, shared among all users, and disjoint sets of private data accessible to the owners only.

We call this more general framework for data summarization, public-private data summarization, where the private data of one user should not be included in another user's summary (see also [Chi+15]). This model naturally reduces to solving one instance of the submodular cover problem for each user, as their view of the dataset and the specific utility function specifying users' preferences differ across users. When the number of users is small, one can solve the public-private data summarization separately for each user, using the greedy algorithm (for datasets of small size) or the distributed algorithm DisCover introduced in Chapter 6 (for datasets of moderate size). However, when there are many users or the dataset is massive, none of the prior work truly scales.

We report performance of FastCover using Spark [Zah+10] on concrete applications of the public-private data summarization, including personalized movie recommendation on a dataset containing 2 million ratings by more than 100K users for 1000 movies, personalized location recommendation based on 20 users and their collected GPS locations, and finding the dominating set on a social network containing more than 65 million nodes and 1.8 billion edges. For small to moderate sized datasets, we compare our results with previous work, namely, classical greedy algorithm and DisCover. For truly large-scale experiments, where the data is big and/or there are many users involved (e.g., movie recommendation), we cannot run DisCover as the number of MapReduce rounds in addition to their communication costs is prohibitive. In our experiments, we constantly observe that FastCover provides solutions of size similar to the greedy algorithm (and very often even smaller) with the number of rounds that are orders of magnitude smaller than DisCover. This makes FastCover the first distributed algorithm that solves the public-private data summarization fast and at scale.

7.1 Problem Statement: Public-Private Summarization

In this section, we formally define the public-private model of data summarization[1]. Here, we consider a potentially large dataset (sometimes called universe of items) $\mathbb{V}$ of size n and a set of users $\mathbb{U}$. The dataset consists of public data $\mathbb{V}^P$ and subsets of private data $\mathbb{V}_u$ for each user $u \in \mathbb{U}$.

The public-private aspect of data summarization realizes in two dimensions.

- First, each user $u \in \mathbb{U}$ has her own utility function $f_u(S)$ according to which she scores the value of a subset $S \subseteq \mathbb{V}$. Throughout this chapter we assume that $f_u(\cdot)$ is integer-valued[2], non-negative, and monotone submodular. $\Delta_{f_u}(e|A) \doteq f_u(A \cup \{e\}) - f_u(A)$ is the marginal gain (or added value) of e to the set A. Whenever it is clear from the context we drop f_u from $\Delta_{f_u}(e|A)$. Without loss of generality, we normalize all users' functions so that they achieve the same maximum value, i.e., $f_u(\mathbb{V}) = f_v(\mathbb{V})$ for all $u, v \in \mathbb{U}$.

- Second, and in contrast to public data that is shared among all users, the private data of a user cannot be shared with others. Thus, a user $u \in \mathbb{U}$ can only evaluate the public and her own private part of a summary S, i.e., $S \cap (\mathbb{V}^P \cup \mathbb{V}_u)$. In other words, if the summary S contains private data of a user $v \neq u$, the user u cannot have access or evaluate v's private part of S, i.e., $S \cap \mathbb{V}_v$.

In public-private data summarization, we would like to find the smallest subset $S \subseteq \mathbb{V}$ such that all users reach a desired utility $Q \leq f_u(\mathbb{V}) = f_u(\mathbb{V}^P \cup \mathbb{V}_u)$ simultaneously, i.e.,

$$\mathsf{OPT} = \arg\min_{S \subseteq \mathbb{V}} |S|, \text{ such that } f_u(S \cap (\mathbb{V}^P \cup \mathbb{V}_u)) \geq Q \ \forall u \in \mathbb{U}. \tag{7.1.1}$$

A naive way to solve the above problem is to find a separate summary for each user and then return the union of all summaries as S. A more clever way is to realize that

[1]All the results are applicable to submodular cover as a special case where there is only public data.

[2]For the submodular cover problem it is a standard assumption that the function is integer-values for the theoretical results to hold. In applications where this assumption is not satisfied, either we can appropriately discretize and rescale the function, or instead of achieving the desired utility Q, try to reach $(1-\delta)Q$, for some $0 < \delta < 1$. In the latter case, we can simply replace Q with Q/δ in the theorems to get the correct bounds.

problem (7.1.1) is in fact equivalent to the following problem [Kra+08b; IB13]

$$\mathsf{OPT} = \arg\min_{S \subseteq \mathbb{V}} |S|, \text{ such that } f(S) \doteq \sum_{u \in \mathbb{U}} \min\{f_u(S \cap (\mathbb{V}^P \cup \mathbb{V}_u)), Q\} \geq Q \times |\mathbb{U}|. \tag{7.1.2}$$

Note that the surrogate function $f(\cdot)$ is also monotone submodular as a thresholded submodular function remains submodular. Thus, finding a set S that provides each user with utility Q is equivalent of finding a set S with $f(S) \geq L \doteq Q \times |\mathbb{U}|$. This reduction lets us focus on developing a fast distributed solution for solving a single submodular cover problem. Our method FASTCOVER is explained in detail in Section 7.3.

7.2 Applications of Pubic-Private Data Summarization

In this section, we discuss 3 concrete applications where parts of data are private and the remaining parts are public. All objective functions are non-negative, monotone, and submodular.

7.2.1 Personalized Movie Recommendation

Consider a movie recommender system that allows users to anonymously and privately rate movies. The system can use this information to recognize users' preferences using existing matrix completion techniques [CR09]. A good set of recommended movies should meet two criteria: 1) be correlated with user's preferences, and 2) be diverse and contains globally popular movies. To this end, we define the following sum-coverage function to score the quality of the selected movies S for a user u:

$$f_u(S) = \alpha_u \sum_{i \in S, j \in \mathbb{V}_u} s_{i,j} + (1 - \alpha_u) \sum_{i \in S, j \in \mathbb{V}^P \setminus S} s_{i,j}, \tag{7.2.1}$$

where $\mathbb{V}_u$ is the list of highly ranked movies by user u (i.e., private information), $\mathbb{V}^P$ is the set of all movies in the database[3], and $s_{i,j}$ measures the similarity between movie i and j. The similarity can be easily calculated using the inner product between the corresponding feature vectors of any two movies i and j. The term $\sum_{i \in S, j \in \mathbb{V}_u} s_{i,j}$

[3]Two private lists may point to similar movies, but for now we treat the items on each list as unique entities.

measures the similarity between the recommended set S and the user's preferences. The second term $\sum_{i\in S, j\in \mathbb{V}^P\setminus S} s_{i,j}$ measures the similarity between the recommended set S and the popular movies in the public data. Finally, the parameter $0 \leq \alpha_u \leq 1$ provides the user the freedom to specify how much she cares about personalization versus diversity, i.e., $\alpha_u = 1$ indicates that all the recommended movies should be very similar to the movies she highly ranked and $\alpha_u = 0$ means that she prefers to receive a set of globally popular movies among all users, irrespective of her own private ratings.

Note that in this application, the universe of items (i.e., movies) is public. What is private is the users' ratings through which we identify the set of highly ranked movies by each user $\mathbb{V}_u$. The effect of private data is expressed in users' utility functions. The objective is to find the smallest set S of movies $\mathbb{V}$, from which we can build recommendations for all users in a way that all reach a certain utility.

7.2.2 Personalized Location Recommendation

Nowadays, many mobile apps collect geolocation data of their users. To comply with privacy concerns, some let their customers have control over their data, i.e., users can mark some part of their data private and disallow the app to share it with other users. In the personalized location recommendation, a user is interested in identifying a set of locations that are correlated with the places she visited and popular places everyone else visited.

Note that as close by locations are likely to be similar it is very typical to define a kernel matrix K capturing the similarity between data points. A commonly used kernel in practice is the squared exponential kernel $\mathcal{K}(e_i, e_j) = \exp(-||e_i - e_j||_2^2/h^2)$. To define the information gain of a set of locations indexed by S, it is natural to use $f(S) = \log\det(I + \sigma K_{S,S})$ (*c.f.* Section 3.1.1). The information gain objective captures the diversity and is used in many ML applications, e.g., active set selection for nonparametric learning [Section 3.1.1], sensor placement [Kra+08b], determinantal point processes [Section 3.1.2], among many others. Then, the personalized location recommendation can be modeled by

$$f_u(S) = \alpha_u f(S \cap \mathbb{V}_u) + (1 - \alpha_u) f(S \cap \mathbb{V}^P), \tag{7.2.2}$$

where $\mathbb{V}_u$ is the set of locations that user u does not want to share with others and $\mathbb{V}^P$ is the collection of all publicly disclosed locations. Again, the parameter α_u lets the user indicate to what extent she is willing to receive recommendations based on her private information. The objective is to find the smallest set of locations from which we can build recommendations for all users such that each reaches a desired threshold. Note that private data is usually small and private functions are fast to compute. Thus, the function evaluation is mainly affected by the amount of public data. Moreover, for many objectives, e.g., information gain, each machine can evaluate $f_u(S)$ by using its own portion of the private data.

7.3 FastCover for Fast Distributed Submodular Cover

In this section, we explain in detail our fast distributed Algorithm FastCover shown in Alg. 7. It receives a universe of items $\mathbb{V}$ and an integer-valued, non-negative, monotone submodular function $f : 2^{\mathbb{V}} \to \mathbb{R}_+$. Similar to the previous chapter, the objective is to find the smallest set S that achieves a value $f(S) \geq L$ (*c.f.* Eq. 6.1.1).

FastCover starts with $S = \emptyset$, and keeps adding those items $x \in \mathbb{V}$ to S whose marginal values $\Delta(e|S)$ are at least some threshold τ. In the beginning, τ is set to a conservative initial value $M \doteq \max_{x \in \mathbb{V}} f(x)$. When there are no more items with a marginal value τ, FastCover lowers τ by a factor of $(1-\epsilon)$, and iterates anew through the elements. Thus, τ ranges over $\tau_0 = M, \tau_1 = (1-\epsilon)M, \cdots, \tau_\ell = (1-\epsilon)^\ell M, \cdots$. FastCover terminates when $f(S) \geq L$. The parameter ϵ determines the size of the final solution. When ϵ is small, we expect to find better solutions (i.e., smaller in size) while having to spend more number of rounds.

One of the key ideas behind FastCover is that finding elements with marginal values $\tau = \tau_\ell$ can be done in a distributed manner. Effectively, FastCover partitions $\mathbb{V}$ into m sets $T_1, \ldots, T_m$, one for each cluster node/machine. A naive distributed implementation is the following. For a given set S (whose elements are communicated to all machines) each machine i finds all of its items $x \in T_i$ whose marginal values $\Delta(x|S)$ are larger than τ and send them all to a central machine (note that S is fixed on each machine). Then, this central machine sequentially augments S with elements whose marginal values are more than τ (here S changes by each insertion). The new elements of S are communicated back to all machines and they run the same procedure, this time with a smaller threshold $\tau(1-\epsilon)$.

The main problem with this approach is that there might be many items on each machine that satisfy the chosen threshold τ at each round (i.e., many more than $|\mathsf{OPT}|$).

A flood of such items from m machines overwhelms the central machine. Instead, what FastCover does is to enforce each machine to randomly pick only k items from their potentially big set of candidates (i.e., ThresholdSample algorithm shown in Alg. 8). The value k is carefully chosen (line 5). This way the number of items the central machine processes is never more than $O(m|\mathsf{OPT}|)$.

Algorithm 7: FastCover

```
**Input:** $\mathbb{V}$, $\epsilon$, $L$, and $m$
**Output:** $S \subseteq \mathbb{V}$ where $f(S) \geq L$
 1: Find a balanced partition $\{T_i\}_{i=1}^m$ of $\mathbb{V}$
 2: $S \leftarrow \emptyset$
 3: $\tau \leftarrow \max_{x \in \mathbb{V}} f(x)$
 4: while $\tau \geq 1$ do
 5:   $k \leftarrow \lceil (L - f(S))/\tau \rceil$
 6:   for $1 \leq i \leq m$ do
 7:     $<S_i, Full_i> \leftarrow ThresholdSample(i,\tau,k,S)$
 8:   end for
 9:   for $x \in \cup_{i=1}^m S_i$ do
10:     if $f(\{x\} \cup S) - f(S) \geq \tau$ then
11:       $S \leftarrow S \cup \{x\}$
12:       if $f(S) \geq L$ then
13:         Break
14:       end if
15:     end if
16:   end for
17:   if $\forall i : Full_i = False$ then
18:     $\tau \leftarrow \max\{1, (1-\epsilon)\tau\}$
19:   end if
20: end while
21: Return $S$
```

Theorem 21. FastCover *terminates with at most* $\log_{3/2}(n/(|OPT|m))(1+\log(M)/\epsilon) + \log_2(L)$ *rounds (with high probability) and a solution of size at most* $|OPT|\ln(L)/(1-\epsilon)$.

Although FastCover is distributed and unlike centralized algorithms does not enjoy the benefits of accessing all items together, its solution size is truly competitive with the greedy algorithm and is only away by a factor of $1/(1-\epsilon)$. Moreover, its number of rounds is logarithmic in n and L. This is in sharp contrast with the previously best known algorithm, DisCover (introduced in Chapter 6), where the number of rounds

Algorithm 8: ThresholdSample

```
Input: Index $i$, $\tau$, $k$, and $S$
Output: $S_i \subset T_i$ with $|S_i| \leq k$
 1: $S_i \leftarrow \emptyset$
 2: for $x \in T_i$ do
 3:     if $f(S \cup \{x\}) - f(S) \geq \tau$ then
 4:         $S_i \leftarrow S_i \cup \{x\}$
 5:     end if
 6: end for
 7: if $|S_i| \leq k$ then
 8:     Return $< S_i, False >$
 9: else
10:     $S_i \leftarrow k$ random items of $S_i$
11:     Return $< S_i, True >$
12: end if
```

scales with $\sqrt{\min(m, |OPT|)}$[4]. Thus, FastCover not only improves exponentially over DisCover in terms of speed but also its number of rounds decreases as the number of available machines m increases. Even though FastCover is a simple distributed algorithm, its performance analysis is technical and is deferred to Appendix A.3. Below, we provide the main ideas behind the proof of Theorem 21.

Proof sketch. *We say that an item has a high value if its marginal value to S is at least τ. We define an epoch to be the rounds during which τ does not change. In the last round of each epoch, all high value items are sent to the central machine (i.e., the set $\cup_{i=1}^{m} S_i$) because $Full_i$ is false for all machines. We also add every high value item to S in lines $10-11$. So, at the end of each epoch, marginal values of all items to S are less than τ. Since we reduce τ by a factor of $(1-\epsilon)$, we can always say that $\tau \geq (1-\epsilon)\max_{x\in\mathbb{V}} \Delta(x|S)$ which means we are only adding items that have almost the highest marginal values. By the classic analysis of greedy algorithm for submodular maximization, we can conclude that every item we add has an added value that is at least $(1-\epsilon)(L-f(S))/|OPT|$. Therefore, after adding $|OPT|\ln(L)/(1-\epsilon)$ items, $f(S)$ becomes at least L.*

To upper bound rounds, we divide the rounds into two groups. In a good round, the algorithm adds at least $\frac{k}{2}$ items to S. The rest are bad *rounds. In a good round, we add $k/2 \geq (L - f(S))/(2\tau)$ items, and each of them increases the value of S by τ. Therefore in a good round,*

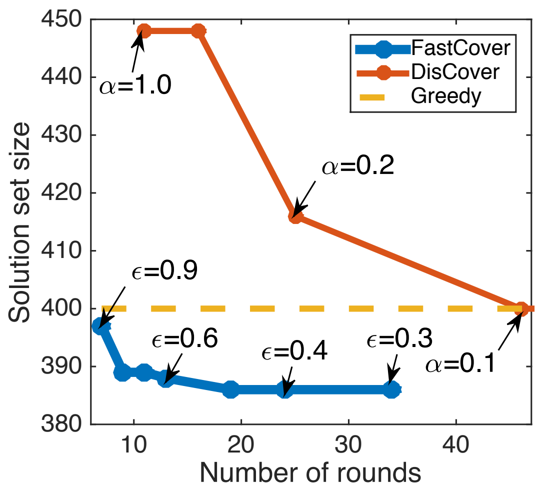
[4]Note that $\sqrt{\min(m, |OPT|)}$ can be as large as $n^{1/6}$ when $|OPT| = n^{1/3}$ and the memory limit of each machine is $n^{2/3}$ which results in $m \geq n^{1/3}$.

we see at least $(L - f(S))/2$ increase in value of S. In other words, the gap $L - f(S)$ is reduced by a factor of at least 2 in each good round. Since f only takes integer values, once $L - f(S)$ becomes less than 1, we know that $f(S) \geq L$. Therefore, there cannot be more than $\log_2 L$ good rounds. Every time we update τ (start of an epoch), we decrease it by a factor of $1 - \epsilon$ (except maybe the last round for which $\tau = 1$). Therefore, there are at most $1 + \log_{\frac{1}{1-\epsilon}}(M) \leq 1 + \frac{\log(M)}{\log(1/(1-\epsilon))} \leq 1 + \frac{\log(M)}{\epsilon}$ epochs. In a bad round, a machine with more than k high value items, sends k of those to the central machine, and at most $k/2$ of them are selected. In other words, the addition of these items to S in this bad round caused more than half of high value items of each machine to become of low value (marginal values less than τ). Since there are n/m items in each machine, and $Full_i$ becomes False once there are at most k high value items in the machine, we conclude that in expectation there should not be more than $\log_2(n/km)$ bad rounds in each epoch. Summarizing the upper bounds yields the bound on total number of rounds. Finer analysis leads to the high probability claim. □

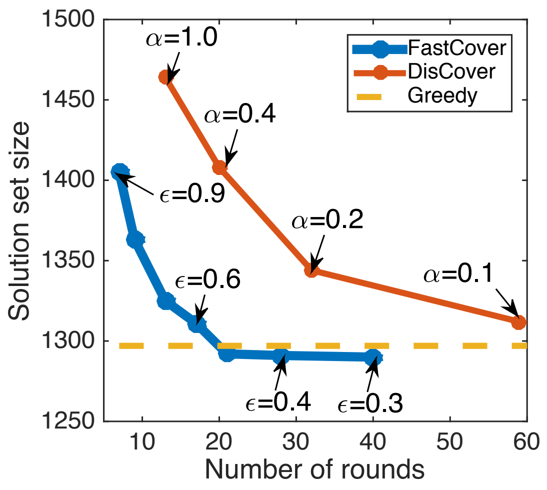
7.4 Experiments

In this section, we evaluate the performance of FastCover on the three applications that we described in Section 7.2: personalized movie recommendation, personalized location recommendation, and dominating set on social networks. To validate our theoretical results and demonstrate the effectiveness of FastCover, we compare the performance of our algorithm against DisCover (Alg. 6 from Chapter 6) and the centralized greedy algorithm (when possible).

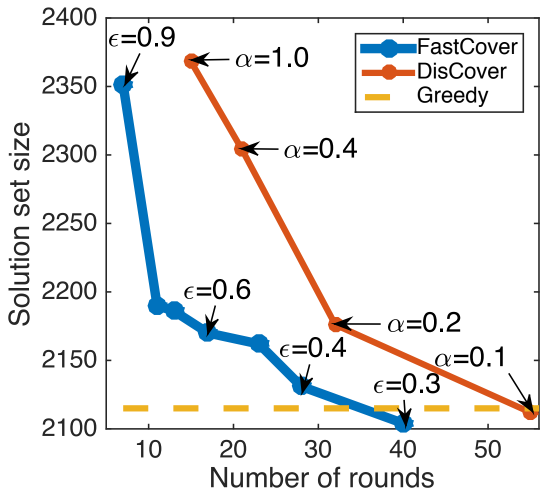
Our experimental infrastructure was a cluster of 16 quad-core machines with 20GB of memory each, running Spark [Zah+10]. The cluster was configured with one master node responsible for resource management, and the remaining 15 machines working as executors. We set the number of reducers to $m = 60$. To run FastCover on Spark, we first distributed the data uniformly at random to the machines, and performed a map/reduce task to find the highest marginal gain $\tau = M$. Each machine then carries out a set of map/reduce tasks in sequence, where each map/reduce stage filters out elements with a specific threshold τ on the whole dataset. We then tune the parameter τ, communicate back the results to the machines and perform another round of map/reduce calculation. We continue performing map/reduce tasks until we get to the desired value L.



(a) Location data (60%)



(b) Location data (80%)

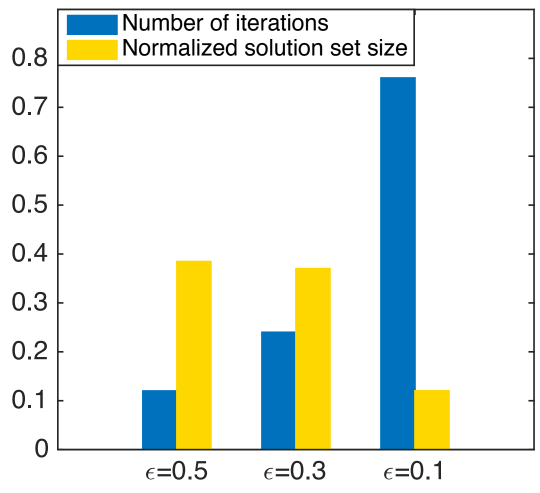


(c) Location data (90%)

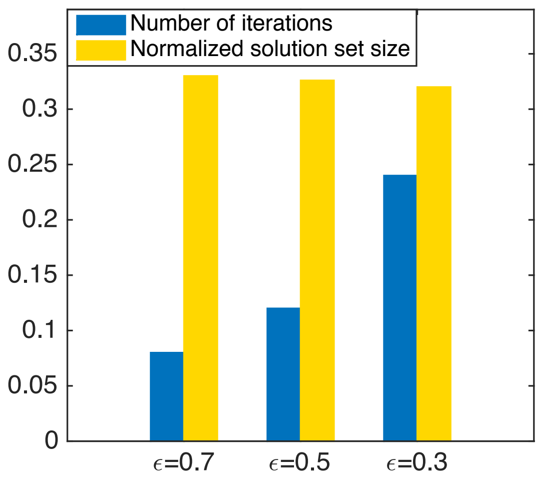
Figure 7.1: *Performance of* FastCover *vs. other baselines. a), b), c) solution set size vs. number of rounds for personalized location recommendation on a set of 3,056 GPS measurements, for covering 60%, 80%, 90% of the maximum utility of each user.*

7.4.1 Personalized Location Recommendation with Spark

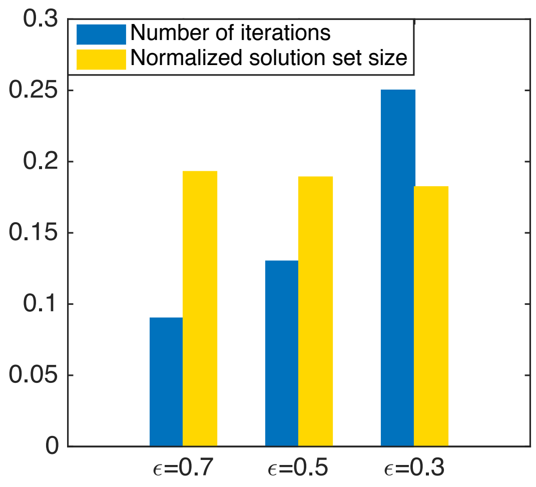
Our location recommendation experiment involves applying FastCover to the information gain utility function, described in Eq. (7.2.2). Our dataset consists of 3,056 GPS measurements from 20 users in the form of (latitude, longitude, altitude) collected during bike tours around Zurich [Tsa+10]. The size of each path is between 50 and 500 GPS coordinates. For each pairs of points i and j we used the corresponding GPS coordinates to calculate their distance in meters $d(i, j)$ and then formed a squared



(a) Movies (10%)



(b) Movies (20%)



(c) Movies (30%)

Figure 7.2: *Performance of* FastCover *vs. other baselines. Solution set size vs. number of rounds for personalized movie recommendation on a set of 1000 movies, 138,493 users and 20,000,263 ratings, for covering a) 10%, b) 20%, c) 30% of the maximum utility of each user.*

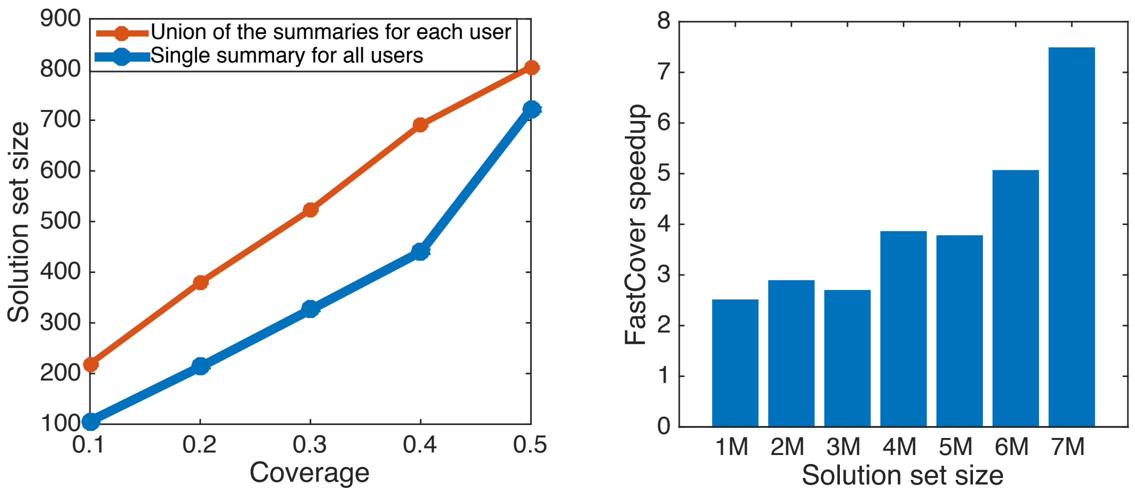
exponential kernel $\mathcal{K}_{i,j} = \exp(-d(i,j)^2/h^2)$ with $h = 1500$. For each user, we marked 20% of her data private (data points are chosen consecutively) selected from each path taken by the biker. The parameter α_u is set randomly for each user u.

Figures 7.1a, 7.1b, 7.1c compare the performance of FastCover to the benchmarks for building a recommendation set that covers 60%, 80%, and 90% of the maximum utility of each user. We considered running DisCover (Alg. 6) with different values of parameter α that makes a trade off between the size of the solution and number of rounds of the algorithm. It can be seen that by avoiding the doubling steps of DisCover, our algorithm FastCover is able to return a significantly smaller solution than that of DisCover in considerably fewer number of rounds. Interestingly, for small values of ϵ, FastCover returns a solution that is even smaller than the centralized greedy algorithm.

7.4.2 Personalized Movie Recommendation with Spark

Our personalized public-private recommendation experiment involves FastCover applied to a set of 1,313 movies, and 20,000,263 users' ratings from 138,493 users of the MovieLens database [Mov]. All selected users rated at least 20 movies. Each movie is associated with a 25 dimensional feature vector calculated from users' ratings. We use the inner product of the non-normalized feature vectors to compute the similarity $s_{i,j}$ between movies i and j [LWD15]. Our final objective function consists of 138,493 coverage functions -one per user- and a global sum-coverage function defined on the whole pool of movies (see Eq. (7.2.1)). Each function is normalized by its maximum value to make sure that all functions have the same scale.

Figures 7.2a, 7.2b, 7.2c show the ratio of the size of the solutions obtained by Fast-



(a) Movie (1K) (b) Friendster (14M)

Figure 7.3: *Performance of* FastCover *vs. other baselines. a) solution set size vs. coverage for simultaneously covering all users vs. covering users one by one and taking the union. The recommendation is on a set of 1000 movies for 1000 users. b) Exponential speedup of* FastCover *over* DisCover *on a subgraph of 14M nodes.*

Cover to that of the greedy algorithm. The figures demonstrate the results for 10%, 20%, and 30% covers for all the 138,493 users' utility functions. The parameter α_u is set to 0.7 for all users. We scaled down the number of iterations by a factor of 0.01, so that the corresponding bars can be shown in the same figures. Again, FastCover was able to find a considerably smaller solution than the centralized greedy. Here, we couldn't run DisCover because of its prohibitive running time on Spark.

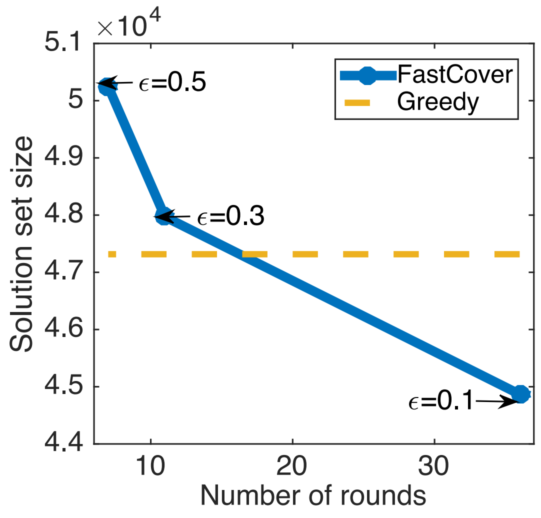
Figure 7.3a shows the size of the solution set obtained by FastCover for building recommendations from a set of 1000 movies for 1000 users vs. the size of the merged solutions found by finding recommendations separately for each user. It can be seen that FastCover was able to find a much smaller solution by covering all the functions at the same time.

7.4.3 Large Scale Dominating Set with Spark

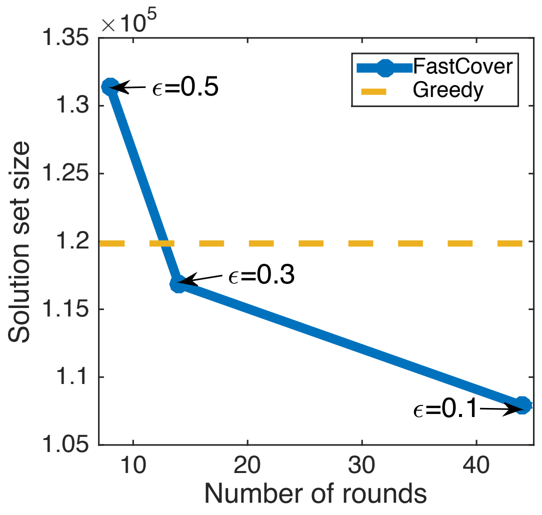
In order to be able to compare the performance of our algorithm with DisCover more precisely, we applied FastCover to the Friendster network consists of 65,608,366 nodes and 1,806,067,135 edges [YL15]. This dataset was used in Chapter 6 to evaluate the performance of DisCover. This is a trivial instance of public-private data summarization as all the data is public and there is a single utility function. We use the dominating set problem to run a large-scale application for which DisCover terminates in a reasonable amount of time and its performance can be compared to our algorithm FastCover.

Figures 7.4a, 7.4b, 7.4c show the performance of FastCover for obtaining covers for 50%, 40%, 30% of the whole graph, compared to the centralized greedy solution. Again, the size of the solution obtained by FastCover is smaller than the greedy algorithm for small values of ϵ. Note that running the centralized greedy is impractical if the dataset cannot fit into the memory of a single machine. Figure 7.4d compares the solution set size and the number of rounds for FastCover and DisCover with different values of ϵ and α. The points in the bottom left correspond to the solution obtained by FastCover which confirm its superior performance. We further measured the actual running time of both algorithms on a smaller instance of the same graph with 14,043,721 nodes. We tuned ϵ and α to get solutions of approximately equal size for both algorithms. Figure 7.3b shows the speedup of FastCover over DisCover.

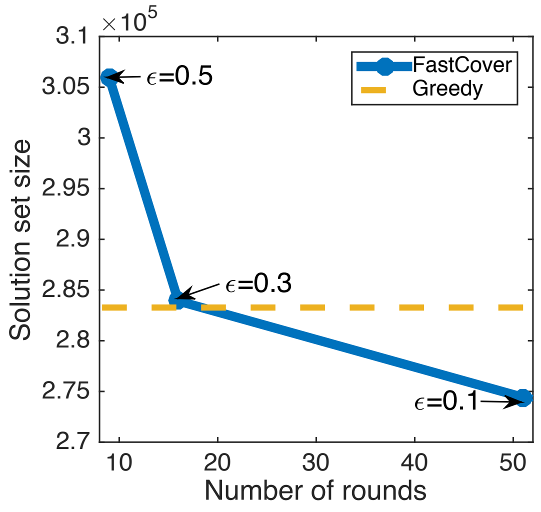
It can be observed that by increasing the coverage value L, FastCover shows an exponential speedup over DisCover.



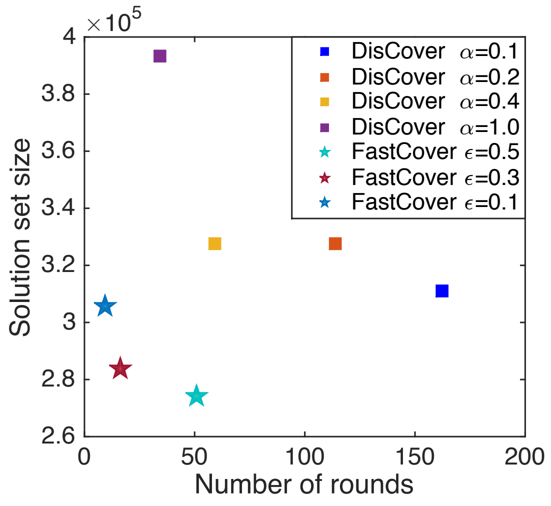
(a) Friendster (30%)



(b) Friendster (40%)



(c) Friendster (50%)



(d) Friendster (50%)

Figure 7.4: *Performance of* FastCover *vs. other baselines. Solution set size vs. the number of rounds for covering a) 30%, b) 40%, c) 50% of the Friendster network with 65,608,366 vertices. d) solution set size vs. the number of rounds for* FastCover *and* DisCover *for covering 50% of the Friendster network.*

7.5 Summary

In this chapter, we introduced the public-private model of data summarization motivated by privacy concerns of recommender systems. We also developed a fast distributed algorithm, FastCover, that provides a succinct summary for all users without violating their privacy. We showed that FastCover returns a solution that is competitive to that of the best centralized, polynomial-time algorithm (i.e., greedy solution). We also showed that FastCover runs exponentially faster than DisCover, the distributed algorithm introduced in the previous chapter. The superior practical performance of FastCover against all the benchmarks was demonstrated through a large set of experiments, including movie recommendation, location recommendation and dominating set (all were implemented with Spark). Our theoretical results combined with the practical performance of FastCover makes it the only existing distributed algorithm for the submodular cover problem that truly scales to massive data.

Part III

Streaming Algorithms

8 Overview of part III

In part II of the Thesis, we studied distributed algorithms for submodular summarization. In this part, we consider extracting representative elements from a large stream of data "on the fly". Such methods –with a limited memory available to them– can process quickly arriving data in a timely manner, facilitating real-time analytics.

Constrained streaming submodular maximization. The need for real time analysis of rapidly producing data streams (e.g., video and image streams) motivated the design of streaming algorithms that can efficiently extract and summarize useful information from massive data "on the fly". Such problems can often be reduced to maximizing a submodular set function subject to various constraints. While efficient streaming methods have been recently developed for monotone submodular maximization, in a wide range of applications, such as video summarization, the underlying utility function is non-monotone, and there are often various constraints imposed on the optimization problem to consider privacy or personalization. In Chapter 9 we develop the first efficient single pass streaming algorithm, Streaming Local Search, that for any streaming monotone submodular maximization algorithm with approximation guarantee α under a collection of independence systems $\mathcal{I}$, provides a constant $1/\big(1 + 2/\sqrt{\alpha} + 1/\alpha + 2d(1+\sqrt{\alpha})\big)$ approximation guarantee for maximizing a non-monotone submodular function under the intersection of $\mathcal{I}$ and d knapsack constraints. Our experiments show that for video summarization, our method runs more than 1700 times faster than previous work, while maintaining practically the same performance.

Deletion-robust streaming submodular maximization. How can we summarize a *dynamic* data stream when elements selected for the summary can be deleted at any time? This is an important challenge in online services, where the users generating the data may decide to exercise their right to restrict the service provider from using (part of) their data due to privacy concerns. Motivated by this challenge, we introduce the *dynamic deletion-robust submodular maximization* problem. In Chapter 10, we develop the first resilient streaming algorithm, called ROBUST-STREAMING, with a constant factor approximation guarantee to the optimum solution. We evaluate the effectiveness of our approach on several real-world applications, including summarizing (1) streams of geo-coordinates (2); streams of images; and (3) click-stream log data, consisting of 45 million feature vectors from a news recommendation task.

Summary of contributions. The key contributions of this part of the Thesis are:

1. We consider submodular optimization in a streaming setting, where at any point of time the algorithm has access only to a small fraction of data stored in primary memory. We develop novel, efficient approximation algorithms:
   - STREAMING LOCAL SEARCH, a one pass streaming algorithm for (non-monotone) submodular maximization under a collection of independence systems and d knapsacks constrains, and
   - ROBUST-STREAMING, a deletion robust method for summarizing a dynamic data stream when subsets of elements of the summary can be deleted at any time.
2. We theoretically analyze our approaches, and provide approximation guarantees for the quality of the solutions.
3. We demonstrate the performance of our algorithms on several real-world streaming problems, including
   - video summarization on YouTube and Open Video Project (OVP) datasets, where our method obtained more than 1700 times speedup compare to previous work,

- dynamic summarization of a collection of images based on a combination of 594 submodular functions with 20% deletion,
- dynamic summarization of a stream of geolocation data with 70% deletion, and
- large scale click through prediction based on 45,811,883 user visits from the Featured Tab of the Today Module on Yahoo! Front Page, with 99% deletion.

9 Constrained Streaming Submodular Maximization

Previously, we have seen that classical methods, such as the celebrated greedy algorithm [NWF78a] or its accelerated versions [Min78; BV14, Chapter 12] require random access to the entire data, make multiple passes, and select elements sequentially in order to produce near optimal solutions. Such centralized methods do not scale to large problems (where the data itself arrives at a fast pace) for several reasons: 1) the high volume of data does not fit in the main memory of the computing device (space limitation), 2) summaries cannot be updated instantaneously due to the sequential nature of these algorithms (time limitation), and 3) real-time decisions cannot be made as a result of multiple passes over the entire data (time-space limitation).

These limitations have recently inspired the design of streaming algorithms for constrained submodular maximization that are able to gain insights from data as it is generated [CGQ15; Bad+14; CK14]. These methods, while providing near-optimal solutions, only access a small fraction of data at any given time and are able to extract representative elements on the fly. Although efficient streaming methods have been recently developed for maximizing a monotone submodular function f with a variety of constraints, there is no effective solution for non-monotone submodular maximization under general types of constraints in the streaming setting.

In this chapter, we are motivated by applications of *non-monotone* submodular maximization. In particular, we consider video summarization in a streaming setting, where video frames are produced at a fast pace, and we want to keep an updated summary of the video so far, with little or no memory overhead. This has important applications e.g. in surveillance cameras, wearable cameras, and astro video cameras, which generate data at too rapid a pace to efficiently analyze and store it in main memory. The same framework can be applied more generally in many settings where we need to extract a small subset of data from a large stream to train or update a machine learning model. At the same time, various constraints may be imposed by the underlying summarization application. These may range from a simple limit on the size of the summary to more complex restrictions such as focusing on particular individuals or objects, or excluding them from the summary. These requirements often arise in real-world scenarios to consider privacy concerns (e.g. in case of surveillance cameras) or personalization (according to users' interests). We will discuss other concrete problem instances, with their corresponding non-monotone submodular objective functions, and various types of constraints in Chapter 13.

We provide STREAMING LOCAL SEARCH, the first single pass streaming algorithm for non-monotone submodular function maximization, subject to the intersection of a collection of independence systems $\mathcal{I}$ and d knapsack constraints. Our approach builds on local search, a widely used technique for maximizing non-monotone submodular functions in a batch mode. Local search, however, needs multiple passes over the input, and hence does not directly extend to the streaming setting, where we are only allowed to make a single pass over the data. This work provides a general framework into which we can plug in any streaming monotone submodular maximization algorithm INDSTREAM with approximation guarantee α under a collection of independence systems $\mathcal{I}$. In particular, any of the monotone streaming algorithms discussed in Section 2.3.2 can be used as INDSTREAM (See Table 2.3 for a list of single pass streaming algorithms). For any such monotone algorithm, STREAMING LOCAL SEARCH provides a constant $1/\big(1+2/\sqrt{\alpha}+1/\alpha+2d(1+\sqrt{\alpha})\big)$ approximation guarantee for maximizing a non-monotone submodular function under the intersection of $\mathcal{I}$ and d knapsack constraints. Furthermore, the memory and update time of STREAMING LOCAL SEARCH scales linearly with $O(\log(k)/\sqrt{\alpha})$ compare to INDSTREAM, where k is the size of the largest feasible solutions. Using parallel computation, the increase in the update time can be reduced to $O(1/\sqrt{\alpha})$, making our approach an appealing solution in real-time scenarios.

We show that for video summarization, our algorithm leads to streaming solutions that provide competitive utility when compared with those obtained via centralized methods, at a small fraction of the computational cost, i.e. more than 1700 times faster.

9.1 Streaming Submodular Maximization

We consider the problem of summarizing a stream of data V, by selecting, on the fly, a subset that maximizes a (non-monotone) submodular utility function f subject to a set ζ of constraints:

$$S^* = \arg\max_{S \in \zeta} f(S)$$

In this work, we consider a collection of independence systems and multiple knapsack constraints. Matroids, matchoids, and p-systems are independence systems with additional properties (See Section 2.2.2 for definitions). For knapsack constraints, we use c_{ij} to denote the cost of element $j \in V$ in the i-th knapsack. Without loss of generality, throughout this chapter we assume that all the knapsack budgets are 1. I.e., we would like to find a set $S \in \zeta$ that maximizes f where for each knapsack $i \in [d]$, we have $\sum_{e \in S} c_i(e) \leq 1$.

In context of streaming submodular maximization we assume that the ground set $V = \{e_1, \cdots, e_n\}$ is received from the stream in some arbitrary order. At each point t in time, the algorithm may maintain a memory $M_t \subset V$ of points, and must be ready to output a candidate feasible solution $S_t \subset M_t$, such that $S_t \in \zeta$. Upon receiving an element e_t from the stream, the algorithm may elect to 1) insert it into its memory, 2) discard some elements in its memory and accept e_t instead, or 3) discard e_t.

The performance of a streaming algorithm is measured by four basic parameters: 1) the *number of passes* the algorithm needs to make over the data stream, 2) the *memory* required by the algorithm (i.e., $\max_t |M_t|$), 3) the *running time* of the algorithm, in particular the number of oracle queries (evaluations of f) made, 4) the *approximation ratio*, i.e., $f(S_T)/\mathsf{OPT}$ where S_T is the final solution produced by the algorithm[1].

[1]Note that T can be bigger than n, if the algorithm makes multiple passes over the data.

9.2 Video Summarization with DPPs

Suppose that we are receiving a stream of video frames, e.g. from a surveillance or a wearable camera, and we wish to select a subset of frames that concisely represents all the diversity contained in the video. Determinantal Point Processes (DPPs), discussed in Section 3.1.2, are good tools for modeling diversity in such applications. The most diverse and informative subset of frames can be found by maximizing a non-monotone submodular function $f(S) = \log\det(L_S)$, where L is a positive semidefinite kernel matrix that captures the similarity between data points [KT+12].

Various constraints can be imposed while maximizing the above non-monotone submodular utility function. In its simplest form, we can partition the video into T segments, and define a diversity-reinforcing partition matroid to select at most k frames from each segment. Alternatively, various content-based constraints can be applied, e.g., we can use object recognition to select at most $0 \leq k_i$ frames from person i in the video, or to find a summary that is focused on a particular person or object. Finally, each frame can be associated with multiple costs, based on qualitative factors such as resolution, contrast, luminance, or the probability that the given frame contains an object. Multiple knapsack constraints, one for each quality factor, can then limit the total costs of the elements of the solution and enable us to produce a summary closer to human-created summaries by filtering uninformative frames.

9.3 Streaming algorithm for constrained submodular maximization

In this section, we describe our streaming algorithm for maximizing a non-monotone submodular function subject to the intersection of a collection of independence systems and d knapsack constraints. Our approach builds on local search, which is a powerful and widely used technique for maximizing non-monotone submodular functions. It starts from a candidate solution S and iteratively increases the value of the solution by either including a new element in S or discarding one of the elements of S [FMV11]. Gupta et al. [Gup+10a] showed that similar results can be obtained with much lower complexity by using algorithms for *monotone* submodular maximization, which, however, are run multiple times. Despite their effectiveness, these algorithms need multiple passes over the input and do not directly extend to the streaming setting, where we are only allowed to make a single pass over the data. In the sequel, we show how local search can be implemented in a single pass in the streaming setting.

9.3.1 Streaming Local Search for a collection of independence systems

The simple yet crucial observation underlying the approach of Gupta et al. [Gup+10a] is the following. The solution obtained by approximation algorithms for monotone submodular functions often satisfy $f(S) \geq \alpha f(S \cup C^*)$, where $1 \geq \alpha > 0$, and C^* is the optimal solution. In the monotone case $f(S \cup C^*) \geq f(C^*)$, and we obtain the desired approximation factor $f(S) \geq \alpha f(C^*)$. However, this does not hold for non-monotone functions. But, if $f(S \cap C^*)$ provides a good fraction of the optimal solution, then we can find a near-optimal solution for *non-monotone* functions even from the result of an algorithm for *monotone* functions, by pruning elements in S using unconstrained maximization. This still retains a feasible set, since the constraints are downward closed. Otherwise, if $f(S \cap C^*) \leq \epsilon \mathsf{OPT}$, then running another round of the algorithm on the remainder of the ground set will lead to a good solution.

Backed by the above intuition, we aim to build multiple disjoint solutions simultane-

Algorithm 9: STREAMING LOCAL SEARCH for independence systems

```
Input: $f : 2^V \to \mathbb{R}_+$, a membership oracle for independence systems $\mathcal{I} \subset 2^V$; and
    a monotone streaming algorithm INDSTREAM with $\alpha$-approximation under $\mathcal{I}$.
Output: A set $S \subseteq V$ satisfying $S \in \mathcal{I}$.
 1: while stream is not empty do
 2:    ▷ $e$ is the next element from the stream
 3:    $D_0 \leftarrow \{e\}$
 4:    ▷ LOCAL SEARCH iterations
 5:    for $i = 1$ to $\lceil 1/\sqrt{\alpha} + 1 \rceil$ do
 6:       ▷ $D_i$ is the discarded set by INDSTREAM$_i$
 7:       $[D_i, S_i]$= INDSTREAM$_i(D_{i-1})$
 8:       $S'_i$ =UNCONSTRAINED-MAX$(S_i)$.
 9:    end for
10:    $S = \arg\max_i \{f(S_i), f(S'_i)\}$
11: end while
12: Return $S$
```

ously within a single pass over the data. Let IndStream be a single pass streaming algorithm for monotone submodular maximization under a collection of independence systems, with approximation factor α. In particular, any of the monotone streaming algorithms listed in Table 2.3 can be used as IndStream. Upon receiving a new element from the stream, IndStream can choose (1) to insert it into its memory, (2) to replace one or a subset of elements in the memory by it, or otherwise (3) the element gets discarded and cannot be used later by the algorithm. The key insight for our approach is that it is possible to build other solutions from the elements discarded by IndStream. Consider a chain of $q = \lceil 1/\sqrt{\alpha} + 1 \rceil$ instances of our streaming algorithm, i.e. $\{\text{IndStream}_1, \cdots, \text{IndStream}_q\}$. Any element e received from the stream is first passed to IndStream_1. If IndStream_1 discards e, or adds e to its solution and instead discards a set D_1 of elements from its memory, then we pass the set D_1 of discarded elements on to be processed by IndStream_2. Similarly, if a set of elements D_2 is discarded by IndStream_2, we pass them to IndStream_3, and so on. The elements discarded by the last instance IndStream_q are discarded forever. Finally, at any point in time that we want to return the final solution, we run unconstrained submodular maximization (e.g. the algorithm of [Buc+15]) on each solution S_i obtained by IndStream_i to get S'_i, and return the best solution among $\{S_i, S'_i\}$ for $i \in [1, q]$.

Theorem 22. *Let IndStream be a streaming algorithm for monotone submodular maximization under a collection of independence systems $\mathcal{I}$ with approximation guarantee α. Algorithm 9 returns a set $S \in \mathcal{I}$ with*

$$f(S) \geq \frac{1}{(1 + 1/\sqrt{\alpha})^2} OPT,$$

using memory $O(M/\sqrt{\alpha})$, and average update time $O(T/\sqrt{\alpha})$ per element, where M and T are the memory and update time of IndStream.

The proofs of the theorems in this chapter can be found in Appendix A.4.

We make Theorem 22 concrete via an example: Chekuri et al [CGQ15] proposed a $1/4p$-approximation algorithm for maximizing a monotone submodular function under a p-matchoid constraint in the streaming setting. Using this algorithm as IndStream in our Streaming Local Search, we obtain the following result:

Corollary 23. *With Streaming Greedy of [CGQ15] as IndStream, Streaming Local Search yields a solution $S \in \mathcal{I}$ with approximation guarantee $1/(1 + 2\sqrt{p})^2$, using*

$O(\sqrt{p}k\log(k)/\epsilon)$ memory and $O(p\sqrt{p}k\log(k)/\epsilon)$ average update time per element, where $\mathcal{I}$ are the independent sets of a p-matchoid, and k is the size of the largest feasible solution.

The above $1/(4p+4\sqrt{p}+1)$ approximation is a significant improvement over the work of [CGQ15], for maximizing a non-monotone function under a p-matchoid constraint. Note that any monotone streaming algorithm with approximation guarantee α under a collection of independence systems $\mathcal{I}$ can be integrated into Algorithm 9 to provide approximation guarantees for non-monotone submodular maximization under the same set $\mathcal{I}$ of constraints. For example, as soon as there is a subroutine for monotone streaming submodular maximization under a p-system in the literature, one can use it in Algorithm 9 as IndStream, and get the guarantee provided in Theorem 22 for maximizing a non-monotone submodular function under a p-system, in the streaming setting.

9.3.2 Streaming Local Search for independence systems and multiple knapsack constraints

To respect multiple knapsack constraints in addition to the collection of independence systems $\mathcal{I}$, we integrate the idea of a density threshold [BV14; Svi04] into our local search algorithm. We use a (fixed) density threshold ρ to restrict the IndStream algorithm to only pick elements if the function value per unit size of the selected elements is above the given threshold. We call this new algorithm IndStreamDensity. The threshold should be carefully chosen to be below the value/size ratio of the optimal solution. To do so, we need to know (a good approximation to) the value of the optimal solution OPT. To obtain a rough estimate of OPT, it suffices to know the maximum value $m = \max_{e\in V} f(e)$ of any singleton element: submodularity implies that $m \leq \mathsf{OPT} \leq km$, where k is an upper bound on the cardinality of the largest feasible solution satisfying all constraints. We update the value of the maximum singleton element on the fly [Bad+14], and lazily instantiate the thresholds to $\log(k)/\epsilon$ different possible values $(1+\epsilon)^i \in [\gamma, \gamma k]$, for γ defined in Algorithm 10. We show that for at least one of the discretized density thresholds we obtain a good enough solution.

Theorem 24. Streaming Local Search *(outlined in Alg. 10) has an approximation*

Algorithm 10: STREAMING LOCAL SEARCH for independence systems $\mathcal{I}$ and d knapsacks

```
Input: $f : 2^V \to \mathbb{R}_+$, a membership oracle for independence systems $\mathcal{I} \subset 2^V$; $d$
    knapsack-cost functions $c_j : V \to [0,1]$; and an upper bound $k$ on the cardinality
    of the largest feasible solution.
Output: A set $S \subseteq V$ satisfying $S \in \mathcal{I}$ and $c_j(S) \leq 1 \ \forall j$.
 1: $m = 0$.
 2: while stream is not empty do
 3:   $e$ is the next element from the stream
 4:   $D_0 \leftarrow \{e\}$
 5:   $m = \max(m, f(e))$, $e_m = \arg\max_{e \in V} f(e)$.
 6:   $\gamma = \frac{2 \cdot m}{(1+1/\sqrt{\alpha})(1+1/\sqrt{\alpha}+2d\sqrt{\alpha})}$
 7:   $R = \{\gamma, (1+\epsilon)\gamma, (1+\epsilon)^2\gamma, (1+\epsilon)^3\gamma, \ldots, \gamma k\}$
 8:   for $\rho \in R$ in parallel do
 9:     ▷ LOCAL SEARCH
10:     for $i = 1$ to $\lceil 1/\sqrt{\alpha} + 1 \rceil$ do
11:       ▷ picks elements only if $\frac{f_{S_i}(e)}{\sum_{j=1}^{d} c_{je}} \geq \rho$
12:       $[D_i, S_i]$= INDSTREAMDENSITY$_i(D_{i-1}, \rho)$
13:       ▷ unconstrained submodular maximization
14:       $S_i'$ =UNCONSTRAINED-MAX$(S_i)$.
15:     end for
16:     $S_\rho = \arg\max_i \{f(S_i), f(S_i')\}$
17:   end for
18:   $S = \arg\max_{\rho \in R} f(S_\rho)$
19: end while
20: Return $\arg\max\{f(S), f(\{e_m\})\}$
```

guarantee

$$f(S) \geq \frac{1-\epsilon}{(1+1/\sqrt{\alpha})(1+2d\sqrt{\alpha}+1/\sqrt{\alpha})} OPT,$$

with memory $O(M \log(k)/(\epsilon\sqrt{\alpha}))$, and average update time $O(T \log(k)/(\epsilon\sqrt{\alpha}))$ per element, where k is an upper bound on the size of the largest feasible solution, and M and T are the memory and update time of the INDSTREAM *algorithm.*

Corollary 25. *By using* STREAMING GREEDY *of [CGQ15], we get that* STREAMING LOCAL SEARCH *has an approximation ratio $(1+\epsilon)(1+4p+4\sqrt{p}+d(2+1/\sqrt{p}))$ with $O(\sqrt{p}k\log^2(k)/\epsilon^2)$ memory and update time $O(p\sqrt{p}k\log^2(k)/\epsilon^2)$ per element, where $\mathcal{I}$ are* *the independent sets of the p-matchoid constraint, and k is the size of the largest feasible solution.*

Beyond the Black-Box. Although the DPP probability in Eq. 3.1.4 only depends on the selected subset S, in many applications $f(S)$ may depend on the entire data set V. So far, we have adopted the common assumption that f is given in terms of a value oracle (a black box) that computes $f(S)$. Although in practical settings this assumption might be violated, as we discussed in Section 5.2.5, many objective functions are *additively decomposable* over the ground set V. That means, $f(S) = \frac{1}{V}\sum_{e\in V} f_e(S)$, where $f_e(S)$ is a non-negative submodular function associated with every data point $e \in V$, and $f_e(.)$ can be evaluated without access to the full set V. For decomposable functions, we can approximate $f(S)$ by

$$f_W(S) = \frac{1}{W}\sum_{e\in W} f_e(S),$$

where W is a uniform sample from the stream (e.g. using reservoir sampling [Vit85]).

Theorem 26 (Badanidiyuru et al. [Bad+14]). *Assume that f is decomposable, all of $f_e(S)$ are bounded, and w.l.o.g. $|f_e(S)| \leq 1$. Let W be uniformly sampled from V. Then for $|W| \geq \frac{2k^2\log(2/\delta)+2k^3\log(V)}{\varepsilon^2}$, we can ensure that with probability $1-\delta$,* Streaming Local Search *guarantees*

$$f(S) \geq \frac{1-\epsilon}{(1+1/\sqrt{\alpha})(1+2d\sqrt{\alpha}+1/\sqrt{\alpha})}(OPT-\varepsilon).$$

9.4 Experiments

In this section, we apply Streaming Local Search to video summarization in the streaming setting. The main goal of this section is to validate our theoretical results and demonstrate the effectiveness of our method in practical scenarios, where the existing streaming algorithms are incapable of providing any guarantee for the quality of the solution. In particular, for streaming non-monotone submodular maximization under a collection of independence systems and multiple knapsack constraints, none of the previous works provide any theoretical guarantees. We use the streaming algorithm of [CGQ15] for monotone submodular maximization under a p-matchoid constraint as IndStream, and compare the performance of our method with exhaustive search [Gon+14], and a centralized method for maximizing a non-monotone submodular function under a p-system and multiple knapsack constraints, FANTOM, that will be introduced in Chapter 13.

Dataset. For our experiments, we use the Open Video Project (OVP), and the YouTube datasets with 50 and 39 videos, respectively [DA+11]. We use the pruned video frames as described in [Gon+14], where one frame is uniformly sampled per second, and uninformative frames are removed. Each video frame is then associated with a feature vector that consists of Fisher vectors [PD07] computed from SIFT features [Low04], contextual features, and features computed from the frame saliency map [Rah+10]. The size of the feature vectors, v_i, are 861 and 1581 for the OVP and YouTube dataset, respectively.

The DPP kernel L (*c.f.* Section 3.1.2), can be parametrized and learned via maximum likelihood estimation [Gon+14]. For parametrization, we follow [Gon+14], and use both a linear transformation, i.e. $L_{ij} = v_i^T W^T W v_i$, as well as a non-linear transformation using a one-hidden-layer neural network, i.e. $L_{ij} = z_i^T W^T W z_j$ where $z_i = \tanh(Uv_i)$, and $\tanh(.)$ stands for the hyperbolic transfer function. The parameters, U and W or just W, are learned on 80% of the videos, selected uniformly at random. By the construction of [Gon+14], we have $\det(L) > 0$. However, $\det(L)$ can take values less than 1, and the function is non-monotone. We added a positive constant to the function values to make them non-negative. Following [Gon+14] for evaluation, we treat each of the 5 human-created summaries per video as ground truth for each video.

Sequential DPP. To capture the sequential structure in video data, [Gon+14] proposed a sequential DPP. Here, a long video sequence is partitioned into T disjoint yet consecutive short segments, and for selecting a subset S_t from each segment $t \in [1, T]$, a DPP is imposed over the union of the frames in the segment t and the selected subset S_{t-1} in the immediate past frame $t-1$. The conditional distribution of the selected subset from segment t is thus given by $\mathcal{P}(S_t|S_{t-1}) = \frac{\det(K_{S_t \cup S_{t-1}})}{\det(I_t + K_{S_{t-1} \cup V_t})}$, where V_t denotes all the video frames in segment t, and I_t is a diagonal matrix in which the elements corresponding to S_{t-1} are zeros and the elements corresponding to S_t are 1. Intuitively, the sequential DPP only captures the diversity between the frames in segment t, and the selected subset S_{t-1} from the immediate past segment $t-1$. MAP inference for the sequential DPP is as hard as for the standard DPP, but submodular optimization

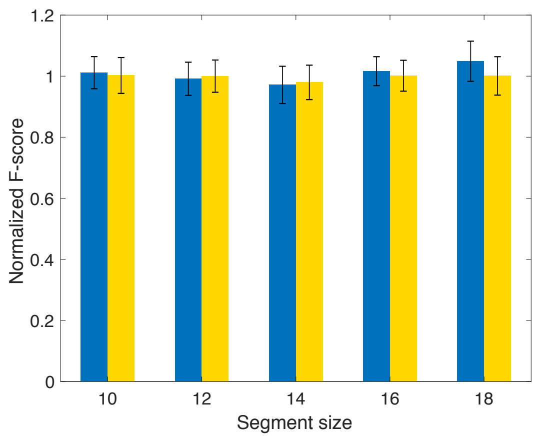
Table 9.1: *Performance of various video summarization methods with segment size 10 on YouTube and OVP datasets, measured by F-Score (F), Precision (P), and Recall (R). The methods marked by (c) are centralized.*

		Alg. of [Gon+14]$^{(c)}$		FANTOM [ch. 13]$^{(c)}$		STREAMING LS	
		Linear	N. Nets	Linear	N. Nets	Linear	N. Nets
YouTube	F	57.8±0.5	60.3±0.5	57.7±0.5	60.3±0.5	58.3±0.5	59.8±0.5
	P	54.2±0.7	59.4±0.6	54.1±0.5	59.1±0.6	55.2±0.5	58.6±0.6
	R	69.8±0.5	64.9±0.5	70.1±0.5	64.7±0.5	70.1±0.5	64.2±0.5
OVP	F	75.5±0.4	77.7±0.4	75.5±0.3	78.0±0.5	74.6±0.2	75.6±0.5
	P	77.5±0.5	75.0±0.5	77.4±0.3	75.1±0.7	76.7±0.2	71.8±0.7
	R	78.4±0.5	87.2±0.3	78.4±0.3	88.6±0.2	76.5±0.3	86.5±0.2

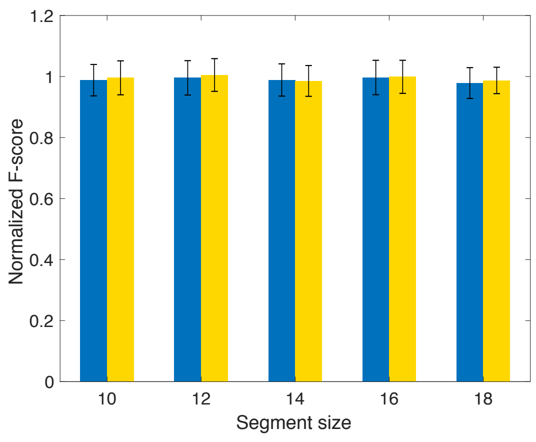
techniques can be used to find approximate solutions. In our experiments, we use a sequential DPP as the utility function in all the algorithms.

Results. Table 9.1 shows the F-score, Precision and Recall for our algorithm, that of [Gon+14] and FANTOM (Alg. 15 from Chapter 13), for segment size $|V_t| = 10$. It can be seen that in all three metrics, the summaries generated by STREAMING LOCAL SEARCH are competitive to the two centralized baselines.

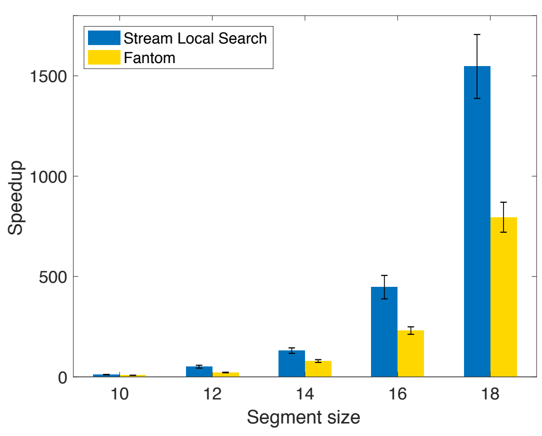
Figures 9.1a, 9.2a show the ratio of the F-score obtained by STREAMING LOCAL SEARCH and FANTOM vs. the F-score obtained by exhaustive search [Gon+14] for varying segment sizes, using linear embeddings on the YouTube and OVP datasets. It can be observed that our streaming method achieves the same solution quality as the centralized baselines. Figures 9.1a, 9.2a show the speedup of STREAMING LOCAL SEARCH and FANTOM over the method of [Gon+14], for varying segment sizes. We note that both FANTOM and STREAMING LOCAL SEARCH obtain a speedup that is exponential in the segment size. In summary, STREAMING LOCAL SEARCH achieves solution qualities comparable to [Gon+14], but 1700 times faster than [Gon+14], and 2 times faster than FANTOM for larger segment size. This makes our streaming method an appealing solution for extracting real-time summaries. In real-world scenarios, video frames are typically generated at such a fast pace that larger segments make sense. Moreover, unlike the centralized baselines that need to first buffer an entire segment, and then produce summaries, our method generates real-time summaries after receiving each video frame. This capability is crucial in privacy sensitive applications.



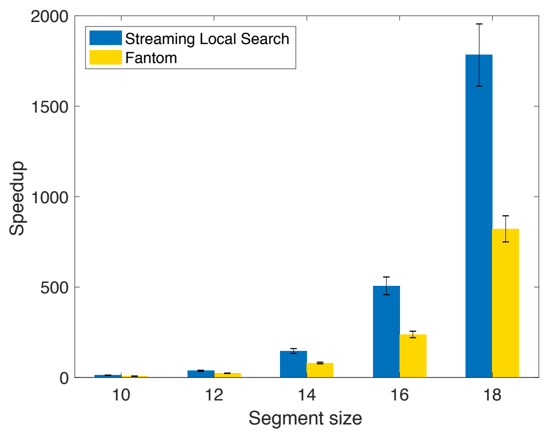
(a) YouTube Linear



(b) YouTube N. Nets



(c) YouTube Linear

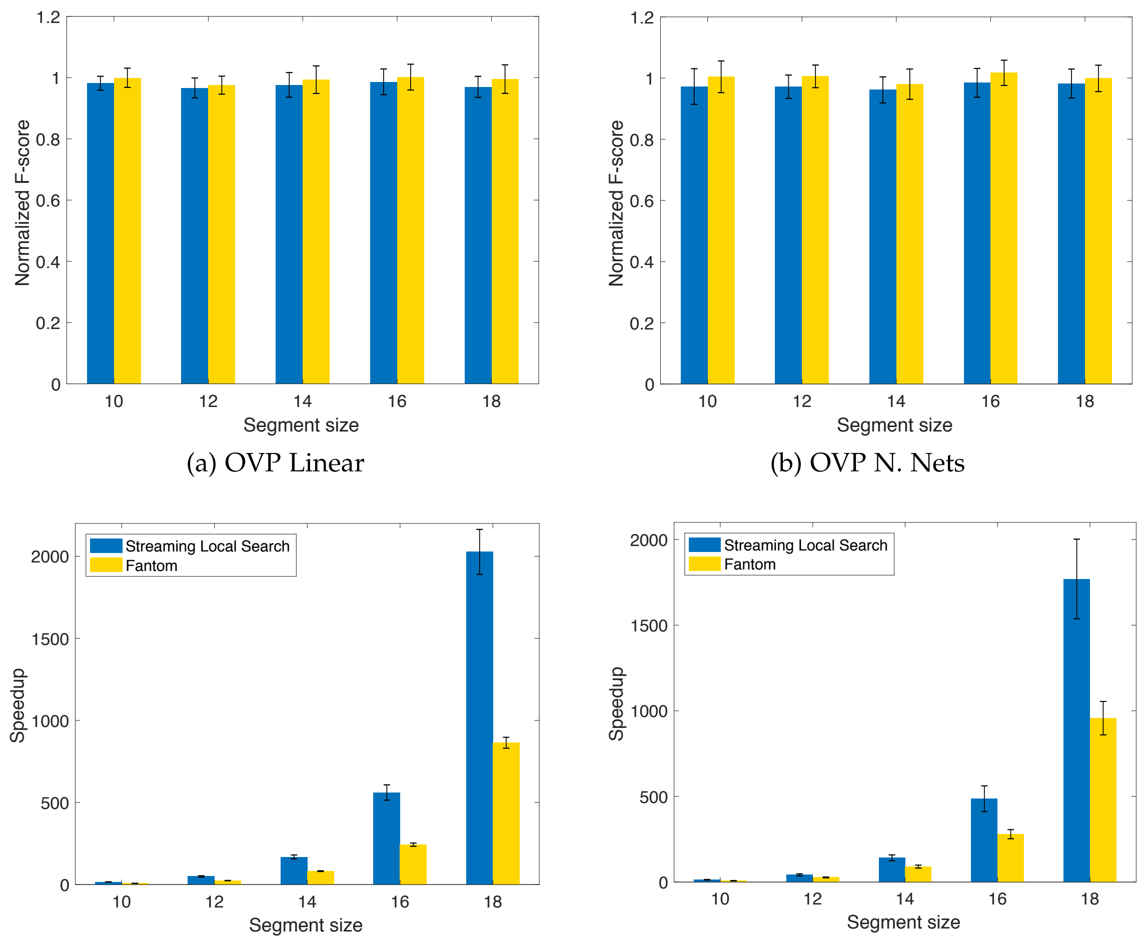


(d) YouTube N. Nets

Figure 9.1: *Performance of* STREAMING LOCAL SEARCH *compared to the other benchmarks. a) shows the ratio of the F-score obtained by Streaming Local Search and Fantom vs. the F-score obtained by the method of [Gon+14], using the sequential DPP objective and linear embeddings on YouTube dataset. b) shows the relative F-scores for non-linear features from a one-hidden-layer neural network. c), d) show the speed up of* STREAMING LOCAL SEARCH *and* FANTOM *over the method of [Gon+14].*

Figures 9.1b and 9.2b show similar results for nonlinear representations, where a one-hidden-layer neural network is used to infer a hidden representation for each frame. We make two observations: First, non-linear representations generally improve the solution quality. Second, as before, our streaming algorithm achieves exponential speed up (Figures 9.1d and 9.2d).

Finally, we also compared the three algorithms with a "standard", non-sequential DPP as the utility function, for generating summaries of length 5% of the video length. Again, our method yields competitive performance with a much shorter running time



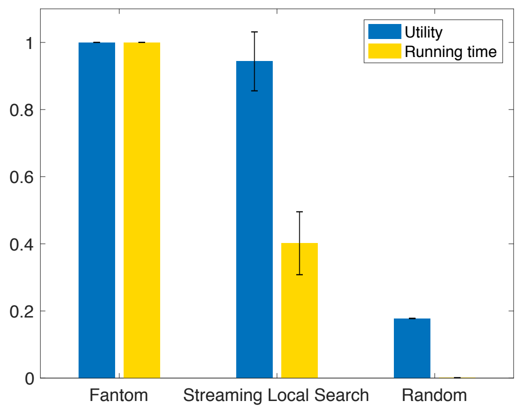
(a) OVP Linear (b) OVP N. Nets

(c) OVP Linear (d) OVP N. Nets

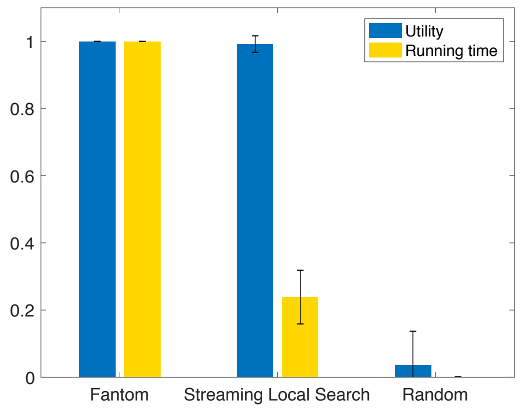
Figure 9.2: *Performance of* Streaming Local Search *compared to the other benchmarks. a) shows the ratio of the F-score obtained by* Streaming Local Search *and* Fantom *vs. the F-score obtained by the method of [Gon+14], using the sequential DPP objective and linear embeddings on OVP dataset. b) shows the relative F-scores for non-linear features from a one-hidden-layer neural network. c), d) show the speedup of Streaming Local Search and Fantom over the method of [Gon+14].*

(Figures 9.3c, 9.3a, 9.3d, 9.3b).

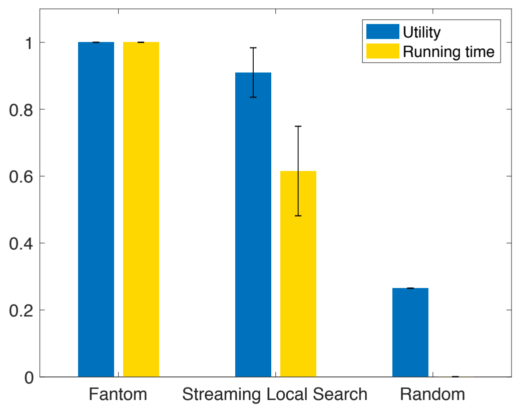
Using constraints to generate customized summaries. In our second experiment, we show how constraints can be applied to generate customized summaries. We apply Streaming Local Search to YouTube video 106, which is a part of America's Got Talent series. It features a singer and three judges in the judging panel. Here, we generated two sets of summaries using different constraints. The top row in Figure 9.4 shows a summary focused on the judges. Here we considered 3 uniform matroid



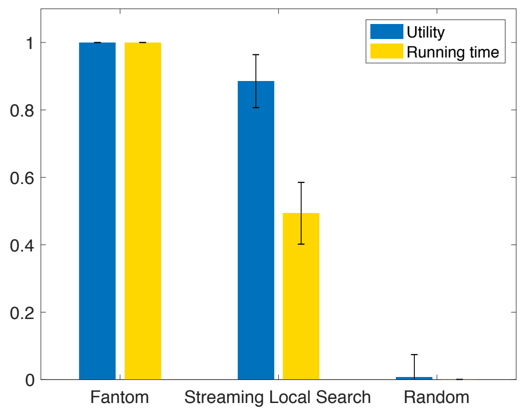
(a) YouTube Linear



(b) YouTube N. Nets



(c) OVP Linear



(d) OVP N. Nets

Figure 9.3: *Performance of* STREAMING LOCAL SEARCH *compared to the other benchmarks. a), c) show the utility and running time for* STREAMING LOCAL SEARCH *and random selection vs. the utility and running time of* FANTOM, *using the original DPP objective and linear embeddings on YouTube and OVP datasets. b), d) show similar qualities using non-linear features from a one-hidden-layer neural network.*

constraints to limit the number of frames chosen containing each of the judges, i.e., $\mathcal{I} = \{S \subseteq V : |S \cap V_j| \leq l_j\}$, where $V_j \subseteq V$ is the subset of the frames (not necessarily non-overlapping) including judge j, and $j \in [1,3]$. The limits l_j for all the matroid constraints are set to 3. To produce real-time summaries while receiving the video, we used the Viola-Jones algorithm [VJ04] to detect faces in each frame, and trained a multiclass support vector machine using histograms of oriented gradients (HOG) to recognize different faces. The bottom row in Figure 9.4 shows a summary focused on the singer using one matroid constraint.

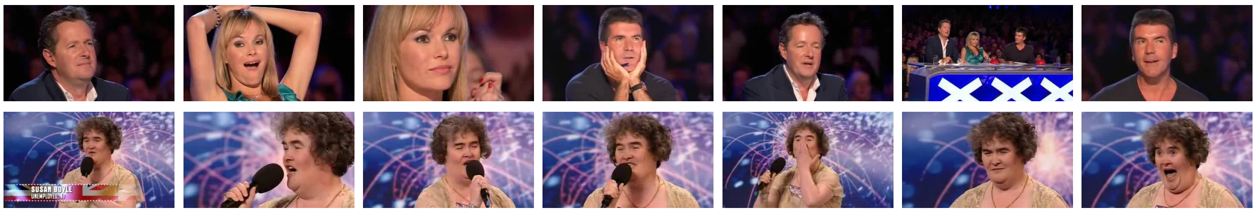


Figure 9.4: *Summary focused on judges, and singer for YouTube video 106.*

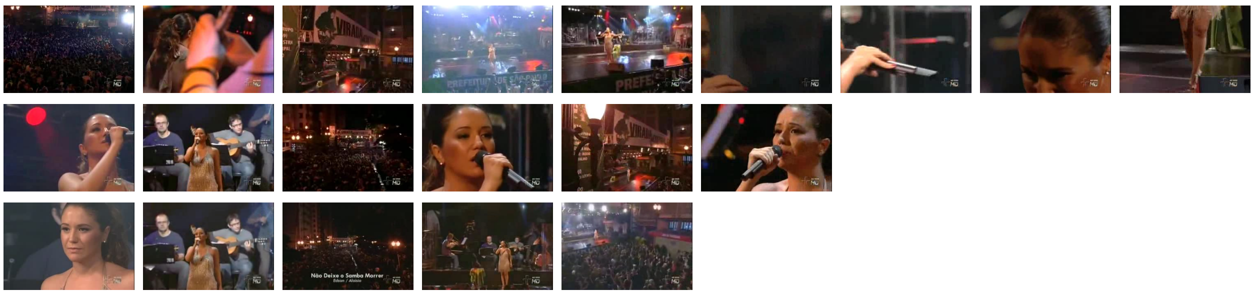


Figure 9.5: *Summary produced by method of [Gon+14] (top row), vs.* STREAMING LOCAL SEARCH *(middle row), and a user selected summary (bottom row), for YouTube video 105.*

To further enhance the quality of the summaries, we assigned different weights to the frames based on the probability for each frame to contain objects, using selective search [Uij+13]. Assigning a higher cost to the frames with a low probability of having objects, and having a knapsack constraint that limits the total cost of the elements of the solution, let us filter uninformative and blurry frames, and produce a summary closer to human-created summaries. Figure 9.5 compares the result obtained by our method and the method of [Gon+14] with a human-created summary.

9.5 Related Work

9.5.1 Video Summarization

Video summarization aims to retain diverse and representative frames according to criteria such as representativeness, diversity, interestingness, or importance of the frames [NMZ03; LK06; LGG12]. This often requires hand-crafting to combine the criteria effectively. Recently, [Gon+14] proposed a supervised subset selection method using DPPs. Despite its superior performance, this method uses an exhaustive search for MAP inference, which makes it inapplicable for producing real-time summaries.

9.5.2 Local Search

Local search has been widely used for submodular maximization subject to various constraints. This includes the analysis of greedy and local search by Nemhauser et al. [NWF78b] providing a $1/(p+1)$ approximation for monotone submodular maximization under p matroid constraints. Among the most recent results for non-monotone submodular maximization are a $(1+O(1/\sqrt{p}))p$-approximation subject to a p-system constraints [FHK17], a $1/5-\varepsilon$ approximation under d knapsack constraints [Lee+09], and a $(p+1)(2p+2d+1)/p$-approximation for maximizing a general submodular function subject to a p-system and d knapsack constraints [MZK16].

9.6 Summary

We have developed the first streaming algorithm, Streaming Local Search, for maximizing non-monotone submodular functions subject to a collection of independence systems and multiple knapsack constraints. In fact, our work provides a general framework for converting monotone streaming algorithms to non-monotone streaming algorithms for general constrained submodular maximization. We demonstrated its applicability to streaming video summarization with various personalization constraints. Our experimental results showed that our method is able to speed up the summarization task more than 1700 times, while achieving a similar performance to the centralized baselines. This makes it a promising approach for real-time summarization tasks. Indeed, our method applies to any summarization task with a non-monotone (nonnegative) submodular utility function, and a collection of independence systems and knapsack constraints. This includes but is not limited to the other concrete problem instances, with their corresponding non-monotone submodular objective functions, and various types of constraints that will be discussed in Chapter 13. Streaming Local Search facilitates real-time analytics and predictions on large streams of data, by efficiently extracting online summaries, in applications with a (non-monotone) submodular utility function and various constraints.

10 Deletion-Robust Submodular Maximization

As we saw in Chapter 9, streaming algorithms for submodular summarization not only avoids the need for vast amounts of random-access memory but also provides predictions in a timely manner based on the data seen so far, facilitating real-time analytics. While extracting useful information from big data in real-time promises many benefits, the development of more sophisticated methods for extracting, analyzing and using personal information has made privacy a major public issue. Various web services rely on the collection and combination of data about individuals from a wide variety of sources. At the same time, the ability to control the information an individual can reveal about herself in online applications has become a growing concern.

The *"right to be forgotten"* (with a specific mandate for protection in the European Data Protection [Reg12], and concrete regulation [EU15] released in 2014) allows individuals to claim the ownership of their personal information and gives them the authority to their online activities (videos, photos, tweets, etc). As an example, consider a road traffic information system that continuously monitors traffic speeds, travel times and incidents in real time. It combines and matches the massive amount of control messages available at the cellular network with GPS coordinates for each message while generating the area-wide traffic information service. However, some consumers, while using the service and providing some data, may not be willing to share information about specific locations due to privacy considerations (e.g., their place of residence). With the right to be forgotten, an individual can have certain data deleted from online database records so that third parties (e.g., search engines) can no longer trace them [Web11]. The data could be in the form of messages, pictures, events, and interests posted to an online social networking website (e.g., Facebook, Flickr, Google+, or Twitter). Similarly, it could be in the form of videos and images shared by a Google Glass while the user is traveling and exploring an area. Or finally, it could just be the behavioral patterns or feedback provided by a user through clicking on a piece of advertisement or news.

In this chapter, we propose the first framework that offers instantaneous data summarization while preserves the right of an individual to be forgotten. We cast this problem as an instance of *robust* streaming submodular maximization where the goal is to produce a concise real-time summary in the face of data deletion requested by users. We develop, Robust-Streaming, that transforms any streaming algorithm with provable guarantees and makes it robust against m deletions by instantiating m additional parallel, but crucially non-overlapping, solutions. More precisely, for a generic streaming algorithm StreamingAlg with γ approximation guarantee (discussed in Section 2.3.2, and are listed in Table 2.3), Robust-Streaming outputs a robust solution, against *any* m deletions from the summary at *any* given time, with a γ approximation guarantee to a computationally unrestricted, omniscient algorithm that knows in advance that such m data points will be deleted and hence will not select them at all. To the best of our knowledge, Robust-Streaming is the first algorithm with such strong theoretical guarantees and general applicability.

Our experimental results demonstrate the effectiveness of Robust-Streaming on several submodular maximization problems. We show that for active set selection on location data as well as click-stream log data, and interactive image collection summarization, Robust-Streaming leads to summaries with competitive utilities compared against those obtained via classical streaming methods that have the knowledge of which elements will be deleted later.

10.1 Deletion-Robust Model

So far in this Thesis, we discussed the *static* submodular data summarization, where we have a large but fixed dataset V of size n, and we are interested in finding a summary that best represents the data by maximizing a submodular utility function f subject to a family of constraints:

$$\mathsf{OPT} = \max\{f(A)|A \in \mathcal{I}\}, \tag{10.1.1}$$

where $\mathcal{I}$ is the family of feasible solutions. In this section, we formalize a novel *dynamic* variant, and constraints on time and memory that algorithms need to obey.

10.1.1 Dynamic Data: Additions and Deletions

In *dynamic* deletion-robust submodular maximization problem, the data V is generated at a fast pace and in real-time, such that at any point t in time, a subset $V_t \subseteq V$ of the data has arrived. Naturally, we assume that $V_1 \subseteq V_2 \subseteq \cdots \subseteq V_n$, with no assumption made on the *order* or the size of the datastream. Importantly, we allow data to be *deleted* dynamically as well. We use D_t to refer to data deleted by time t, where again $D_1 \subseteq D_2 \subseteq \cdots \subseteq D_n$. Without loss of generality, below we assume that at every time step t exactly one element $e_t \in V$ is either added or deleted, i.e., $|D_t \setminus D_{t-1}| + |V_t \setminus V_{t-1}| = 1$.

We now seek to solve a dynamic variant of Problem 10.1.1. More formally, at any time t, we are interested in the following dynamic but constrained submodular optimization problem:

$$\mathsf{OPT}_t = \max\{f(A_t)|A_t \in \mathcal{I}_t\} \tag{10.1.2}$$

where

$$\mathcal{I}_t = \{S : S \in \mathcal{I}, S \subseteq V_t \setminus D_t\}$$

is a potentially dynamic family of feasible solutions. We also denote by OPT_t the maximum utility achievable for Problem (10.1.2) at time t. Note that in general a feasible solution at time t might not be a feasible solution at a later instance t'. This is particularly important in practical situations where a subset of the elements D_t should be removed from the solution. We do not make any assumptions on the order or the size of the data stream V, but we assume that the *total number of deletions is limited to m*, i.e., $|D_n| \leq m$.

10.1.2 Dealing with Limited Time and Memory

In principle, we could solve Problem (10.1.2) by repeatedly – at every time t – solving a static submodular maximization problem by restricting the ground set V to $V_t \setminus D_t$. This is impractical even for moderate problem sizes. For large problems, we may not even be able to fit V_t into the main memory of the computing device (space constraints). Moreover, in real-time applications, one needs to make decisions in a timely manner while the data is continuously arriving (time constraints).

We hence focus on *streaming algorithms* which may maintain a limited memory $M_t \subset V_t \setminus D_t$, and must have an updated feasible solution $\{A_t \mid A_t \subseteq \mathcal{M}_t, A_t \in \mathcal{I}_t\}$ to output at any given time t. Ideally, the capacity of the memory $|\mathcal{M}_t|$ should not depend on t and V_t. As discussed in the previous chapter (*c.f.* Section 9.1), whenever a new element is received, the algorithm can choose 1) to insert it into its memory, provided that the memory does not exceed a pre-specified capacity bound, 2) to replace it with one or a subset of elements in the memory (In case of preemptive streaming algorithms), or otherwise 3) the element gets discarded and cannot be used later by the algorithm.

In addition, if the algorithm receives a deletion request for a subset $D_t \subset V_t$ at time t (in which case $\mathcal{I}_t$ will be updated to accommodate this request) it has to drop D_t from $\mathcal{M}_t$ in addition to updating A_t to make sure that the current solution is feasible (all subsets $A'_t \subset V_t$ that contain an element from D_t are infeasible, i.e., $A'_t \notin \mathcal{I}_t$). To account for such losses, the streaming algorithm can only use other elements maintained in its memory in order to produce a feasible candidate solution, i.e. $A_t \subseteq \mathcal{M}_t \subseteq ((V_t \setminus V_{t-1}) \cup \mathcal{M}_{t-1}) \setminus D_t$. We say that the streaming algorithm is *robust* against m deletions, if it can provide a feasible solution $A_t \in \mathcal{I}_t$ at any given time t such that $f(A_t) \geq \tau \mathsf{OPT}_t$ for some constant $\tau > 0$. Note that the streaming algorithm is competing against the optimal centralized method that has access to the full V and knows all the elements that will be removed by time t. In the following, we show how robust streaming algorithms can be obtained by carefully increasing the memory and running multiple instances of existing streaming methods simultaneously.

10.2 Example Applications

We now discuss three concrete applications, with their submodular objective functions f, where the size of the datasets and the nature of the problem often require a deletion-robust streaming solution.

10.2.1 Summarizing Click-stream and Geolocation Data

There exists a tremendous opportunity of harnessing prevalent activity logs and sensing resources. For instance, GPS traces of mobile phones can be used by road traffic information systems (such as Google traffic, TrafficSense, Navigon) to monitor travel times and incidents in real time. In another example, stream of user activity logs is recorded while users click on various parts of a webpage such as ads and news while browsing the web, or using social media. Continuously sharing all collected data is problematic for several reasons. First, memory and communication constraints may limit the amount of data that can be stored and transmitted across the network. Second, reasonable privacy concerns may prohibit continuous tracking of users.

In many such applications, the data can be described in terms of a kernel matrix K which encodes the similarity between different data elements. As discussed in Section 3.1.1, a small diverse subset (*active set*) of elements can be found by maximizing the following monotone submodular function, $f(S) = \log\det(I + \alpha K_{S,S})$ [KG13], where $\alpha > 0$ and $K_{S,S}$ is the principal sub-matrix of K indexed by the set S. In light of privacy concerns, it is natural to consider *participatory* models that empower users to decide what portion of their data could be made available. If a user decides not to share, or to revoke information about parts of their activity, the monitoring system should be able to update the summary to comply with users' preferences. Therefore, we use Robust-Streaming to identify a robust set of the k most informative data points.

10.2.2 Summarizing Image Collections

In the image collection summarization problem, the goal is to select a small subset of images that best represents different categories. For example, one may have a collection of images taken on a holiday trip, and want to select a small subset that concisely represents all the diversity from the trip. As discussed in Section 3.5, this problem can be addressed by maximizing the weighted linear combination of multiple submodular functions that capture different notions of representativeness, including *facility location*, *sum-coverage*, and *truncated graph cut*.

Now, consider a situation where a user want to summarize a large collection of her photos that she is taking while traveling. If she does not like some of the returned photos in the summary, we would like to be able to update the result without processing the whole collection from scratch. Robust-Streaming can be used as an appealing method of choice. Another scenario is when users upload their photos on social media such as Flicker where a summary of recent photos are shown. Now, if a user decides to remove some of her photos, we should be able to immediately update the summary. Again, Robust-Streaming comes to the rescue.

10.3 Robust-Streaming Algorithm

In this section, we first elaborate on why naively increasing the solution size does not help. Then, we present our main algorithm, Robust-Streaming, for deletion-robust streaming submodular maximization. Our approach builds on the following key ideas: 1) simultaneously constructing non-overlapping solutions, and 2) appropriately merging solutions upon deleting an element from the memory.

10.3.1 Increasing the Solution Size Does Not Help

One of the main challenges in designing streaming solutions is to immediately discover whether an element received from the data stream at time t is good enough to be added to the memory $\mathcal{M}_t$. This decision is usually made based on the added value or marginal gain of the new element which in turn depends on the previously chosen elements in the memory, i.e., $\mathcal{M}_{t-1}$. Now, let us consider the opposite scenario when an element e should be deleted from the memory at time t. Since now we have a smaller context, submodularity guarantees that the marginal gains of the elements added to the memory after e was added, could have only increased if e was not part of the stream (diminishing returns). Hence, if some elements had large marginal values to be included in the memory before the deletion, they still do after the deletion. Based on this intuition, a natural idea is to keep a solution of a bigger size, say $m + k$ (rather than k) for at most m deletions. However, this idea does not work as shown by the following example.

Bad Example (Coverage): Consider a collection of n subsets $V = \{B_1, \ldots, B_n\}$, where $B_i \subseteq \{1, \ldots, n\}$, and a coverage function $f(A) = |\cup_{i \in A} B_i|$, $A \subseteq V$. Suppose we receive $B_1 = \{1, \ldots, n\}$, and then $B_i = \{i\}$ for $2 \leq i \leq n$ from the stream. Streaming algorithms that select elements according to their marginal gain and are allowed to pick $k + m$ elements, will only pick up B_1 upon encounter (as other elements provide no gain), and return $A_n = \{B_1\}$ after processing the stream. Hence, if B_1 is deleted after the stream is received, these algorithms return the empty set $A_n = \emptyset$ (with $f(A_n) = 0$). An optimal algorithm which knows that element B_1 will be deleted, however, will return set $A_n = \{B_2, \ldots, B_{k+2}\}$, with value $f(A_n) = k + 1$. Hence, standard streaming algorithms fail arbitrarily badly even under a single deletion (i.e., $m = 1$), even when we allow them to pick sets larger than k.

In the following, we show how we can solve the above issue by carefully constructing not one but multiple solutions.

10.3.2 Building Multiple Solutions

As stated earlier, the existing one-pass streaming algorithms for submodular maximization work by identifying elements with marginal gains above a carefully chosen threshold. This ensures that any element received from the stream which is fairly similar to the elements of the solution set is discarded by the algorithm. Since elements are chosen as diverse as possible, the solution may suffer dramatically in case of a deletion.

One simple idea is to try to find m (near) duplicates for each element e in the memory, i.e., find e' such that $f(e') = f(e)$ and $\Delta(e'|e) = 0$ [OSU16]. This way if we face m deletions we can still find a good solution. The drawback is that even one duplicate may not exist in the data stream (see the bad example above), and we may not be able to recover for the deleted element. Instead, what we will do is to construct non-overlapping solutions such that once we experience a deletion, only one solution gets affected.

In order to be robust against m deletions, we take a generic streaming algorithm StreamingAlg with γ approximation guarantee. A list of streaming algorithms for submodular maximization with corresponding approximation guarantees is shown in Table 2.3. We then run a cascading chain of r instances of StreamingAlgs as follows. Let $\mathcal{M}_t = M_t^{(1)}, M_t^{(2)}, \ldots, M_t^{(r)}$ denote the content of their memories at time t. When we receive a new element $e \in V_t$ from the data stream at time t, we pass it to the first instance of StreamingAlg$^{(1)}$. If StreamingAlg$^{(1)}$ discards e, the

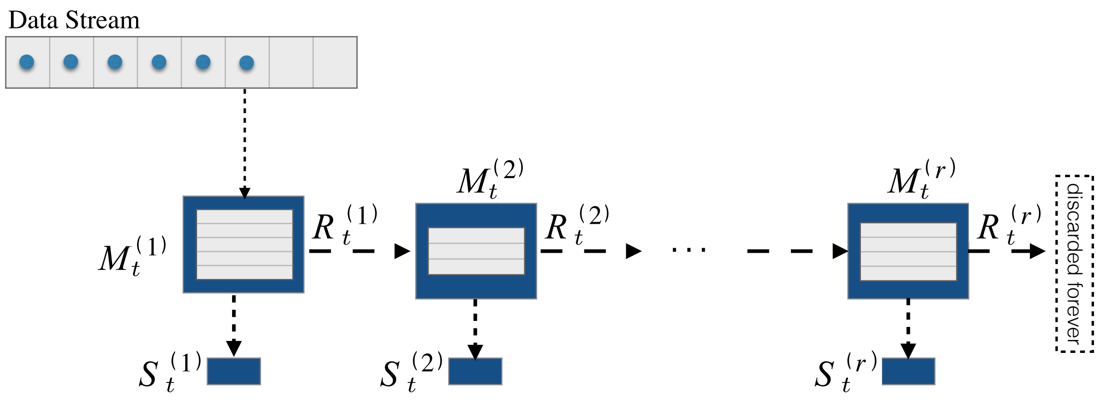


Figure 10.1: ROBUST-STREAMING *uses r instances of a generic* STREAMINGALG *to construct r non-overlapping memories at any given time t, i.e., $M_t^{(1)}, M_t^{(2)}, \ldots, M_t^{(r)}$. Each instance produces a solution $S_t^{(i)}$ and the solution returned by* ROBUST-STREAMING *is the first valid solution $S_t = \{S_t^{(i)} | i = \min j \in [1 \cdots r], M_t^{(i)} \neq null\}$.*

discarded element is cascaded in the chain and is passed to its successive algorithm, i.e. STREAMINGALG$^{(2)}$. If e is discarded by STREAMINGALG$^{(2)}$, the cascade continues and e is passed to STREAMINGALG$^{(3)}$. This process continues until either e is accepted by one of the instances or discarded for good. Now, let us consider the case where e is accepted by the i-th instance, SIEVE-STREAMING$^{(i)}$, in the chain. As discussed in Section 10.1.2, STREAMINGALG may choose to discard a set of points $R_t^{(i)} \subset M_t^{(i)}$ from its memory before inserting e, i.e., $\mathcal{M}_t^{(i)} \leftarrow \mathcal{M}_t^{(i)} \cup \{e\} \setminus R_t^{(i)}$. Note that $R_t^{(i)}$ is empty, if e is inserted and no element is discarded from $\mathcal{M}_t^{(i)}$. For every discarded element $r \in R_t^{(i)}$, we start a new cascade from $(i+1)$-th instance, STREAMINGALG$^{(i+1)}$.

Note that in the worst case, every element of the stream can go once through the whole chain during the execution of the algorithm, and thus the processing time for each element scales linearly by r. An important observation is that at any given time t, all the memories $M_t^{(1)}, M_t^{(2)}, \ldots, M_t^{(r)}$ contain disjoint sets of elements. In the next section, we show how this data structure leads to a deletion-robust streaming algorithm.

10.3.3 Dealing with Deletions

Equipped with the above data structure shown in Figure 10.1, we now demonstrate how deletions can be treated. Assume an element e_d is being deleted from the memory of the j-th instance of STREAMINGALG$^{(j)}$ at time t, i.e., $M_t^{(j)} \leftarrow M_t^{(j)} \setminus \{e_d\}$. As discussed in Section 10.3.1, the solution of the streaming algorithm can suffer dramatically from a deletion, and we may not be able to restore the quality of the solution by substituting similar elements. Since there is no guarantee for the quality of the solution after a deletion, we remove StreamingAlg $^{(j)}$ from the chain by making $R_t^{(i)}=$null and for all the remaining elements in its memory $M_t^{(j)}$, namely, $R_t^{(j)} \leftarrow M_t^{(j)} \setminus \{e_d\}$, we start a new cascade from $j+1$-th instance, StreamingAlg$^{(j+1)}$.

The key reason why the above algorithm works is that the guarantee provided by the streaming algorithm is independent of the order of receiving the data elements. Note that at any point in time, the first instance i of the algorithm with $M_t^{(i)} \neq$ null has processed all the elements from the stream V_t (not necessarily in the order the stream is originally received) except the ones deleted by time t, i.e., D_t. Therefore, we can guarantee that StreamingAlg $^{(i)}$ provides us with its inherent γ-approximation guarantee for reading $V_t \setminus D_t$. More precisely, $f(S_t^{(i)}) \geq \alpha \mathsf{OPT}_t$, where OPT_t is the optimum solution for the constrained optimization problem (10.1.2) at time t, when we have at most m deletions.

In case of adversary deletions, there will be one deletion from the solution of m instances of StreamingAlg in the chain. Therefore, having $r = m+1$ instances, we will remain with only one StreamingAlg that gives us the desired result. However, as shown later in this section, if the deletions are i.i.d. (which is often the case in practice), and we have m deletions in expectation, we need r to be much smaller than $m+1$. Finally, note that we do not need to assume that $m \leq k$ where k is the size of the largest feasible solution.The above idea works for arbitrary $m \leq n$.

The pseudocode of Robust-Streaming is given in Algorithm 11. It uses $r \leq m+1$ instances of StreamingAlg as subroutines in order to produce r solutions. We denote by $S_t^{(1)}, S_t^{(1)}, \ldots, S_t^{(r)}$ the solutions of the r StreamingAlgs at any given time t. We assume that an instance i of StreamingAlg$^{(i)}$ receive an input element and produces a solution $S_t^{(i)}$ based on the input. It may also change its memory content $M_t^{(i)}$, and discard a set $R_t^{(i)}$. Among all the remained solutions (i.e., the ones that are not "null"), it returns the first solution in the chain, i.e. the one with the lowest index.

Theorem 27. *Let* StreamingAlg *be a 1-pass streaming algorithm that achieves an γ-approximation guarantee for the constrained maximization problem* (10.1.2) *with an update time of T, and a memory of size M when there is no deletion. Then* Robust-Streaming *uses $r \leq m+1$ instances of* StreamingAlgs *to produce a feasible solution $S_t \in \mathcal{I}_t$ (now $\mathcal{I}_t$ encodes deletions in addition to constraints) such that $f(S_t) = \gamma OPT_t$ as long as no more*

Algorithm 11: ROBUST-STREAMING

```
Input: data stream $V_t$, deletion set $D_t$, $r \leq m+1$.
Output: solution $S_t$ at any time $t$.
1: $t = 1$, $M_t^{(i)} = 0$, $S_t^{(i)} = \emptyset$                $\forall i \in [1 \cdots r]$
2: while ($\{V_t \setminus V_{t-1}\} \cup \{D_t \setminus D_{t-1}\} \neq \emptyset$) do
3:    if $\{D_t \setminus D_{t-1}\} \neq \emptyset$ then
4:       $e_d \leftarrow \{D_t \setminus D_{t-1}\}$
5:       Delete($e_d$)
6:    else
7:       $e_t \leftarrow \{V_t \setminus V_{t-1}\}$
8:       Add(1, $e_t$)
9:    end if
10:   $t = t + 1$
11:   $S_t = \{S_t^{(i)} \mid i = \min\{j \in [1 \cdots r],\ M_t^{(j)} \neq \text{null}\}\}$
12: end while
```

```
13: function Add($i, R$)
14:    for $e \in R$ do
15:       $[R_t^{(i)}, M_t^{(i)}, S_t^{(i)}]$ = STREAMINGALG$^{(i)}(e)$
16:       if $R_t^{(i)} \neq \emptyset$ and $i < r$ then
17:          Add($i + 1$, $R_t^{(i)}$)
18:       end if
19:    end for
20: end function
```

```
21: function Delete($e$)
22:    for $i = 1$ to $r$ do
23:       if $e \in M_t^{(i)}$ then
24:          $R_t^{(i)} = M_t^{(i)} \setminus \{e\}$
25:          $M_t^{(i)} \leftarrow$ null
26:          Add($i + 1, R_t^{(i)}$)
27:          return
28:       end if
29:    end for
30: end function
```

than m elements are deleted from the data stream. Moreover, ROBUST-STREAMING *uses a memory of size rM, and has worst case update time of $O(r^2MT)$, and average update time of $O(rT)$.*

Algorithm	Problem	Constraint	Apprx.	Memory
Robust + Sieve-Streaming [Bad+14]	Mon. Subm.	Cardinality	$1/2-\epsilon$	$O(mk\log k/\epsilon)$
Robust + f-MSM [CK14]	Mon. Subm.	p-matroids	$\frac{1}{4p}$	$O(mk(\log\lvert V\rvert)^{O(1)})$
Robust + Streaming-Greedy [CGQ15]	Non-mon. Subm.	p-matchoid	$\frac{(1-\epsilon)(2-o(1))}{(8+e)p}$	$O(mk\log k/\epsilon^2)$
Robust + Streaming-Local Search [Chapter 9]	Non-mon. Subm.	ind. systems + d knapsacks	$\frac{(1-\epsilon)}{1+\frac{2}{\sqrt{\alpha}}+\frac{1}{\alpha}+2d(1+\sqrt{\alpha})}$	$O(mM\log(k)/\epsilon\sqrt{\alpha})$

Table 10.1: Robust-Streaming *can be combined with the existing one-pass streaming algorithms in order to make them robust against m deletions.* α, M *is the approximation guarantee, and memory for streaming monotone submodular maximization under a collection of independence systems and d knapsack constraints.*

The proofs of the theorems in this chapter can be found in Appendix A.5. In Table 10.1 we combine the result of Theorem 27 with the existing streaming algorithms that satisfy our requirements.

Theorem 28. *Assume each element of the stream is deleted with equal probability* $p = m/n$*, i.e., in expectation we have m deletions from the stream. Then, with probability* $1-\delta$*,* Robust-Streaming *provides an* γ*-approximation as long as*

$$r \geq \left(\frac{1}{1-p}\right)^k \log\left(1/\delta\right).$$

Theorem 28 shows that for fixed k, δ and p, a *constant* number r of StreamingAlgs is sufficient to support $m = pn$ (expected) deletions *independently* of n. In contrast, for adversarial deletions, as analyzed in Theorem 27, $pn+1$ copies of StreamingAlg are required, which grows linearly in n. Hence, the required dependence of r on m is much milder for random than adversarial deletions. This is also verified by our experiments in Section 10.4.

10.4 Experiments

We address the following questions: 1) How much can Robust-Streaming recover and possibly improve the performance of StreamingAlg in case of deletions? 2) How much does the time of deletions affect the performance? 3) To what extent does deleting representative vs. random data points affect the performance? To this end, we run Robust-Streaming on the applications we described in Section 10.2, namely, image collection summarization, summarizing stream of geolocation sensor data, as well as summarizing a clickstream of size 45 million.

Throughout this section we consider the following streaming algorithms introduced in Section 2.3.2: Sieve-Streaming [Bad+14], Stream-Greedy [GK10], and Streaming-Greedy [CGQ15]. We allow all streaming algorithms, including the non-preemptive Sieve-Streaming, to update their solution after each deletion. We also consider a stronger variant of Sieve-Streaming, called ExtSieve, that aims to pick $k \cdot (m+1)$ elements to protect for deletions, i.e., is allowed the same memory as Robust-Streaming. After the deletions, the remaining solution is pruned to k elements.

To compare the effect of deleting representative elements to that of deleting random elements from the stream, we use two stochastic variants of the greedy algorithm, namely, Stochastic-Greedy [Chapter 12] and Random-Greedy [Buc+14]. This way we introduce randomness into the deletion process in a principled way. Hence, we have:

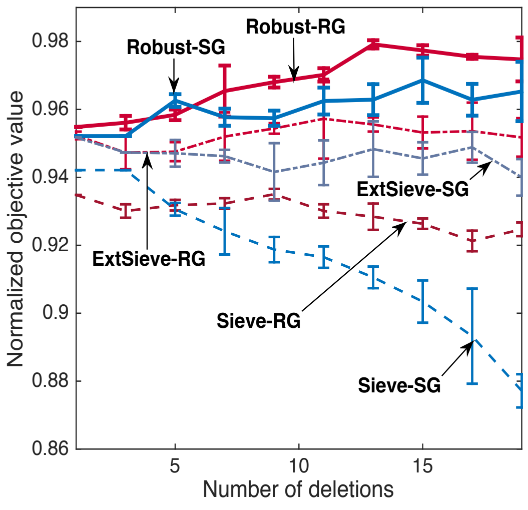
Stochastic-Greedy (SG): Similar to the the greedy algorithm, Stochastic-Greedy starts with an empty set and adds one element at each iteration until obtains a solution of size m. But in each step it first samples a random set R of size $(n/m)\log(1/\epsilon)$ and then adds an element from R to the solution which maximizes the marginal gain. We will discuss Stochastic-Greedy later in details, in Chapter 12.

Random-Greedy (RG): This algorithm iteratively selects a random element from the top m elements with the highest marginal gains, until finds a solution of size m.

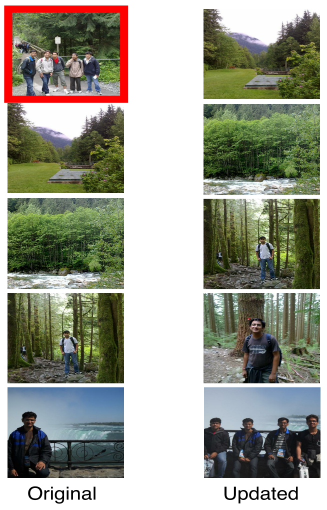
For each deletion method, the m data points are deleted either while receiving the data (where the steaming algorithms have the chance to update their solutions by selecting new elements) or after receiving the data (where there is no chance of updating the solution with new elements). Finally, *the performance of all algorithms are normalized against the utility obtained by the centralized algorithm that knows the set of deleted elements in advance.*

10.4.1 Image Collection Summarization

We first apply Robust-Streaming to a collection of 100 images from [Tsc+14]. We used the weighted combination of 594 submodular functions either capturing coverage or rewarding diversity (c.f. Section 10.2.2). Here, despite the small size of the dataset, computing the weighted combination of 594 functions makes the function evaluation considerably expensive.



(a) Images



(b) Images

Figure 10.2: *Performance of* Robust-Streaming *vs* Sieve-Streaming *for different deletion strategies (SG, RG) on a collection of 100 images. Here we fix $k = 5$ and $r = 3$. a) performance of* Robust-Streaming *and* Sieve-Streaming *normalized by the utility obtained by greedy that knows the deleted elements beforehand. b) updated solution of size $k = 5$ returned by* Robust-Streaming *after deleting the 1 image from the summary.*

Figure 10.2a compares the performance of Sieve-Streaming with its robust version Robust-Streaming for $r = 3$ and solution size k=5. Here, we vary the number m of deletions from 1 to 20 after the whole stream is received. We see that Robust-Streaming maintains its performance by updating the solution after deleting subsets of data points imposed by different deletion strategies. It can be seen that, even for a larger number m of deletions, Robust-Streaming, run with parameter $r < m$, is able to return a solution competitive with the strong centralized benchmark that knows the deleted elements beforehand. For the image collection, we were not able to compare the performance of Stream-Greedy with its robust version due to the prohibitive running time. Figure 10.2b shows an example of an updated image summary returned by Robust-Streaming after deleting the first image from the summary.

10.4.2 Summarizing a stream of geolocation data

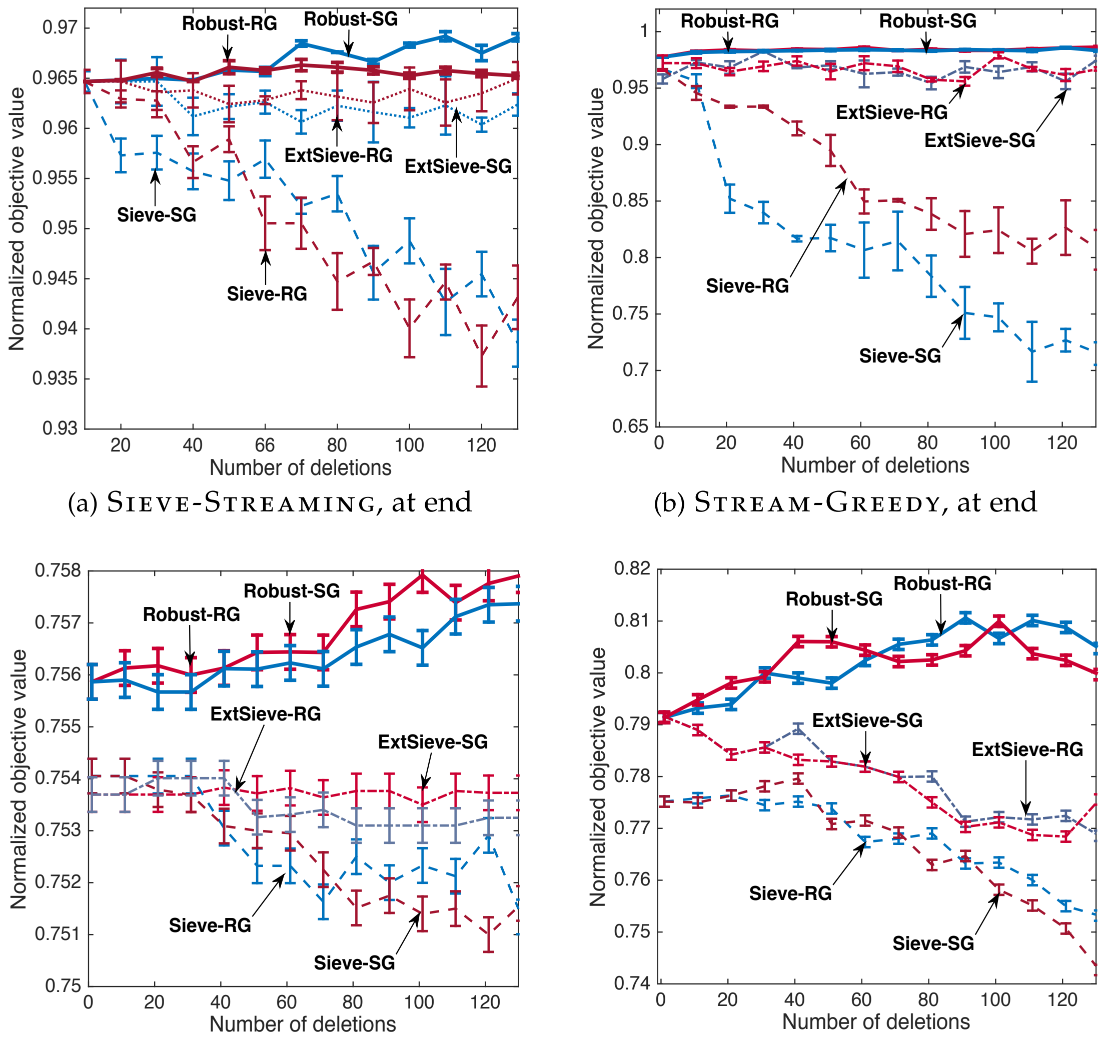
Next we apply Robust-Streaming to the active set selection objective described in Section 10.2.1. Our dataset consists of 3,607 geolocations, collected during a one hour bike ride around Zurich [Fat15]. For each pair of points i and j we used the corresponding (latitude, longitude) coordinates to calculate their distance in meters $d_{i,j}$ and chose a Gaussian kernel $\mathcal{K}_{i,j} = \exp(-d_{i,j}^2/h^2)$ with $h = 1500$.

Figure 10.4a shows the dataset where red and green triangles show a summary of size 10 found by Sieve-Streaming, and the updated summary provided by Robust-Streaming with $r = 5$ after deleting $m = 70\%$ of the datapoints. Figures 10.3a and 10.3c compare the performance of Sieve-Streaming with its robust version when the data is deleted after or during the stream, respectively. As we see, Robust-Streaming provides a solution very close to the hindsight centralized method. Figures 10.3b and 10.3d show similar behavior for Stream-Greedy. Note that deleting data points via Stochastic-Greedy or Random-Greedy are much more harmful on the quality of the solution provided by Stream-Greedy. We repeated the same experiment by dividing the map into grids of length 2km. We then considered a partition matroid by restricting the number of points selected from each grid to be 1. The red and green triangles in Figure 10.4b are the summary found by Streaming-Greedy and the updated summary provided by Robust-Streaming after deleting the shaded area in the figure.

10.4.3 Large scale click through prediction

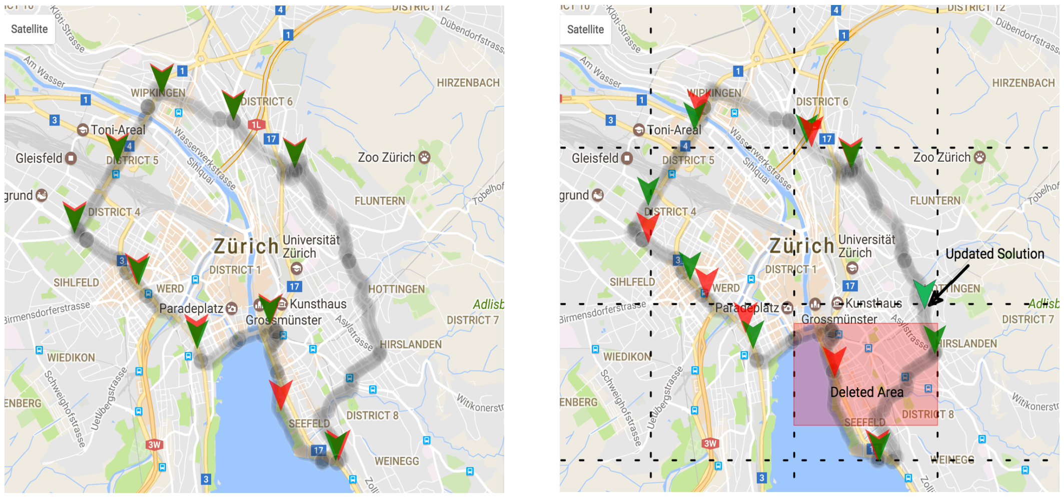
For our large-scale experiment we consider again the active set selection objective, described in Section 10.2.1. We used *Yahoo! Webscope* data set containing 45,811,883 user click logs for news articles displayed in the Featured Tab of the Today Module on Yahoo! Front Page during the first ten days in May 2009 [Yah12]. For each visit, both the user and shown articles are associated with a feature vector of dimension 6. We take their outer product, resulting in a feature vector of size 36.

The goal was to predict the user behavior for each displayed article based on historical clicks. To do so, we considered the first 80% of the data (for the fist 8 days) as our



(a) Sieve-Streaming, at end (b) Stream-Greedy, at end (c) Sieve-Streaming, during (d) Stream-Greedy, during

Figure 10.3: Robust-Streaming *vs* Sieve-Streaming *and* Stream-Greedy *for different deletion strategies (SG, RG) on geolocation data. We fix $k = 20$ and $r = 5$. a) and c) show the performance of robustified* Sieve-Streaming*, whereas b) and d) show performance for robustified* Stream-Greedy*. a) and b) consider the performance after deletions at the end of the stream, while c) and d) consider average performance while deletions happen during the stream.*



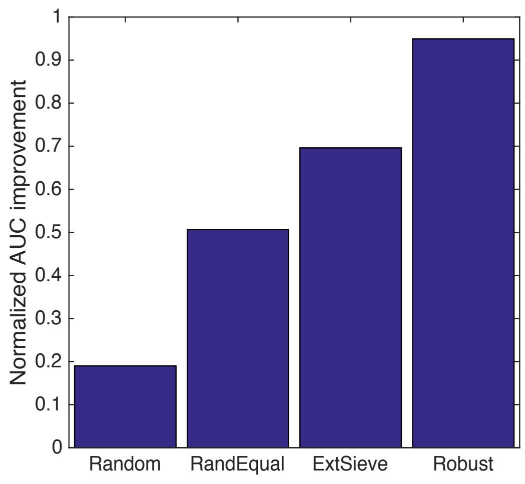
(a) Cardinality constraints (b) Matroid constraints

Figure 10.4: Robust-Streaming *vs* Sieve-Streaming *and* Stream-Greedy *for different deletion strategies (SG, RG) on geolocation data. We fix* $k = 20$ *and* $r = 5$. *a) red and green triangles show a set of size 10 found by* Sieve-Streaming *and the updated solution found by* Robust-Streaming *where 70% of the points are deleted. b) set found by* Streaming-Greedy, *constrained to pick at most 1 point per grid cell (matroid constraint). Here* $r = 5$, *and we deleted the shaded area.*

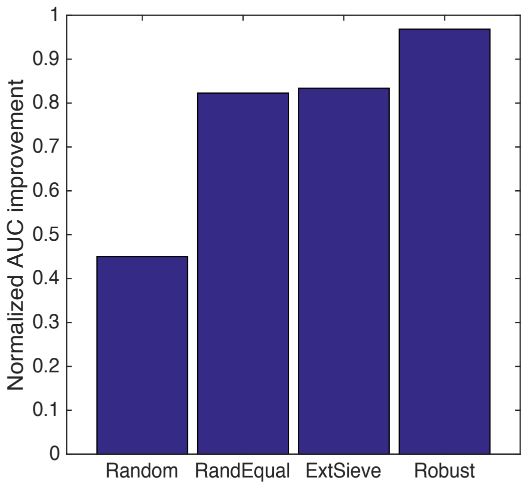
training set, and the last 20% (for the last 2 days) as our test set. We used Vowpal-Wabbit [LLS07] to train a linear classifier on the full training set. Since only 4% of the data points are clicked, we assign a weight of 10 to each clicked vector. The AUC score of the trained classifier on the test set was 65%.

We then used Robust-Streaming and Sieve-Streaming to find a representative subset of size k consisting of $k/2$ clicked and $k/2$ not-clicked examples from the training data. Due to the massive size of the dataset, we used Spark on a cluster of 15 quad-core machines with 32GB of memory each. We partitioned the training data to the machines keeping its original order. We ran Robust-Streaming on each machine to find a summary of size $k/15$, and merged the results to obtain the final summary of size k. We then start deleting the data uniformly at random until we left with only 1% of the data, and trained another classifier on the remaining elements from the summary.

Figure 10.5a compares the performance of Robust-Streaming for a fixed active set of size $k = 10,000$, and $r = 2$ with random selection, randomly selecting equal numbers of clicked and not-clicked vectors, and using Sieve-Streaming for selecting equal numbers of clicked and not-clicked data points. The y-axis shows the improvement in



(a) Yahoo! Webscope, $r = 2$



(b) Yahoo! Webscope, $r = 5$

Figure 10.5: ROBUST-STREAMING *vs random unbalanced and balanced selection and* SIEVE-STREAMING *selecting equal numbers of clicked and not-clicked data points, on 45,811,883 feature vectors from* Yahoo! Webscope *data. We fix* $k = 10,000$ *and delete 99% of the data points.*

AUC score of the classifier trained on a summary obtained by different algorithms over random guessing (AUC=0.5), normalized by the AUC score of the classifier trained on the whole training data. To maximize fairness, we let other baselines select a subset of $r.k$ elements before deletions. Figure 10.5b shows the same quantity for $r = 5$. It can be seen that a slight increase in the amount of memory helps boosting the performance for all the algorithms. However, ROBUST-STREAMING benefits from the additional memory the most, and can almost recover the performance of the classifier trained on the *full training data, even after 99% deletion*.

10.5 Summary

We have developed the first deletion-robust streaming algorithm–ROBUST-STREAMING – for constrained submodular maximization. Given any single-pass streaming algorithm STREAMINGALG with α-approximation guarantee, ROBUST-STREAMING uses $r \leq m+1$ instances of STREAMINGALG to output a solution that is robust against m deletions. The returned solution also satisfies an α-approximation guarantee w.r.t. to the solution of the optimum centralized algorithm that knows the set of m deletions in advance. We have also demonstrated the effectiveness of our approach through extensive set of experiments. As shown in Section 10.4, ROBUST-STREAMING can immediately update the solution in case of deletions, and significantly improves the performance of the existing streaming approaches. This property of ROBUST-STREAMING makes it an appealing approach for solving very large scale applications on dynamic datasets that experience deletions very often.

Part IV

Fast Centralized Algorithms

11 Overview of part IV

In Part II and III of this Thesis, we studied distributed and streaming methods for submodular miaximization. A natural complementary goal to the aforementioned methods for scaling up submodular maximization techniques, is to develop faster centralized algorithms for submodular maximization. Such methods can be easily integrated into the existing distributed methods, or in approaches that decompose the submodular function into simpler functions for faster evaluation. In part IV of this Thesis, we will first present a stochastic method for monotone submodular maximization under cardinality constraint. We then discuss a fast algorithm for maximizing a (non-monotone) submodular miaximization under a p-system and d knapsack constraints.

Lazier than Lazy Greedy. Is it possible to maximize a monotone submodular function faster than the widely used lazy greedy algorithm (also known as accelerated greedy), both in theory and practice? In Chapter 12, we develop the first linear-time algorithm for maximizing a general monotone submodular function subject to a cardinality constraint. We show that our randomized algorithm, Stochastic-Greedy, can achieve a $(1 - 1/e - \varepsilon)$ approximation guarantee, in expectation, to the optimum solution in time *linear* in the size of the data and *independent* of the cardinality constraint. We empirically demonstrate the effectiveness of our algorithm on submodular functions arising in data summarization, including training large-scale kernel methods and exemplar-based clustering. We observe that Stochastic-Greedy practically achieves the same utility value as lazy greedy but runs much faster. More surprisingly, we observe that in many practical scenarios Stochastic-Greedy does not evaluate the whole fraction of data points even once and still achieves indistinguishable results compared to lazy greedy.

Fast Constrained Submodular Maximization. Can we summarize multi-category data based on user preferences in a scalable manner? We cast personalized data summarization as an instance of a general submodular maximization problem subject to multiple constraints. In Chapter 13, we develop the first practical and FAst coNsTrained submOdular Maximization algorithm, Fantom, with strong theoretical guarantees. Fantom maximizes a submodular function (*not necessarily monotone*) subject to the intersection of a p-system and d knapsack constrains. It achieves a $(1+\epsilon)(p+1)(2p+2d+1)/p$ approximation guarantee with only $O(\frac{nrp\log(n)}{\epsilon})$ query complexity (n and r indicate the size of the ground set and the size of the largest feasible solution, respectively). We then show how we can use Fantom for personalized data summarization. In particular, a p-system can model different aspects of data, such as categories or time stamps, from which the users choose. In addition, knapsacks encode users' constraints including budget or time. In our set of experiments, we consider several concrete applications: movie recommendation, personalized image summarization, and revenue maximization. We observe that Fantom constantly provides the highest utility against all the baselines.

Summary of contributions. The key contributions of this part of the Thesis are:

1. We consider fast centralized algorithms for submodular maximization. We develop novel, efficient approximation algorithms:
   - Stochastic-Greedy, a linear-time algorithm for maximizing a general monotone submodular function subject to a cardinality constraint, and
   - Fantom, a fast algorithm for maximizing a (*not necessarily monotone*) submodular function subject to the intersection of a p-system and d knapsacks constrains.
2. We theoretically analyze our approaches, and provide approximation guarantees for the quality of the solutions.

3. We demonstrate the performance our algorithms on several real-world problems, including
    - selecting placements of size 200 from a set of 12,527 possible locations for sensor placement,
    - selecting active sets of size 200 from Parkinsons Telemonitoring dataset consisting of 5,875 biomedical voice measurements,
    - selecting summaries of size 200 from 50K Tiny Images,
    - movie recommendation over 11K movies from the MovieLens database,
    - personalized image summarization with 10K images, and
    - revenue maximization on the YouTube social networks with 5000 communities.

12 Lazier than Lazy Greedy

Previously, we have seen that in many machine learning and data mining applications, a summary of manageable size can be obtained by maximizing a submodular set function under a cardinality constraint. In this chapter, we propose the first linear-time algorithm, Stochastic-Greedy, for maximizing a non-negative monotone submodular function subject to a cardinality constraint k. We show that Stochastic-Greedy achieves a $(1 - 1/e - \epsilon)$ approximation guarantee to the optimum solution with running time $O(n \log(1/\epsilon))$ (measured in terms of function evaluations) that is *independent* of k. We start by a brief review of the classical greedy algorithm. We then discuss an accelerated version of the greedy algorithm Lazy-Greedy, and then introduce our fast randomized method, Stochastic-Greedy.

12.1 Greedy Algorithm

As we discussed in Chapter 2, the greedy algorithm is the simplest and the most efficient for maximizing a monotone submodular function under cardinality constraint. Here, the optimization problem is to find a subset A^* of size at most k that maximizes the monotone submodular utility function f, i.e.,

$$A^* = \arg\max_{A:|A|\leq k} f(A), \tag{12.1.1}$$

We saw in Section 2.2.1 that for non-negative monotone submodular functions, the greedy algorithm starts with the empty set A_0 and in iteration i, adds an element maximizing the marginal gain $\Delta(e|A_{i-1})$, where $\Delta(i|A) \doteq f(A \cup \{i\}) - f(A)$ measures the marginal *gain* of adding a new element i to a summary A. For a ground set V of size n, this greedy algorithm needs $O(n \cdot k)$ function evaluations in order to find a summarization of size k. However, in many data intensive applications, evaluating f is expensive and running the standard greedy algorithm is infeasible.

12.1.1 Lazy-Greedy

Submodularity can be exploited to implement an accelerated version of the classical greedy algorithm, usually called Lazy-Greedy [Min78]. Instead of computing $\Delta(e|A_{i-1})$ for each element $e \in V$, the Lazy-Greedy algorithm keeps an upper bound $\rho(e)$ (initially ∞) on the marginal gain sorted in decreasing order. In each iteration i, the Lazy-Greedy algorithm evaluates the element on top of the list, say e, and updates its upper bound, $\rho(e) \leftarrow \Delta(e|A_{i-1})$. If after the update $\rho(e) \geq \rho(e')$ for all $e' \neq e$, submodularity guarantees that e is the element with the largest marginal gain. Even though the exact cost (i.e., number of function evaluations) of Lazy-Greedy is unknown, this algorithm leads to orders of magnitude speedups in practice. As a result, it has been used as the state-of-the-art implementation in numerous applications including network monitoring [Les+07], network inference [RLK12], document summarization [LB11a], and speech data subset selection [Wei+13], to name a few. However, as the size of the data increases, even for small values of k, running Lazy-Greedy is infeasible. A natural question to ask is whether it is possible to further accelerate Lazy-Greedy by a procedure with a weaker dependency on k. Or even better, is it possible to have an algorithm that does not depend on k at all and scales linearly with the data size n?

In this chapter, we develop the first centralized algorithm whose cost (i.e., number of function evaluations) is independent of the cardinality constraint, which in turn directly addresses the shortcoming of Lazy-Greedy. Our experimental results on exemplar-based clustering and active set selection in nonparametric learning also confirms that Stochastic-Greedy consistently outperforms Lazy-Greedy by a large margin while achieving practically the same utility value. More surprisingly, in our experiments we observe that Stochastic-Greedy sometimes does not even evaluate all the items and shows a running time that is less than n while still providing solutions close to the ones returned by LAZY-GREEDY. Due to its independence of k, STOCHASTIC-GREEDY is the first algorithm that truly scales to voluminous datasets.

12.2 Stochastic-Greedy Algorithm

In this section, we present our randomized greedy algorithm STOCHASTIC-GREEDY and then show how to combine it with lazy evaluations. We will show that STOCHASTIC-GREEDY has provably linear running time independent of k, while simultaneously having the same approximation ratio guarantee (in expectation). In the following section we will further demonstrate through experiments that this is also reflected in practice, i.e., STOCHASTIC-GREEDY is substantially faster than LAZY-GREEDY, while being practically identical to it in terms of the utility.

The main idea behind STOCHASTIC-GREEDY is to produce an element which improves the value of the solution roughly the same as greedy, but in a fast manner. This is achieved by a sub-sampling step. At a very high level this is similar to how stochastic gradient descent improves the running time of gradient descent for convex optimization.

12.2.1 Random Sampling

The key reason that the classic greedy algorithm works is that at each iteration i, an element is identified that reduces the gap to the optimal solution by a significant amount, i.e., by at least $(f(A^*) - f(A_{i-1}))/k$. This requires n oracle calls per step, the main bottleneck of the classic greedy algorithm. Our main observation here is that by submodularity, we can achieve the same improvement by adding a uniformly random element from A^* to our current set A. To get this improvement, we will see that it is enough to randomly sample a set R of size $(n/k)\log(1/\epsilon)$, which in turn overlaps with A^* with probability $1-\epsilon$. This is the main reason we are able to achieve a boost in performance.

The algorithm is formally presented in Algorithm 12. Similar to the greedy algorithm, our algorithm starts with an empty set and adds one element at each iteration. But in each step it first samples a set R of size $(n/k)\log(1/\epsilon)$ uniformly at random and then adds the element from R to A which increases its value the most.

Algorithm 12: STOCHASTIC-GREEDY

```
input $f : 2^V \rightarrow \mathbb{R}_+$, $k \in \{1, \ldots, n\}$.
output A set $A \subseteq V$ satisfying $|A| \leq k$.
1: $A \leftarrow \emptyset$.
2: for ($i \leftarrow 1$; $i \leq k$; $i \leftarrow i + 1$) do
3:     $R \leftarrow$ a random subset obtained by sampling $s$ random elements from $V \setminus A$.
4:     $a_i \leftarrow \arg\max_{a \in R} \Delta(a|A)$.
5:     $A \leftarrow A \cup \{a_i\}$
6: end for
Return: $A$.
```

Our main theoretical result is the following. It shows that STOCHASTIC-GREEDY achieves a near-optimal solution for general monotone submodular functions, with computational complexity independent of the cardinality constraint.

Theorem 29. *Let f be a non-negative monotone submoduar function. Let us also set $s = \frac{n}{k} \log \frac{1}{\epsilon}$. Then* STOCHASTIC-GREEDY *achieves a $(1 - 1/e - \epsilon)$ approximation guarantee in expectation to the optimum solution of problem (12.1.1) with only $O(n \log \frac{1}{\epsilon})$ function evaluations.*

The proof can be found in Appendix A.6. Since there are k iterations in total and at each iteration we have $(n/k) \log(1/\epsilon)$ elements, the total number of function evaluations cannot be more than $k \times (n/k) \log(1/\epsilon) = n \log(1/\epsilon)$. The proof of the approximation guarantee is given in the analysis section.

12.2.2 Random Sampling with Lazy Evaluation

While our theoretical results show a provably linear time algorithm, we can combine the random sampling procedure with lazy evaluation to boost its performance. There are mainly two reasons why lazy evaluation helps. First, the randomly sampled sets can overlap and we can exploit the previously evaluated marginal gains. Second, as in LAZY-GREEDY although the marginal values of the elements might change in each step of the greedy algorithm, often their ordering does not change [Min78]. Hence in line 4 of Algorithm 12 we can apply directly lazy evaluation as follows. We maintain an upper bound $\rho(e)$ (initially ∞) on the marginal gain of all elements sorted in decreasing order. In each iteration i, STOCHASTIC-GREEDY samples a set R. From this set R it evaluates the element that comes on top of the list. Let's denote this element by e. It then updates the upper bound for e, i.e., $\rho(e) \leftarrow \Delta(e|A_{i-1})$. If after the update $\rho(e) \geq \rho(e')$ for all $e' \neq e$ where $e, e' \in R$, submodularity guarantees that e is the element with the largest marginal gain in the set R. Hence, lazy evaluation helps us reduce function evaluation in each round.

12.3 Experimental Results

In this section, we address the following questions: 1) how well does Stochastic-Greedy perform compared to previous art and in particular Lazy-Greedy, and 2) How does Stochastic-Greedy help us get near optimal solutions on large datasets by reducing the computational complexity? To this end, we compare the performance of our Stochastic-Greedy method to the following benchmarks: Random-Selection, where the output is k randomly selected data points from V; Lazy-Greedy, where the output is the k data points produced by the accelerated greedy method [Min78]; Sample-Greedy, where the output is the k data points produced by applying Lazy-Greedy on a subset of data points parametrized by sampling probability p; and Threshold-Greedy, where the output is the k data points provided by the algorithm of [BV14].

In order to compare the computational cost of different methods independently of the concrete implementation and platform, in our experiments we measure the computational cost in terms of the number of function evaluations used. Moreover, to implement the Sample-Greedy method, random subsamples are generated geometrically using different values for probability p. Higher values of p result in subsamples of larger size from the original dataset. To maximize fairness, we implemented an accelerated version of Threshold-Greedy with lazy evaluations (not specified in the paper) and report the best results in terms of function evaluations. Among all benchmarks, Random-Selection has the lowest computational cost (namely, one) as we need to only evaluate the selected set at the end of the sampling process. However, it provides the lowest utility. On the other side of the spectrum, Lazy-Greedy makes k passes over the full ground set, providing typically the best solution in terms of utility. The lazy evaluation eliminates a large fraction of the function evaluations in each pass. Nonetheless, it is still computationally prohibitive for large values of k.

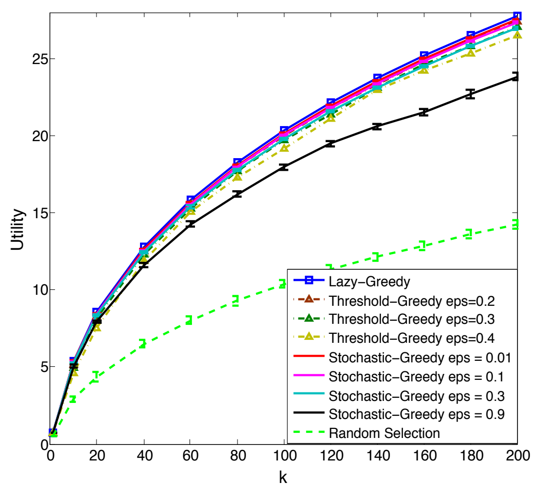
In our experimental setup, we focus on three important and classic machine learning applications: nonparametric learning, exemplar-based clustering, and sensor placement.

12.3.1 Nonparametric Learning

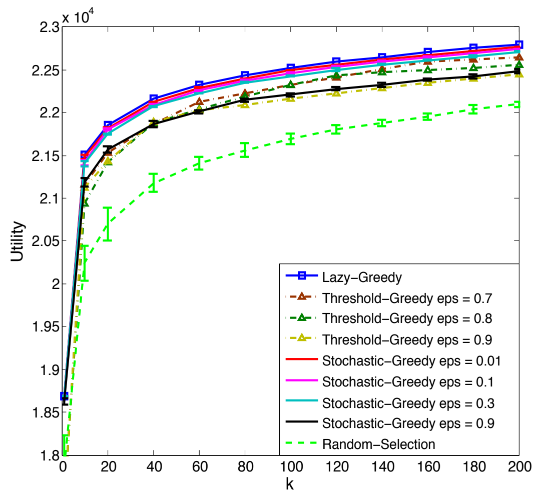
Our first application is data subset selection in nonparametric learning discussed in Section 3.1.1. In our experiment we chose a Gaussian kernel with $h = 0.75$ and $\sigma = 1$. We used the Parkinsons Telemonitoring dataset [Tsa+10] consisting of 5,875 bio-medical voice measurements with 22 attributes from people with early-stage Parkinson's disease. We normalized the vectors to zero mean and unit norm. Figures 12.1a and 12.1c compare the utility and computational cost of STOCHASTIC-GREEDY to the benchmarks for different values of k. For THRESHOLD-GREEDY, different values of ϵ have been chosen such that a performance close to that of LAZY-GREEDY is obtained. Moreover, different values of p have been chosen such that the cost of SAMPLE-GREEDY is almost equal to that of STOCHASTIC-GREEDY for different values of ϵ. As we can see, STOCHASTIC-GREEDY provides the closest (practically identical) utility to that of LAZY-GREEDY with much lower computational cost. Decreasing the value of ε results in higher utility at the price of higher computational cost. Figure 12.3a shows the utility versus cost of STOCHASTIC-GREEDY along with the other benchmarks for a fixed $k = 200$ and different values of ϵ. STOCHASTIC-GREEDY provides very compelling tradeoffs between utility and cost compared to all benchmarks, including LAZY-GREEDY.

12.3.2 Exemplar-based clustering

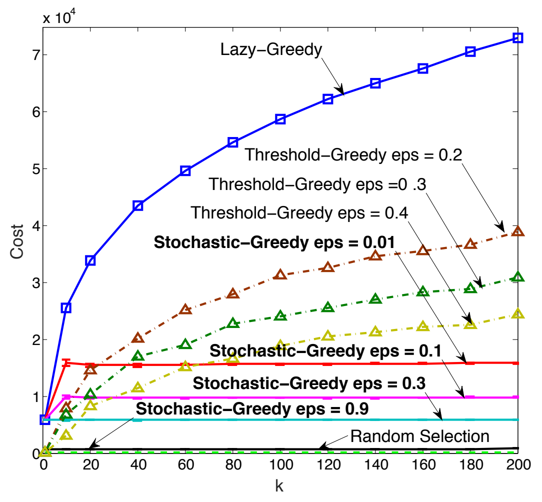
We also applied STOCHASTIC-GREEDY to the problem of exemplar-based clustering discussed in Section 3.2. In our experiment we chose $d(x, x') = ||x - x'||^2$ for the dissimilarity measure. We used a set of 10,000 Tiny Images [TFF08] where each 32×32 RGB image was represented by a 3,072 dimensional vector. We subtracted from each vector the mean value, normalized it to unit norm, and used the origin as the auxiliary exemplar. Figures 12.1b and 12.1d compare the utility and computational cost of STOCHASTIC-GREEDY to the benchmarks for different values of k. It can be seen that STOCHASTIC-GREEDY outperforms the benchmarks with significantly lower computational cost. Figure 12.3b compares the utility versus cost of different methods for a fixed $k = 200$ and various p and ϵ. Similar to the previous experiment, STOCHASTIC-



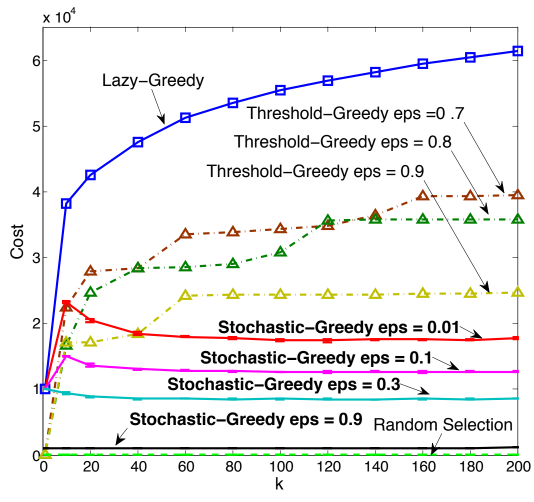
(a) Parkinsons



(b) Images 10K



(c) Parkinsons

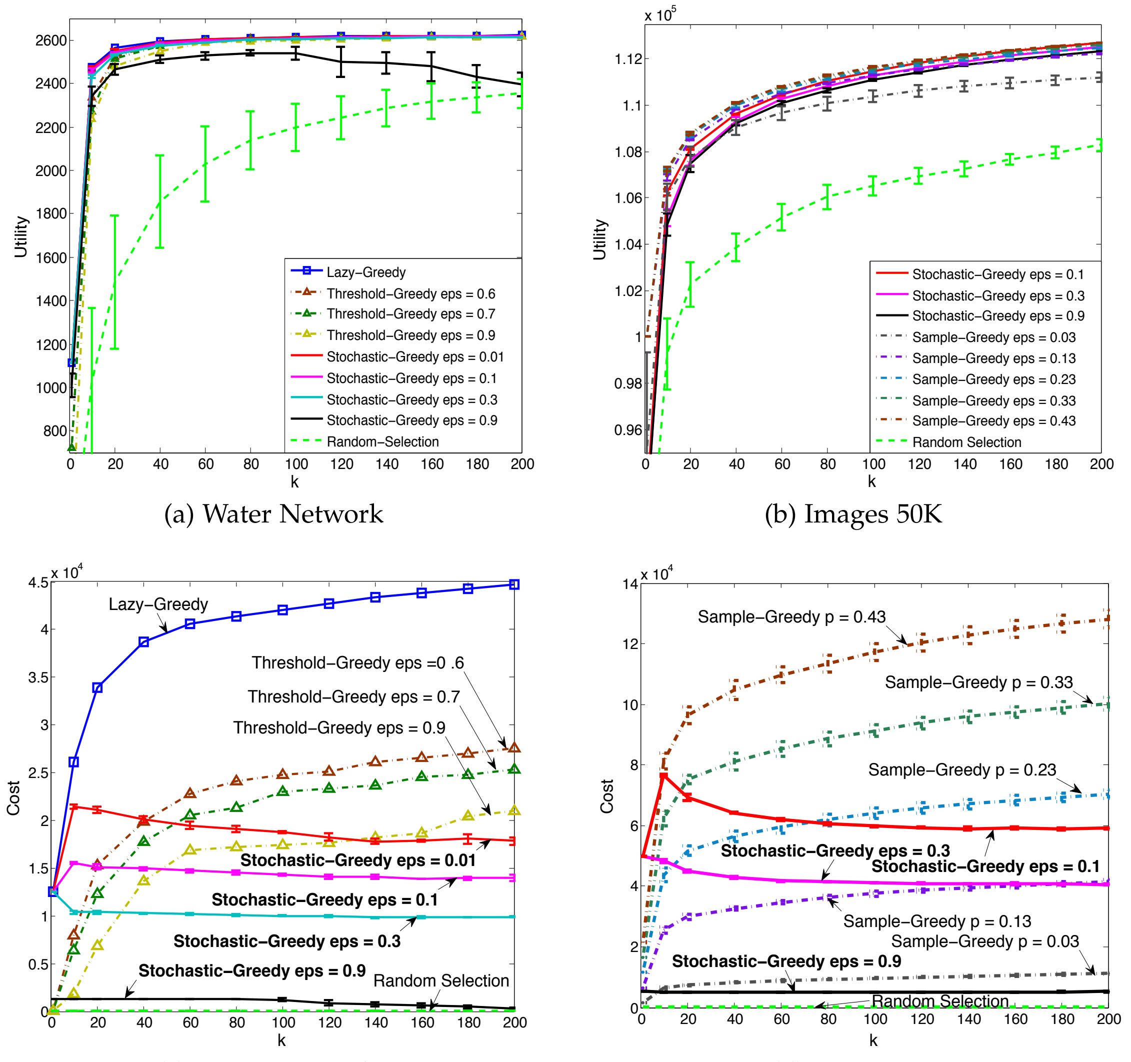


(d) Images 10K

Figure 12.1: *Performance comparisons. a), and b) show the performance of all the algorithms for different values of k on Parkinsons Telemonitoring, and a set of 10,000 Tiny Images respectively. c), and d) show the cost of all the algorithms for different values of k on the same datasets.*

Greedy achieves near-maximal utility at substantially lower cost compared to the other benchmarks.

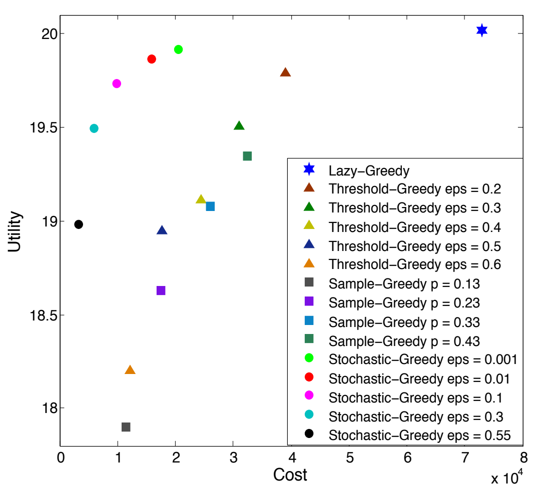
Large scale experiment. We also performed a similar experiment on a larger set of 50,000 Tiny Images. For this dataset, we were not able to run Lazy-Greedy and Threshold-Greedy. Hence, we compared the utility and cost of Stochastic-Greedy with Random-Selection using different values of p. As shown in Figures 12.2b and 12.2d, Stochastic-Greedy outperforms Sample-Greedy in terms of both utility and cost for different values of k. Finally, as Figure 12.3d shows that Stochastic-Greedy achieves the highest utility but performs much faster compare to Sample-Greedy which is the only practical solution for this larger dataset.



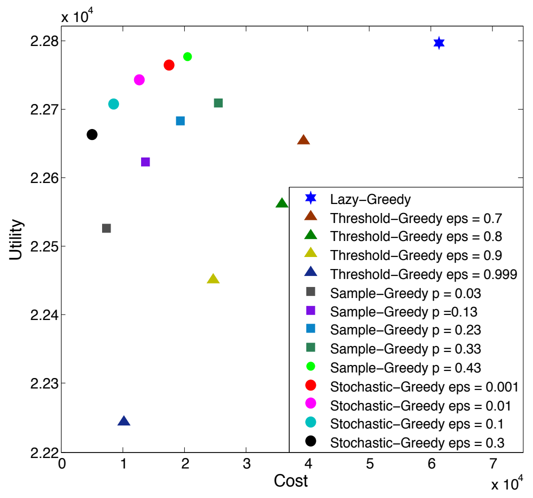
(a) Water Network (b) Images 50K

(c) Water Network (d) Images 50K

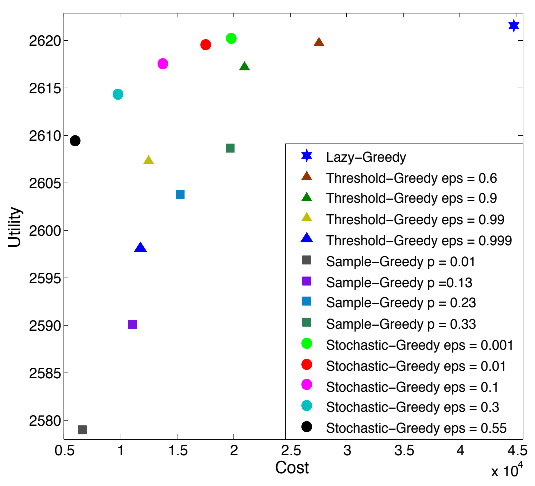
Figure 12.2: *Performance comparisons. a), and b) show the performance of all the algorithms for different values of k on Water Network, and a set of 50,000 Tiny Images respectively. c) and d) show the cost of all the algorithms for different values of k on the same datasets.*



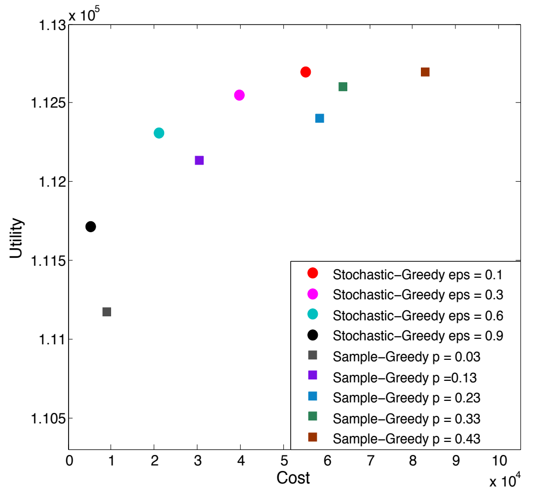
(a) Parkinsons



(b) Images 10K



(c) Water Network



(d) Images 50K

Figure 12.3: *Performance comparisons. The utility obtained versus cost for a fixed* $k = 200$ *on a) Parkinsons Telemonitoring, b) a set of 10,000 Tiny Images, c) Water Network, d) and a set of 50,000 Tiny Images respectively.*

12.3.3 Sensor Placement

Our last experiment involves STOCHASTIC-GREEDY applied to sensor placement (see Sec. 3.2). In our experiments we used the 12,527 node distribution network provided as part of the Battle of Water Sensor Networks (BWSN) challenge [Ost+08]. Figures 12.2a and 12.2c compare the utility and computational cost of STOCHASTIC-GREEDY to the benchmarks for different values of k. It can be seen that STOCHASTIC-GREEDY outperforms the benchmarks with significantly lower computational cost. Figure 12.3c compares the utility versus cost of different methods for a fixed $k = 200$ and various p and ϵ. Again STOCHASTIC-GREEDY shows similar behavior to the previous experiments by achieving near-maximal utility at much lower cost compared to the other benchmarks.

12.4 Summary

We have developed the first linear time algorithm STOCHASTIC-GREEDY with no dependence on k for cardinality constrained submodular maximization. STOCHASTIC-GREEDY provides a $1 - 1/e - \epsilon$ approximation guarantee to the optimum solution with only $n \log \frac{1}{\epsilon}$ function evaluations. We have also demonstrated the effectiveness of our algorithm through an extensive set of experiments. As these show, STOCHASTIC-GREEDY achieves a major fraction of the function utility with much less computational cost. This improvement is useful even in approaches that make use of parallel computing or decompose the submodular function into simpler functions for faster evaluation. The properties of STOCHASTIC-GREEDY make it very appealing and necessary for solving very large scale problems.

13 Fast Constrained Submodular Maximization: Personalized Summarization

In Chapter 12, we saw that data summarization, in the form of extracting a representative subset of k data points, is a natural way to obtain a faithful description of the whole data. In this chapter, we will consider the more general problem of data summarization by maximizing a (non-monotone) submodular function under more general types of constraints. We discussed a similar objective in the streaming setting in Chapter 9. Here, we focus on developing fast centralized algorithms. We start by motivating non-monotone functions for data summarization and various constraints that are often imposed by the underlying application. We then discuss some concrete applications, and introduce our algorithm for constrained non-monotone submodular maximization.

In general, a representative summary has two requirements [Tsc+14; DKR13b]:

- **Coverage:** A good summary is concise so that it contains elements from distinct parts of data. Naturally, a concise summary minimizes information loss.
- **Diversity:** A good summary is compact so that it does not contain elements that are too similar to each other.

Note that coverage and diversity could sometimes be conflicting requirements: higher coverage usually means selecting more elements whereas higher diversity penalizes having similar elements in the summary and prevents the summary from growing too large. Depending on the application, a good summary can trade off between coverage and diversity by putting more emphasis on one or the other. By design, utility functions expressing coverage are *monotone* as it is quit natural to assume that adding more elements to a summary will only decrease the information loss. Such monotone submodular functions have been extensively used for many data summarization applications including clustering [DF07b; GK10], scene summarization [SSS07], document and corpus summarization [LB11b; Sip+12b], recommender systems [EAG11], crowd teaching [Sin+14], and active set selection in kernel machines [SS01; Mir+13]. In contrast, utility functions that accommodate diversity are not necessarily monotone as they penalize larger solutions [Tsc+14; DKR13b]. Consequently, the functions designed to measure both coverage and diversity (e.g., combination of monotone submodular functions and decreasing penalty terms) are naturally *non-monotone*.

On top of maximizing a non-monotone submodular utility function, there are often constraints imposed by the underlying data summarization application. For instance, an individual interested in showing a summary of her recent trip photos may not intend to include more than a handful of them (i.e., cardinality constraint). Or, a user interested in watching representative video clips (with different duration) from a particular category may not wish to spend more than a certain amount of time (i.e., knapsack constraint).

There exist fast and scalable methods to maximize a *monotone* submodular function f with a variety of constraints [BV14; Bad+14; WIB14; Kum+13]. We also discussed a stochastic algorithm for monotone submodular maximization under a cardinality constraint in Chapter 12. As a result, monotone submodular maximization subject to simple constraints (often a cardinality constraint) has been one of the prototypical optimization problems for data summarization.

In this chapter, we aim to significantly push the theoretical boundaries of constrained submodular maximization while providing a practical data summarization method for far richer scenarios. We develop a FAst coNsTrained submOdular Maximization algorithm, FANTOM, for maximizing a not necessarily monotone submodular function f, under the intersection of a p-system and d knapsacks. We show that FANTOM provides a solution with $(1+\epsilon)(p+1)(2p+2d+1)/p$ approximation guarantee with $O(\frac{nrp\log(n)}{\epsilon})$ query complexity (n and r indicate the size of the ground set and the size of the largest feasible solution, respectively).

To the best of our knowledge, there is no algorithm with such strong guarantees for maximizing a *general* submodular function under the aforementioned constrains. Moreover, even in the case of a single matroid and a single knapsack constraint, the best known algorithms suffer from a prohibitive running time that makes them impractical for any effective data summarization applications (see Section 2.3.3). Last but not least, a p-system contains cardinality, matroid, and the intersection of p matroids, as special cases. Thus, it allows us to easily model various data summarization scenarios for which only heuristic methods were known.

We discuss the personalized data summarization in Section 13.2 with three concrete applications: movie recommendation on a dataset containing 11K movies, personalized image summarization on a multi-category dataset with 10K images, and revenue maximization on the YouTube social network with 5000 communities.

13.1 Constrained Submodular Maximization

Our goal in this chapter is to maximize a (non-monotone) submodular function f subject to a set of constraints ζ, i.e.,

$$\max_{S \subseteq V} f(S) \text{ s.t. } S \in \zeta,$$

where ζ is defined by the intersection of a p-system $(V, \mathcal{I})$ and d knapsacks (See Section 2.2.2 for definitions). In other words, we would like to find a set $S \in \mathcal{I}$ that maximizes f where for each knapsack c_i (where $1 \leq i \leq d$) we have $\sum_{e \in S} c_i(e) \leq 1$. For the ease of presentation, we use c_{ij} to denote the cost of element $j \in V$ in the i-th knapsack.

The problem we consider in this chapter is similar to the problem we discussed in Chapter 9. However, unlike Chapter 9 where we considered the streaming setting, our goal in this chapter is to provide fast centralized solutions for constrained (non-monotone) submodular maximization.

13.2 Applications of Personalized Data Summarization

Before explaining how we solve the problem of non-monotone submodular maximization under a p-system and d kanpsack constraints, we discuss three concrete applications with their corresponding utility functions and constraints ζ.

Personalized movie recommendation: Consider a movie recommender system, where a user specifies the genres she is interested in, out of l categories. Moreover, each item has a cost that can represent the monetary cost, duration, or even accessibility of the movie, among many other factors. The recommender system has to provide a *short list* that meets the *user's constraints*, in terms of money, time, or accessibility. To model this scenario, we use intersection of l uniform matroids to prevent each category from having more than a certain number of movies. A knapsack constraint is also used to model the user's limitation in terms of the money she can pay, the time she can spend, or how much effort she has to make to find such movies.

To find a representative set of movies for recommendation, we can use the two non-monotone submodular utility functions introduced in Section 3.6 for movie recommendation. Such utility functions capture both coverage and diversity of the selected subsets, and we can find a diverse and representative set for recommendation by maximizing them under the user specified constraints.

Personalized image summarization: Here, we have a collection of images V from l disjoint categories (e.g., mountains, bikes, birthdays, etc) and the user is interested in a summary only from a few categories. For simplicity, we assume that each image is only a member of a single category. This assumption lets us define a partition matroid consisting of l groups. The user basically identifies the set of desired groups, and a limit on the number of images that can be chosen from each group. The cost of an image is chosen as a function of its quality, such as the resolution, contrast, luminance, etc. For the utility function, we can use the facility location objective penalized by the similarity within the selected subset. This non-monotone submodular utility functions has been discussed in Section 3.5.

Revenue maximization with multiple products: Here, the goal is to offer for free or advertise some of the products $q \in Q$ to a set of users $S \subseteq V$ such that through their influence on others, the revenue increases. The non-monotone submodular objective function introduced in Section 3.8 can be used to find the influential set of individuals for advertising each product. Now, users in a social network may want to see only a small number of advertisements. This requirement can be modeled by a partition matroid. Moreover, nodes with higher (weighted) degrees are usually more influential and harder to get. So we also define a cost $c_i = c_i(\sum_{j \in V} w_{ij})$ for including a node i to a set of users targeted for advertising any product. Again, the total cost cannot exceed a threshold modeled by a knapsack.

13.3 Our Algorithm: Fantom

In this section, we describe a very fast algorithm for maximizing a non-monotone submodular function subject to the intersection of a p-system and d knapsack constraints. Our algorithm is a novel combination of two algorithms: a local search algorithm discussed in Chapter 9 (*c.f.* Section 9.3.1) for maximizing *non-monotone* submodular function subject to a p-system [Gup+10b], and an algorithm for maximizing *monotone* submodular function subject to a p-system and d knapsack constraints [BV14]. Additionally we tighten the analysis of [Gup+10b] to get a better approximation ratio even for the case of $d = 0$.

Our algorithm is split into three parts. In the first part we take the most natural algorithm for maximizing submodular functions, i.e., the celebrated greedy algorithm. We restrict the greedy algorithm to only pick elements with enough "density" for knapsack constraints. In general, greedy algorithms don't tend to work well for non-monotone submodular functions. We prove that the algorithm either picks a good enough of a solution, or if we throw away the greedy solution, the optimal solution in the remaining elements is not too bad. Based on this observation, in the second part we iterate the greedy algorithm multiple times on the remaining elements to generate multiple solutions and pick the best among them. In the third and final part, we discretize and iterate over all possible values of "density" as defined in the first part.

13.3.1 Greedy with Density Threshold (GDT)

In the first part, we consider a natural variant of the greedy algorithm, where we pick elements in a greedy manner while simultaneously restricting it to pick elements with enough "density" for knapsack constraints. I.e., GDT (outlined in Alg. 13) does not pick elements if the ratio of the marginal value of the element to the sum of its costs for each knapsack is below a given threshold.

Algorithm 13: GDT - Greedy with density threshold

```
input  $f : 2^V \to \mathbb{R}_+$, a membership oracle for $p$-system $\mathcal{I} \subset 2^V$, and $d$ knapsack
       cost functions $c_i : V \to [0,1]$, density threshold $\rho$.
output A set $S \subseteq V$ satisfying $S \in \mathcal{I}$ and $c_i(S) \leq 1 \forall i$.
  1: Run greedy and at each step pick the element if and only if $\frac{f_S(j)}{\sum_{i=1}^{l} c_{ij}} \geq \rho$, where
     $f_S(j) = f(S \cup \{j\}) - f(S)$
  2: Let $z = \arg\max\{f(j) | j \in V\}$
  3: Return $\arg\max(f(S), f(\{z\}))$
```

Theorem 30. *For any set $C \in \mathcal{I}$,* GDT *outputs a set $S \in \mathcal{I}$ such that*

$$f(S) \geq \min\left(\frac{\rho}{2}, \frac{1}{p+1} f(S \cup C) - \frac{d\rho}{p+1}\right).$$

The proofs of the theorems in this Chapter can be found in Appendix A.7.

13.3.2 Iterated Greedy with Density Threshold (IGDT)

While greedy tends to perform well for monotone functions, it can pick really bad solutions for non-monotone functions. In this part, we run GDT multiple times, each time on remaining elements to get multiple solutions. We prove that this process produces at least one reasonable solution.

Theorem 31. *For any set $C \in \mathcal{I}$,* IGDT *(outlined in Alg. 14) outputs a set $S \in \mathcal{I}$ such that*

$$f(S) \geq \min\left(\frac{\rho}{2}, \frac{p}{(p+1)(2p+1)} f(C) - \frac{d\rho}{2p+1}\right)$$

13.3.3 Fantom

In this section, we consider the final piece of the puzzle. In the previous two algorithms, we consider the density threshold to be a given number. In our final algorithm we

Algorithm 14: IGDT: Iterated greedy with density threshold

```
input $f : 2^V \to \mathbb{R}_+$, a membership oracle for $p$-system $\mathcal{I} \subset 2^V$, and $d$ knapsack
    cost functions $c_i : V \to [0,1]$, density threshold $\rho$.
output A set $S \subseteq V$ satisfying $S \in \mathcal{I}$ and $c_i(S) \leq 1 \forall i$.
1: $\Omega = V$
2: for $i = 1; i \leq p+1; i++$ do
3:     $S_i = \text{GDT}(f, \Omega, \rho)$
4:     $S'_i = \text{Unconstrained-Maximization}(S_i)$
5:     $U = U \cup \{S_i, S'_i\}$
6:     $\Omega = \Omega - S_i$
7: end for
8: Return $\arg\max\{f(S) | S \in U\}$
```

discretize the set of density thresholds into $\log(n)/\epsilon$ different possible values and run the previous algorithm on each of them. We finally show that for at least one of the discretized density thresholds we should get a good enough solution.

Algorithm 15: FANTOM

```
input $f : 2^V \to \mathbb{R}_+$, a membership oracle for $p$-system $\mathcal{I} \subset 2^V$, and $d$ knapsack
    cost functions $c_i : V \to [0,1]$.
output A set $S \subseteq V$ satisfying $S \in \mathcal{I}$ and $c_i(S) \leq 1 \forall i$.
1: $M = \max_{j \in V} f(j), \gamma = \frac{2 \cdot p \cdot M}{(p+1)(2p+1)}, U = \{\}$
2: $R = \{\gamma, (1+\epsilon)\gamma, (1+\epsilon)^2\gamma, (1+\epsilon)^3\gamma, \ldots, \gamma \cdot n\}$
3: for $\rho \in R$ do
4:     $S = \text{IGDT}(f, \Omega, \rho)$
5:     $U = U \cup \{S\}$
6: end for
7: Return $\arg\max\{f(S) | S \in U\}$
```

Theorem 32. FANTOM *(outlined in Alg. 15) has an approximation ratio* $(1+\epsilon)(p+1)(2p+2d+1)/p$ *with running time* $O(\frac{nrp\log(n)}{\epsilon})$.

Without any knapsack constraints ($d = 0$), each call to IGDT (Alg. 14) in FANTOM returns the same solution. Hence, for the case of $d = 0$, we obtain an improved approximation guarantee of $(p+1)(2p+1)/p$, with a similar running time $O(nrp)$ to [Gup+10b].

Proposition 33. *For the case of* $d = 0$, FANTOM *has a* $(p+1)(2p+1)/p$*-approximation ratio with* $O(nrp)$ *running time.*

As discussed in Section 2.3.3, (see Table 2.4), even for the special case of 1 matroid and 1 knapsack constraints, all the existing algorithms have exorbitant running times and cannot be implemented in any reasonable time in practice. There are two main reasons for this. The first is due to an expensive enumeration step running over all subsets of very large size, and the second is due to running the continuous greedy algorithm. To compare our algorithms against practical baselines in Section 13.4, we consider two heuristics based on classical methods for maximizing submodular functions.

Greedy: Our first baseline starts with an empty set $S = \phi$ and keeps adding elements one by one greedily while the p-system and d knapsack constraints are satisfied.

Density Greedy: Our second baseline starts with an empty set $S = \phi$ and keeps adding elements greedily by their value to total-knapsack cost ratio while the p-system and d knapsack constraints are satisfied.

The above heuristics do not have provable performance guarantees as shown by the following examples.

Bad example for Greedy. Let $n = |V|$ be the number of elements in the ground set and let $m = n/2$. Define sets $T_i = \{y_i, z_i\}$ for $1 \leq i \leq m$. Let $V = \cup_{i=1}^{m} T_i$. Let $Y = \{y_1, y_2, \ldots, y_m\}$ and $Z = \{z_1, z_2, \ldots, z_m\}$. Let $\epsilon > 0$ be a small constant. Define the submodular function

$$\forall S \subseteq V, f(S) = (1+\epsilon) \cdot |S \cap Y| + |S \cap Z|$$

and the following two constraints.

1. A partition matroid constraint where S is a feasible solution if for $1 \leq i \leq m, |S \cap T_i| \leq 1$.
2. A knapsack constraint where cost is defined as follows. For any $e \in Y, c(e) = 1 - \frac{1}{2m}$ and for any $e \in Z, c(e) = 1/m$.

Then, it is easy to see that Baseline 1 picks a set $S = \{y_i\}$ for some i and gets value $1+\epsilon$, while the optimal solution is Z of value $m = n/2$.

Bad example for Density Greedy. Let $T_1 = \{y_1, z_1\}$ and $T_2 = \{y_2, z_2\}$ and $V = T_1 \cup T_2$. Let $Y = \{y_1, y_2\}$ and $Z = \{z_1, z_2\}$. Let $\epsilon > 0$ be a small constant. Define the submodular function

$$\forall S \subseteq V, f(S) = \epsilon \cdot |S \cap Y| + |S \cap Z|$$

and the following two constraints.

1. A partition matroid constraint where S is a feasible solution if for $1 \leq i \leq 2, |S \cap T_i| \leq 1$,
2. A knapsack constraint where cost is defined as follows. For any $e \in Y, c(e) = \epsilon/2$ and for any $e \in Z, c(e) = 1/2$.

Then, it is easy to see that Baseline 2 picks a set $S = Y$ for some i and gets value 2ϵ, while the optimal solution is Z of value 2. Hence the approximation ration is at least $1/\epsilon$ and as $\epsilon \to 0$ we get unbounded approximation ratio.

13.4 Experiments

In this section, we evaluate FANTOM on the three real-world applications we described in Section 13.2: personalized movie recommendation, personalized image summarization, and revenue maximization. The main goal of this section is to validate our theoretical results, and demonstrate the effectiveness of FANTOM in practical scenarios where existing algorithms are incapable of providing desirable solutions.

13.4.1 Personalized movie recommendation

Our personalized recommendation experiment involves FANTOM applied to a set of 10,437 movies from the MovieLens ratings database [Mov]. Each movie is associated with a 25 dimensional feature vector calculated from user ratings. There are 19 genres in total, and each movie is associated with at most 8 genres. We used the inner product of the non-normalized feature vectors to compute the similarity $s_{i,j}$ between movies i and j (this idea was inspired by [LWD15]). The costs c_i are drawn from the $\text{Beta}(10, 2)$ cumulative distribution $c_i = F_{\text{Beta}(10,2)}(r_i)$, where $r_i \in (0, 1)$ is the normalized average rating of movie i. The Beta distribution lets us differentiate the highly rated movies from those with lower ratings and can be used as a proxy for the cost of watching different movies.

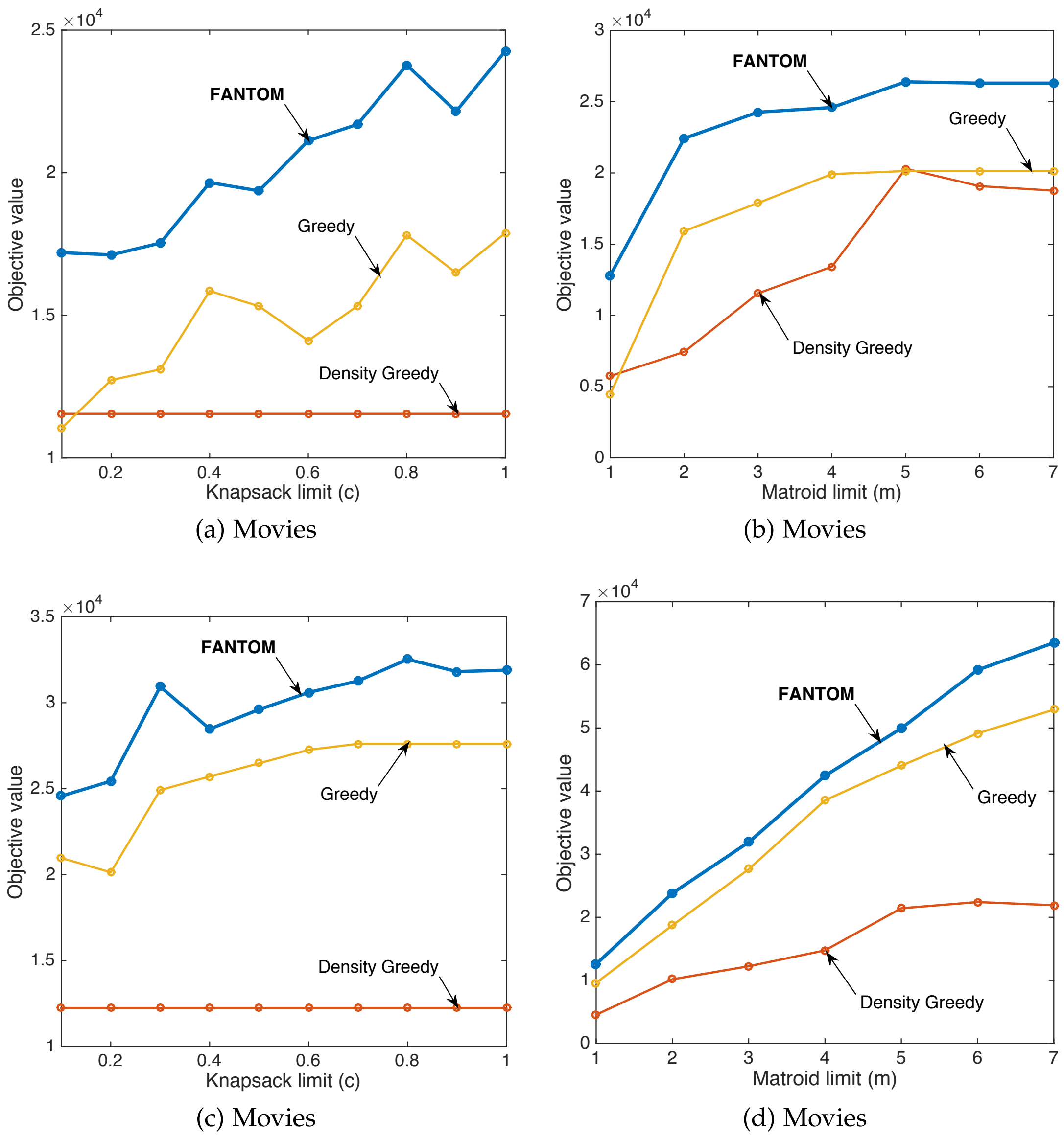


Figure 13.1: *Performance of* FANTOM *compared to the benchmarks for movie recommendation from a set of 10,437 movies from MovieLens. a) shows the performance of* FANTOM *based on Eq. 3.6.1 for recommending movies from three genres: adventure, animation, and fantasy for $m = 3$, and varying knapsack limit c. b) shows the same quantity for $c = 1$, and varying the matroid limits m. c) shows the solution value based on Eq. 3.6.2 with $m = 3$, and varying c. d) shows the same quantity for $c = 1$, and varying m.*

Figure 13.1a compares the performance of our approach to the benchmarks using Eq. 3.6.1 with $\lambda = 1$ for three genres: adventure, animation, and fantasy. A total of $l = 19$ uniform matroid constraints are considered to limit the number of movies

Eq.	FANTOM			Greedy			Density Greedy		
	#	Av. rate	Genres	#	Av. rate	Genres	#	Av. rate	Genres
3.6.1	1	3.99	**1,2**,15	6	4.20	**2**,3,**9**	11	1.16	3,4
	2	3.98	3,4,5,**9**	7	4.21	**1,2**	12	1.16	**2**,4
	3	4.00	**2**,3,4,**9**,19	8	3.26	**1,2**,8,14	13	1.25	**2**,4,5,**9**
	4	3.08	**9**,11	9	2.66	**2**,5,12	14	1.34	**2**,3,4,**9**
	5	2.53	3,4,5	10	1.96	**1,2**,4	15	1.39	**1,2**,4
3.6.2	16	3.91	**2**,3,5,**9**,14	21	3.78	**1,2**,3,4,5,**9**	12	1.16	**2**,4
	17	3.70	**2**,3,4,8,**9**	22	3.75	**1,2**,3,4,5,**9**	13	1.25	**2**,4,5,**9**
	18	3.78	**1,2**,5,12,14,16	23	4.06	**1,2**,3,4,**9**,15	14	1.34	**2**,3,4,**9**
	19	3.53	**1,2**,3,4,5	24	3.82	**2**,5,8,**9**,13,15,16	11	1.16	3,4
	20	3.16	**1,2**,5,**9**,11,16	25	2.08	**1,2**,4,5,**9**,15	15	1.39	**1,2**,4

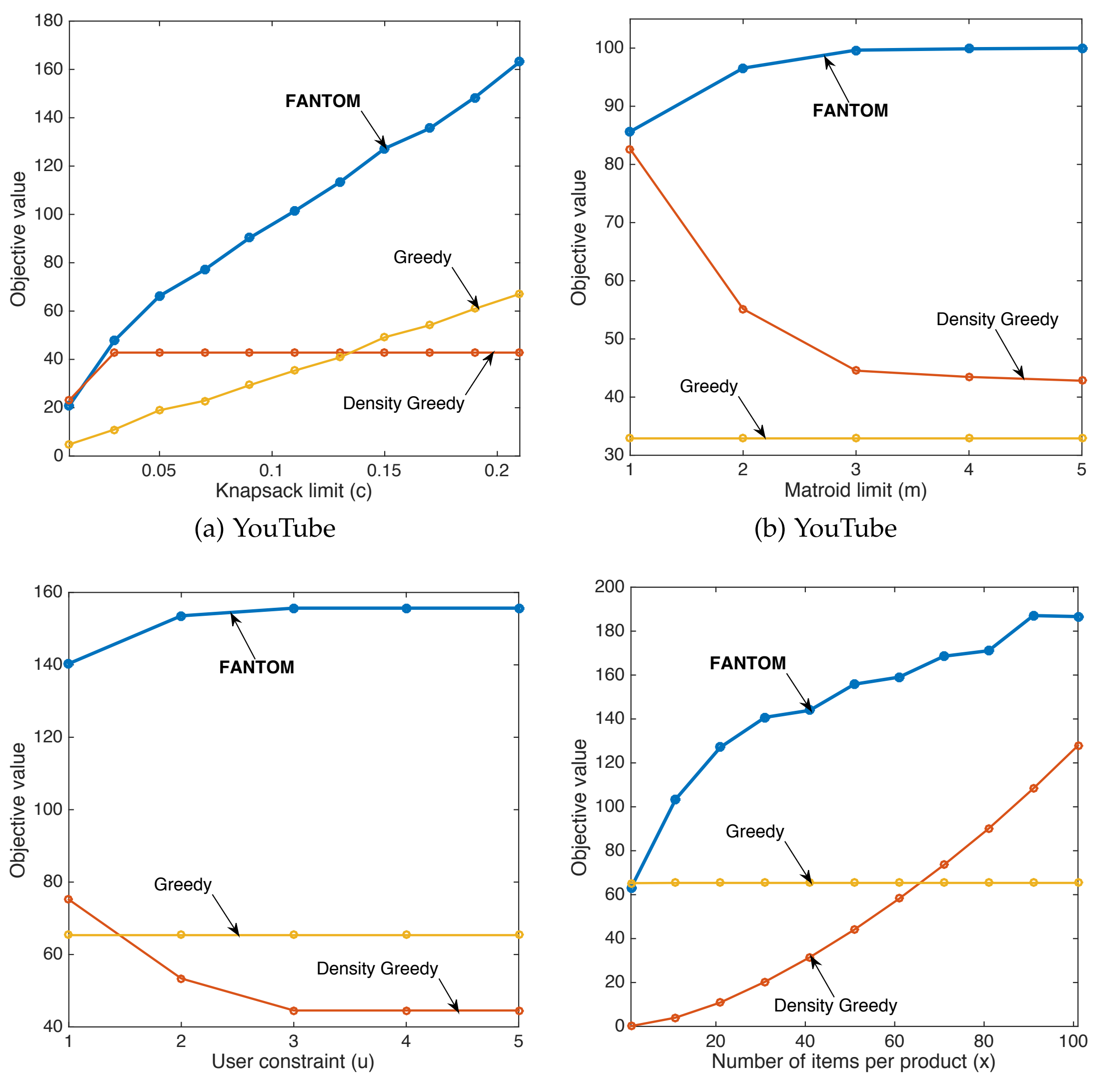
Table 13.1: *Movies recommended by* FANTOM *vs. Greedy and Density Greedy using Eq. 3.6.1, and Eq. 3.6.2 for* $m = 5$ *and* $c = 1$*. There are 19 genres in total: Action(1), Adventure(2), Animation (3), Children (4), Comedy (5), Crime (6), Documentary (7), Drama (8), Fantasy (9), Film-Noir (10), Horror (11), Musical (12), Mystery (13), Romance (14), Sci-Fi (15), Thriller (16), War (17), Western (18), IMAX (19). The user is interested in adventure, animation, and fantasy movies (genres* **1,2,9***). See the Appendix for a complete list of movie names.*

chosen from each of the 19 genres. The limits for all the matroid constraints are set to 3. We also considered an additional uniform matroid constraint to restrict the size of the final solution to 10 movies. Moreover, a knapsack constraint is considered to model the total available budget. It can be seen that FANTOM significantly outperforms the benchmarks for different knapsack limits. Figure 13.1b shows similar qualitative behavior for a fixed knapsack limit $c = 1$, and varying matroid limits m associated with each of the 19 genres. Again, FANTOM is able to show a good performance in scenarios where Greedy and Density Greedy perform arbitrary poorly. Figures 13.1c and 13.1d show the same qualitative behavior using Eq. 3.6.2. Table 13.1 summarized the movies recommended by different methods, along with their average rating and associated genres. We can see that by using Eq. 3.6.2 the recommended movies have more common genres with what the user requested.

13.4.2 Revenue maximization with multiple products

Our larger scale experiment involves applying FANTOM to maximize the revenue function defined in Eq. 3.8.2. We performed our experiment on the top 5000 largest communities of the YouTube social network consists of 39,841 nodes and 224,235 edges [YL15]. We consider the settings where we are to advertise $|Q| = q$ different types of product across all communities of the same social network. For simplicity, we assume that there are x units available from each product, and the influence of individuals on each other is the same for all product types. The edge weights are assigned according to a uniform distribution $\mathcal{U}(0,1)$, and the cost of selecting each node c_i is determined according to an exponential cumulative distribution function of the normalized sum of its edge weights. For the exponential distribution, we chose the parameter $\lambda = 0.2$ to scale the costs to the interval $[0,1]$. To model different characteristics of the products, we model the revenues by the concave function $v_i^q(S) = \alpha_q \sqrt{\sum_{j\in S} w_{i,j}}$, where α_q depends on the type of the product. We used $q = 10$ different values $\alpha_q \in [0.8, 1.3]$ to model the revenue of different product types. Finally, we modeled user constraints by a partition matroid that puts a limits u on the number of products that can be offered to each user. Another partition matroid is employed to restrict the number of products m offered for free to users in each community.

Figure 13.2a shows the revenue obtained by FANTOM versus the budget c when there are $x = 50$ free items available from each product, the number of individuals that can be selected from each community is limited to $m = 5$, and the number of products that can be offered to each user is at most $u = 3$. We note again that FANTOM significantly outperforms the other benchmarks. Figure 13.2b shows the same behavior for varying the matroid limit m, when $x = 50$ and budget $c = 0.1$. Similarly, Figure 13.2c shows the performance of FANTOM for $m = 5$, $x = 50$, $c = 0.2$, and varying the user constraints u. Finally, Figure 13.2d shows the performance of FANTOM for $m = 5$, $u = 3$, $c = 0.2$, and varying the number of available items x from each product type.



(a) YouTube (b) YouTube

(c) YouTube (d) YouTube

Figure 13.2: *Performance of* FANTOM *compared to the benchmarks for revenue maximization on top 5000 communities of YouTube with 39,841 nodes and 224,235 edges. a) shows the performance of* FANTOM *for selling $q = 10$ product types, with $x = 50$ available items per product, matroid limit $m = 5$ for all communities, user constraint $u = 3$, and varying knapsack limit c. b) shows the same quantity for $c = 0.1$, $q = 10$, $x = 50$, $u = 3$ and varying m. c) shows the solution value for $c = 0.2$, $q = 10$, $x = 50$, $m = 3$, and varying u. d) shows the same quantity for $c = 0.2$, $q = 10$, $m = 5$, $u = 3$ and varying x.*

13.4.3 Personalized image summarization

Our personalized recommendation experiment involves FANTOM applied to Eq. 3.7.1. We performed our experiments on a set of 10,000 Tiny Images [KH09]. The images belong to 10 classes, with 1000 images per class. Each 32 by 32 RGB pixel image was represented by a 3,072 dimensional vector. We used the inner product to compute the similarity $s_{i,j}$ between image i and j. The costs are chosen proportional to the normalized variance of the image pixels as a simple technique to calculate image qualities. This way, we assign a higher cost to images with higher contrast and a lower cost to blurry images.

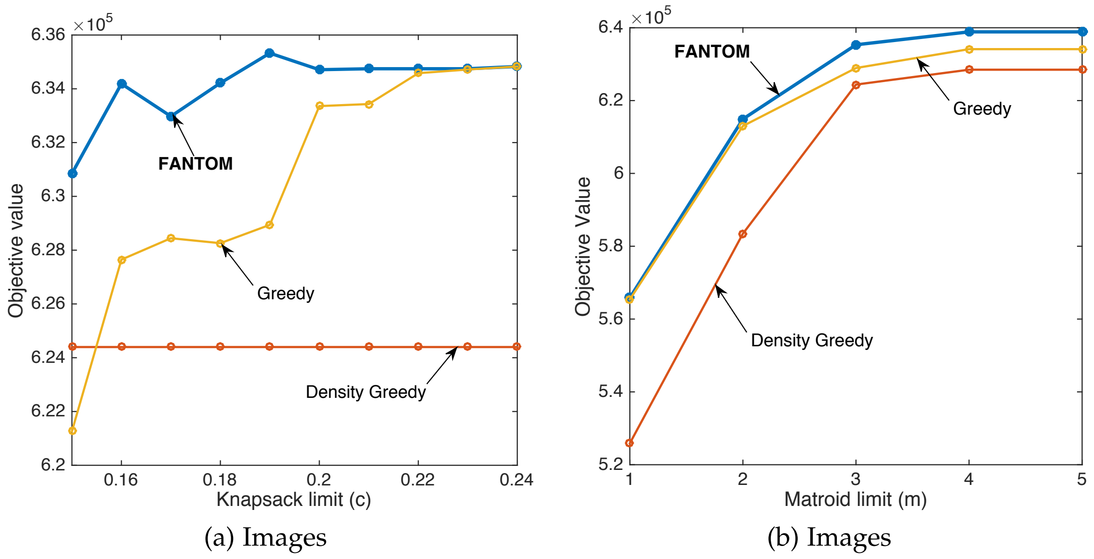


Figure 13.3: *Performance of* FANTOM *compared to the benchmarks for personalized image summarization: a) shows the solution value for summarizing three categories airplane, automobile, and bird for $m = 3$, and varying the knapsack limit c. b) shows the same quantity for $c = 0.1$, and varying the matroid limits m.*

A partition matroid constraint is considered to limit the number of images chosen from each of the specified categories. Moreover, a knapsack constraint is employed to model the limited available budget. Figure 13.3a compares the performance of our approach to the benchmarks for summarizing images from three categories: airplane, automobile, and bird. The results are shown for varying knapsack limit c, while the maximum number of images allowed from each category is set to $m = 3$. Similarly, Figure 13.3b shows the results for fixed $c = 0.1$, and varying m. We find again that FANTOM significantly outperforms the benchmarks.

13.5 Summary

In this chapter, we cast personalized data summarization as an instance of a general submodular maximization problem subject to multiple constraints. We develop the first practical and fast constrained submodular maximization algorithm, FANTOM, that provides a $a(1-\varepsilon)p/(p+1)(2p+2l+1)$ approximation guarantee for maximizing a (not necessarily monotone) submodular function subject to the intersection of a p-system and d knapsack constrains. We have also showed the application of FANTOM to various personalized data summarization problems: movie recommendation on a dataset containing 11K movies, personalized image summarization on a multi-category dataset with 10K images, and revenue maximization on the YouTube social network with 5000 communities. As stated earlier, FANTOM can be easily integrated into the existing distributed methods to further scale up (non-monotone) submodular summarization under general constraints.

Part V

Conclusion and Future Research Directions

14 Conclusions

Data summarization is a compelling (and sometimes the only) approach that aims at both exploiting the richness of large-scale data and being computationally tractable. In this Thesis, we studied massive data summarization using submodular functions and studied the fundamental question:

Is it possible to scale up submodular maximization techniques?

Classical approaches to submodular optimization require random access to the entire data, make multiple passes, and select elements sequentially in order to produce near optimal solutions. Once the size of the dataset increases beyond the memory capacity (typical in many modern datasets) or the data is arriving incrementally over time, neither the greedy algorithm, nor its accelerated versions can be used. Naturally, such solutions cannot scale to large instances. The limitations of centralized methods inspired the design of parallel computing methods, or streaming algorithms that are able to gain insights from data as it is being collected.

In this Thesis, we presented novel approaches for large-scale submodular maxization. Our algorithms can be grouped to three classes, distributed algorithms, streaming algorithms, and fast centralized algorithms.

14.1 Summary

In this section, we will briefly summarize the key contributions presented in this Thesis.

14.1.1 Distributed Algorithms

In part II of this Thesis, we considered distributed approaches for scaling up submodular summarization techniques. We first studied the problem of submodular function maximization in a distributed fashion. We developed a simple, parallel protocol called GreeDi for distributed submodular maximization. It requires minimal communication, and can be easily implemented in MapReduce style parallel computation models. We theoretically characterized its performance, and showed that under some natural conditions, for large datasets the quality of the obtained solution is competitive with the best centralized solution. Our experimental results (implemented with Hadoop) demonstrated that our approach leads to parallel solutions that are typically within 97% of those obtained via centralized methods.

We then considered the problem of submodular cover in a distributed setting. We developed a distributed algorithm – DisCover – for solving the submodular cover problem. It can be easily implemented in MapReduce-style parallel computation models and provides a solution that is competitive with the (impractical) centralized solution. We also studied a natural trade-off between the communication cost (for each round of MapReduce) and the number of rounds. Our experimental results (implemented with Spark) demonstrated the effectiveness of our approach on a variety of submodular cover instances.

We next introduced a more general framework for data summarization, namely, public-private data summarization. This setting is motivated by privacy concerns in many modern online platforms, where the dataset consists of public data, shared among all users, and disjoint sets of private data accessible to the owners only. To address this problem, we proposed a fast distributed algorithm, FastCover, that enables us to solve the problem of covering multiple submodular functions in one run of the algorithm. We showed that FastCover returns a solution that is competitive to that of the best centralized, polynomial-time greedy algorithm. The superior practical performance of FastCover against all the benchmarks was demonstrated through a large set of experiments using Spark.

14.1.2 Streaming Algorithms

In part III, we studied streaming algorithms able to efficiently summarize useful information from massive data "on the fly". We first provided the first single pass streaming algorithm, Streaming Local Search, for maximizing non-monotone submodular functions subject to a collection of independence systems and multiple knapsack constraints. We showed it's effectiveness on video summarization for producing online summaries in the streaming setting. The ability of our method to generates real-time summaries after receiving each element from the stream is of significant importance in privacy sensitive applications.

We then introduced dynamic submodular maximization, and developed the first deletion-robust streaming algorithm –Robust-Streaming – for constrained submodular maximization. For a single-pass streaming algorithm StreamingAlg with approximation guarantee α, Robust-Streaming uses multiple instances of StreamingAlg to output a solution that is robust against m deletions. The returned solution also satisfies an approximation guarantee w.r.t to the solution of the optimum centralized algorithm that knows the set of m deletions in advance. Our experiments showed that Robust-Streaming can immediately update the solution in case of deletions, and significantly improves the performance of the existing streaming approaches. This property of Robust-Streaming makes it an appealing approach for solving very large-scale applications on dynamic datasets that experience deletions very often. Given the importance of submodular optimization to numerous data mining and machine learning applications, we believe our results provide an important step towards addressing such problems at scale.

14.1.3 Fast Centralized Algorithms

In complementary to the aforementioned methods for scaling up submodular optimization techniques, in part IV, we developed a fast centralized algorithms that can be integrated into the existing distributed frameworks to provide further scalability.

In particular, we developed a randomized technique for maximizing a monotone submodular function that is faster than the widely used lazy greedy algorithm, both in theory and practice. We showed that our randomized algorithm, Stochastic-Greedy, can achieve the same approximation guarantee as greedy, in expectation, to the optimum solution in time linear in the size of the data and independent of the cardinality constraint. As shown by our experiments, STOCHASTIC-GREEDY achieves a major fraction of the function utility provided by the greedy algorithm, with much less computational cost.

Lastly, we considered summarization of multi-category data based on user preferences. We showed that this problem can be formulated as maximizing a (not-necessarily monotone) submodular function subject to intersection of a p-system and d knapsack constraints. While algorithms with good approximation ratios existed for this problem, they all suffered from a prohibitive running time that make them impractical for any effective data summarization applications. We developed the first practical and fast constrained submodular maximization, FANTOM, algorithm with strong theoretical guarantees, and showed it's effectiveness on several personalized data summarization applications.

14.1.4 Applications

In addition to providing algorithms and theoretical analyses, we presented extensive empirical evaluation of our approaches on several large-scale real-world problems.

A major focus of this Thesis were on summarizing massive data. We used our large-scale methods to summarize 80 million Tiny Images, more than 45 million user visits from the Featured Tab of the Today Module on Yahoo! Front Page, as well as finding dominating sets in Friendster social network with more than 65.6 million nodes and 1.8 billion edges.

We also considered the problem of public-private data summarizaton, and reported the performance of our distributed algorithms using Spark on concrete applications of this problem, including personalized movie recommendation on a dataset containing more than 2 million ratings by more than 100K users for 11K movies, personalized location recommendation based on 20 users and their collected GPS locations, and finding dominating sets on a social network containing more than 65 million nodes and 1.8 billion edges.

In addition to the above large-scale applications, we used our algorithm for personalized data summarization. In particular, we provided personalized movie recommendations among over 11K movies of the MovieLens database from 19 genres, personalized image summarization with 10K images, and revenue maximization on the YouTube social networks with 5K communities. In the streaming setting, we showed that for the personalized video summarization problem, our method, while achieving practically the same performance, runs more than 1700 times faster than previous work.

Finally, we showed the effectiveness of our algorithms for summarizing a dynamic data stream with “the Right to be Forgotten”. Here, the user can decide to delete a subset of elements from the selected summary at any time, and we should be able to update the summary immediately. We evaluated our approach on several real-world applications, including summarizing geo-tagged mobile air-quality measurements collected on bikes; a streaming image collection application; and click-stream log data, consists of 45.8 million points, from a news recommendation task.

14.2 Future Research Directions

A tremendous amount of data is generated every second, and demand fast analysis and efficient storage. Examples include massive clickstreams (e.g., Google, Yahoo, etc), stock market data, log aggregation, image and videos (e.g., Instagram, YouTube, etc.), and sensor data for environmental or health monitoring. Our long-term research goal is to harness such massive amount of data from various sources, be able to make predictions about future events, and ultimately, enhance social and technological systems. Exploiting such large-scale data allows us to build large-scale machine learning methods for predictive as well as prescriptive analytics, and will have applications in several domains, including medical and health care systems, building and home automation, environmental sensing and urban planning. In order to achieve this long-term goal, we define the following intermediate steps and directions.

- Extracting representative elements and patterns in data from various sources.
- Developing statistical machine learning models and algorithms to make inference.
- Deriving predictive and prescriptive analytics in a timely manner.
- Scaling the developed techniques to massive datasets.

To make progress on this long-term goal, we pursue the following research directions.

Developing appropriate abstractions When combining and interpreting data from potentially heterogeneous sources, we often need to develop appropriate abstractions for each source of data. For example, when discovering routines from human locations, we may use *location labels* instead of the raw geo-location tags. Similar scenario happens, when receiving real-time data from thousands of potentially different types of sensors. Developing appropriate abstractions allow for identical higher-order reasoning about the underlying heterogeneous data. Furthermore, such abstractions facilitate modeling streams of data as time series, and allow for recording historical trends.

Actively deriving predictive and prescriptive analytics. The extracted nuggets of insightful information can be exploited to develop appropriate statistical and probabilistic models. We will then be able to take advantage of machine learning methods to derive predictive analytics. This requires solving difficult challenges, as we need to combine evidence from different sources, data may be incomplete or inconsistent, and the underlying environment may be time varying. Moreover, in many situations, data and observations from various sources may not be independent. In such scenarios, we need scalable hierarchical and mixture models, which are able to handle this type of mutual dependence. Beyond developing predictive models, we want to be able to track the root causes of various events in an autonomous manner, and make appropriate suggestions to improve the performance and efficiency of the underlying system.

Dealing with massive data and scalability. One of our primary research focuses is on large scale data analysis and manipulation. The massive volume of data received from various sources is prohibitively expensive to analyze. In order to permit timely analysis of such rapid data we need techniques able to sift through huge amount of data and extract insightful information. In this Thesis, we made several steps towards this long-term goal. Nevertheless, further scaling up and improving the efficiency of the developed techniques provides rich opportunities for further research.

We further need large-scale, computationally tractable machine learning algorithms for inferring model parameters, and make predictions. The important question here is what kinds of analyses are suitable for making inference on such heterogeneous data, and how to further scale up data mining and machine learning algorithms to deal with such data. These directions pose great opportunities for scholarly study and will allow us to find patterns that are practically observable in the combined data from various sources.

Taking privacy concerns into account. Dealing with personal data, including traces of users' activities on social networks (posts, tweets, etc) or images/videos taken with wearables such as Google Glass, privacy concerns become a growing concern. When mining and analyzing data, we want to preserves the right of individuals to have control over the information they reveal about themselves. This becomes of greater importance when investigating information from a wide variety of sources, as the combined data can increase the chance of revealing undesirable information about the individuals. In this Thesis, we considered dynamic data summarization, where user can decide to delete a subset of elements from the selected summary at any time. We plan to extend such privacy preserving methods to heterogeneous data from various sources.

A

Proofs

A.1 Proofs from Chapter 5

This section presents the complete proofs of theorems presented in the article.

Proof of Theorem 4

$\Rightarrow$ direction:
The proof easily follows from the following lemmas.

Lemma 34. $\max_i f(A_i^c[k]) \geq \frac{1}{m} f(A^c[k])$.

Proof. Let B_i be the elements in V_i that are contained in the optimal solution, $B_i = A^c[k] \cap V_i$. Then we have:

$$f(A^c[k]) = f(B_1 \cup \ldots \cup B_m) = f(B_1) + f(B_2|B_1) + \ldots + f(B_m|B_{m-1}, \ldots, B_1).$$

Using submodularity of f, for each $i \in \{1 \ldots m\}$, we have

$$f(B_i|B_{i-1} \ldots B_1) \leq f(B_i),$$

and thus,

$$f(A^c[k]) \leq f(B_1) + \ldots + f(B_m).$$

Since, $f(A_i^c[k]) \geq f(B_i)$, we have

$$f(A^c[k]) \leq f(A_1^c[k]) + \ldots + f(A_m^c[k]).$$

Therefore,

$$f(A^c[k]) \leq m \max_i f(A_i^c[k]).$$

□

Lemma 35. $\max_i f(A_i^c[k]) \geq \frac{1}{k} f(A^c[k])$.

Proof. Let $f(A^c[k]) = f(\{u_1, \ldots u_k\})$. Using submodularity of f, we have

$$f(A^c[k]) \leq \sum_{i=1}^{k} f(u_i).$$

Thus, $f(A^c[k]) \leq kf(u^*)$ where $u^* = \arg\max_i f(u_i)$. Suppose that the element with highest marginal gain (i.e., u^*) is in V_j. Then the maximum value of f on V_j would be greater or equal to the marginal gain of u^*, i.e., $f(A^c_j[k]) \geq f(u^*)$ and since $f(\max_i f(A^c_i[k])) \geq f(A^c_j[k])$, we can conclude that

$$f(\max_i f(A^c_i[k])) \geq f(u^*) \geq \frac{1}{k} f(A^c[k]).$$

$\square$

Since $f(A^d[m,k]) \geq \max_i f(A^c_i[k])$; from Lemma 34 and 35 we have

$$f(A^d[m,k]) \geq \frac{1}{\min(m,k)} f(A^c[k]).$$

$\Leftarrow$ direction:
Let us consider a set of unbiased and independent Bernoulli random variables $X_{i,j}$ for $i \in \{1,\ldots,m\}$ and $j \in \{1,\ldots,k\}$, i.e., $\Pr(X_{i,j} = 1) = \Pr(X_{i,j} = 0) = 1/2$ and $(X_{i,j} \perp X_{i',j'})$ if $i \neq i'$ or $j \neq j'$. Let us also define $Y_i = (X_{i,1},\ldots,X_{i,k})$ for $i \in \{1,\ldots,m\}$. Now assume that $V_i = \{X_{i,1},\ldots,X_{i,k},Y_i\}, V = \bigcup_{i=1}^m V_i$ and $f(S) = H(S)$, where H is the entropy of the subset S of random variables. Note that H is a monotone submodular function. It is easy to see that $A^c_i[k] = \{X_{i,1},\ldots,X_{i,k}\}$ or $A^c_i[k] = Y_i$ as in both cases $H(A^c_i[k]) = k$. If we assume $A^c_i[k] = \{X_{i,1},\ldots,X_{i,k}\}$, then $B = \{X_{i,j} | 1 \leq i \leq m, 1 \leq j \leq k\}$. Hence, by selecting at most k elements from B, we have $H(A^d[m,k]) = k$. On the other hand, the set of k elements that maximizes the entropy is $\{Y_1,\ldots,Y_m\}$. Note that $H(Y_i) = k$ and $Y_i \perp Y_j$ for $i \neq j$. Hence, $H(A^c) = k \cdot m$ if $m \geq k$ or otherwise $H(A^c[k]) = k^2$.

Proof of Theorem 5

Let us first mention a slight generalization over the performance of the standard greedy algorithm. It follows easily from the argument in [NWF78a].

Lemma 36. *Let f be a non-negative submodular function, and let $A^{gc}[q]$ of cardinality q be the greedy selected set by the standard greedy algorithm. Then,*

$$f(A^{gc}[q]) \geq \left(1 - e^{-\frac{q}{k}}\right) f(A^c[k]).$$

By Lemma 36 we know that

$$f(A_i^{\text{gc}}[\kappa]) \geq (1 - \exp(-\kappa/k)) f(A_i^{\text{c}}[k]).$$

Now, let us define

$$\begin{aligned} B^{\text{gc}} &= \cup_{i=1}^{m} A_i^{\text{gc}}[\kappa], \\ A_{\max}^{\text{gc}}[\kappa] &= \max_i f(A_i^{\text{gc}}[\kappa]), \\ \tilde{A}[\kappa] &= \arg\max_{S \subseteq B^{\text{gc}} \& |S| \leq \kappa} f(S). \end{aligned}$$

Then by using Lemma 36 again, we obtain

$$\begin{aligned} f(A^{\text{gd}}[m,\kappa]) &\geq \max\left\{ f(A_{\max}^{\text{gc}}[\kappa]),\ (1 - \exp(-\kappa/k)) f(\tilde{A}[\kappa]) \right\} \\ &\geq \frac{(1 - \exp(-\kappa/k))}{\min(m,k)} f(A^{\text{c}}[k]). \end{aligned}$$

Proof of Proposition 7

Let K be a positive definite kernel matrix defined in section 3.1.1. If we replace a point $e_i \in S$ with another point $e_i' \in V \setminus S$, the corresponding row and column i in the modified kernel matrix K' will be changed. W.l.o.g assume that we replace the first element $e_1 \in S$ with another element $e_1' \in V \setminus S$, i.e., $\Delta K = K' - K$ has the following form with non-zero entries only on the first row and first column,

$$\Delta K \equiv K' - K \leq \begin{pmatrix} a_1 & a_2 & \cdots & a_k \\ a_2 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_k & 0 & \cdots & 0 \end{pmatrix}.$$

Note that kernel is Lipschitz continuous with constant $\mathcal{L}$, hence we have $|a_i| \leq \mathcal{L} d(e_1, e_1')$

for $1 \leq i \leq k$. Then the absolute value of the change in the objective function would be

$$\begin{aligned}
\left|f(S) - f(S')\right| &= \left|\frac{1}{2}\log\det(\mathbf{I}+K') - \frac{1}{2}\log\det(\mathbf{I}+K)\right| \\
&= \frac{1}{2}\left|\log\frac{\det(\mathbf{I}+K')}{\det(\mathbf{I}+K)}\right| \\
&= \frac{1}{2}\left|\log\frac{\det(\mathbf{I}+K+\Delta K)}{\det(\mathbf{I}+K)}\right| \\
&= \frac{1}{2}\left|\log[\det(\mathbf{I}+K+\Delta K).\det(\mathbf{I}+K)^{-1}]\right| \\
&= \frac{1}{2}\left|\log\det(\mathbf{I}+\Delta K(\mathbf{I}+K)^{-1})\right|. \qquad \text{(A.1.1)}
\end{aligned}$$

Note that since K is positive-definite, $\mathbf{I}+K$ is an invertible matrix. Furthermore, since ΔK and K are symmetric matrices they both have k real eigenvalues. Therefore, $(\mathbf{I}+K)^{-1}$ has k eigenvalues $\lambda_i = \frac{1}{1+\lambda'_i} \leq 1$, for $1 \leq i \leq k$, where $\lambda'_1 \cdots \lambda'_k$ are (non-negative) eigenvalues of kernel matrix K.

Now, we bound the maximum eigenvalues of ΔK and $\Delta K(\mathbf{I}+K)^{-1}$ respectively. Consider vectors $x, x' \in \mathbb{R}^n$, such that $||x||_2 = ||x'||_2 = 1$. We have,

$$\begin{aligned}
\left|x^T \Delta K\, x'\right| &= \left| \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{pmatrix}^T \begin{pmatrix} a_1 & a_2 & \cdots & a_k \\ a_2 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_k & 0 & \cdots & 0 \end{pmatrix} \begin{pmatrix} x'_1 \\ x'_2 \\ \vdots \\ x'_k \end{pmatrix} \right| \\
&= \left| \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{pmatrix}^T \begin{pmatrix} \sum_{i=1}^k a_i x'_i \\ a_2 x'_1 \\ \vdots \\ a_k x'_1 \end{pmatrix} \right| \\
&= \left| x_1 \sum_{i=1}^k a_i x'_i + x'_1 \sum_{i=2}^k a_i x_i \right| \\
&= |x_1|. \left| \sum_{i=1}^k a_i x'_i \right| + |x'_1|. \left| \sum_{i=2}^k a_i x_i \right| \\
&= |x_1|. \sum_{i=1}^k |a_i x'_i| + |x'_1|. \sum_{i=2}^k |a_i x_i| \\
&\leq 2k\mathcal{L}d(e_1, e'_1), \qquad \text{(A.1.2)}
\end{aligned}$$

where we used the following facts to derive the last inequality: 1) the Lipschitz continuity of the kernel gives us an upperbound on the values of $|a_i|$, i.e., $|a_i| \leq \mathcal{L}d(e_1, e'_1)$ for $1 \leq i \leq k$; and 2) since $||x||_2 = ||x'||_2 = 1$, the absolute value of the elements in vectors x and x' cannot be greater than 1, i.e., $|x_i| \leq 1$, $|x'_i| \leq 1$, for $1 \leq i \leq k$. Therefore,

$$\lambda_{\max}(\Delta K) = \max_{x:\ ||x||_2=1} |x^T \Delta K x| \leq 2k\mathcal{L}d(e_1, e'_1).$$

Now, let $v_1, \cdots v_k \in \mathbb{R}^n$ be the k eigenvectors of matrix $(\mathbf{I} + K)^{-1}$. Note that $\{v_1, \cdots v_k\}$ is an orthonormal system and thus for any $x \in \mathbb{R}^n$ we can write it as $x = \sum_{i=1}^k c_i v_i$, and we have $||x||_2^2 = \sum_{i=1}^k c_i^2$. In order to bound the largest eigenvalue of $\Delta K(\mathbf{I} + K)$, we write

$$\begin{aligned}
\left|x^T \Delta K \,(\mathbf{I}+K)^{-1} x\right| &= \left|x^T \Delta K \,(\mathbf{I}+K)^{-1} \sum_{i=1}^{k} c_i v_i\right| \\
&= \left|x^T \Delta K \,\sum_{i=1}^{k} \lambda_i c_i v_i\right| \\
&= \left|\left(\sum_{j=1}^{k} c_j v_j\right)^T \Delta K \left(\sum_{i=1}^{k} \lambda_i c_i v_i\right)\right| \\
&= \left|\sum_{i,j=1}^{k} \lambda_i c_i c_j v_j^T \Delta K v_i\right| \\
&\overset{(a)}{\leq} 2k\mathcal{L}d(e_1, e_1') \sum_{i,j=1}^{k} |c_i||c_j| \\
&= 2k\mathcal{L}d(e_1, e_1') \left(\sum_{i=1}^{k} |c_i|\right)^2,
\end{aligned}$$

where in (a) we used Eq. A.1.2 and the fact that $\lambda_i \leq 1$ for $1 \leq i \leq k$. Using Cauchy-Schwarz inequality

$$\left(\sum_{i=1}^{k} |c_i|\right)^2 \leq k \sum_{i=1}^{k} |c_i|^2$$

and the assumption $||x||_2 = 1$, we conclude

$$\begin{aligned}
\left|x^T \Delta K \,(\mathbf{I}+K)^{-1} x\right| &\leq 2k^2 \mathcal{L}d(e_1, e_1') \sum_{i=1}^{k} \left|c_i^2\right| \\
&\leq 2k^2 ||x||_2^2 \mathcal{L}d(e_1, e_1') \\
&\leq 2k^2 \mathcal{L}d(e_1, e_1').
\end{aligned}$$

Therefore,

$$\lambda_{\max}\left(\Delta K(\mathbf{I}+K)^{-1}\right) = \max_{x:\ ||x||_2=1} \left|x^T \Delta K \,(\mathbf{I}+K)^{-1} x\right| \leq 2k^2 \mathcal{L}d(e_1, e_1'). \tag{A.1.3}$$

Finally, we can write the determinant of a matrix as the product of its eigenvalues, i.e.

$$\det(\mathbf{I} + \Delta K(\mathbf{I}+K)^{-1}) \leq (1+2k^2\mathcal{L}d(e_1,e_1'))^k. \tag{A.1.4}$$

By substituting Eq. A.1.3 and Eq. A.1.4 into Eq. A.1.1 we obtain

$$\begin{aligned} |f(S)-f(S')| &\leq \frac{1}{2}\left|\log(1+2k^2\mathcal{L}d(e_1,e_1'))^k\right| \\ &\leq \frac{k}{2}\left|\log(1+2k^2\mathcal{L}d(e_1,e_1'))\right| \\ &\leq k^3\mathcal{L}d(e_1,e_1'), \end{aligned}$$

where in the last inequality we used $\log(1+x) \leq x$, for $x \geq 0$.

Replacing all the k points in set S with another set S' of the same size, we get

$$|f(S)-f(S')| \leq k^3\mathcal{L}\sum_{i=1}^{k} d(e_i,e_i').$$

Hence, the differential entropy of the Gaussian process is λ-Lipschitz with $\lambda = \mathcal{L}k^3$.

Proof of Proposition 8

Assume we have a set S of k exemplars, i.e., $S_0 = \{e_1, \cdots, e_k\}$, and each element of the dataset $v \in V$ is assigned to its closest exemplar. Now, if we replace set S with another set S' of the same size, the loss associated with every element $v \in V$ may be changed. W.l.o.g, assume we swap one exemplar at a time, i.e., in step $i, 1 \leq i \leq k$, we have $S_i = \{e_1', \cdots, e_i', e_{i+1}, \cdots, e_k\}$. Swapping the i^{th} exemplar $e_i \in S_{i-1}$ with another element $e_i' \in S'$, 4 cases may happen: 1) element v was not assigned to e_i before and doesn't get assigned to e_i', 2) element v was assigned to e_i before and gets assigned to e_i', 3) element v was not assigned to e_i before and gets assigned to e_i', 4) element v was assigned to e_i before and gets assigned to another exemplar $e_x \in S_i \setminus \{e_i'\}$. For any element $v \in V$, we look into the four cases and show that in each case

$$|l(e_i',v) - l(e_i,v)| \leq d(e_i,e_i')\,\alpha R^{\alpha-1}.$$

- Case 1: In this case, element v was assigned to another exemplar $e_x \in S_i \setminus e_i$ and the assignment doesn't change. Therefore, there is no change in the value of the loss function.

- Case 2: In this case, element v was assigned to e_i before and gets assigned to e'_i. let $a = d(e_i, v)$ and $b = d(e'_i, v)$. Then we can write

$$\begin{aligned}|l(e'_i, v) - l(e_i, v)| &= |a^\alpha - b^\alpha| \\ &= |(a-b)|(a^{\alpha-1} + a^{\alpha-2}b + \cdots + ab^{\alpha-2} + b^{\alpha-1}) \\ &\leq d(e_i, e'_i)\ \alpha R^{\alpha-1}, \end{aligned} \tag{A.1.5}$$

  where in the last step we used triangle inequality $|d(e'_t, v) - d(e_t, v)| \leq d(e_t, e'_t)$ and the fact that data points are in a ball of diameter R in the metric space.

- Case 3: In this case, v was assigned to another exemplar $e_x \in S_{i-1} \setminus \{e_i\}$ and gets assigned to e'_i, which implies that $|l(e'_i, v) - l(e_x, v)| \leq |l(e_i, v) - l(e'_i, v)|$, since otherwise e would have been assigned to e_t before.

- Case 4: In the last case, element v was assigned to e_i before and gets assigned to another exemplar $e_x \in S_i \setminus \{e'_i\}$. Thus, we have $|l(e_x, v) - l(e_i, v)| \leq |l(e'_i, v) - l(e_i, v)|$ since otherwise v would have been assigned to e_x before. Hence, in all four cases the following inequality holds:

$$|\min_{e \in S_{i-1}} l(e, v) - \min_{e \in S_i} l(e, v)| \leq |l(e'_i, v) - l(e_i, v)| \leq d(e_i, e'_i)\ \alpha R^{\alpha-1}.$$

By using Eq. A.1.5 and averaging over all elements $v \in V$, we have

$$\begin{aligned}|L(S_{i-1}) - L(S_i)| &= \frac{1}{|V|} \sum_{v \in V} |\min_{e \in S_{i-1}} l(e, v) - \min_{e \in S_i} l(e, v)| \\ &\leq \alpha R^{\alpha-1} d(e_i, e'_i).\end{aligned}$$

Thus, for any point e_0 that satisfies

$$\max_{v' \in V} l(v, v') \leq l(v, e_0), \quad \forall v \in V \setminus S,$$

we have $L(\{e_0 \cup S\}) = L(\{S\})$ and thus

$$\begin{aligned}|f(S_{i-1}) - f(S_i)| &= |L(\{e_0\}) - L(\{e_0 \cup S_{i-1}\}) - L(\{e_0\}) + L(\{e_0 \cup S_i\})| \\ &\leq \alpha R^{\alpha-1} d(e_i, e_i').\end{aligned}$$

Now, if we replace all the k points in set S with another set S' of the same size, we get

$$\begin{aligned}\left|f(S) - f(S')\right| &= \left|\sum_{i=1}^{k} f(S_{i-1}) - f(S_i)\right| \\ &= \sum_{i=1}^{k} |f(S_{i-1}) - f(S_i)| \\ &\leq \alpha R^{\alpha-1} \sum_{i=1}^{k} d(e_i, e_i').\end{aligned}$$

Therefore, for $l = d^\alpha$, the loss function is λ-Lipschitz with $\lambda = \alpha R^{\alpha-1}$.

Proof of Theorem 9

In the following, we say that sets S and S' are γ-close if $|f(S) - f(S')| \leq \gamma$. First, we need the following lemma.

Lemma 37. *If for each $e_i \in A^c[k], |N_\alpha(e_i)| \geq km \log(k/\delta^{1/m})$, and if V is partitioned into sets $V_1, V_2, \ldots V_m$, where each element is randomly assigned to one set with equal probabilities, then there is at least one partition with a subset $A_i^c[k]$ such that $\left|f(A^c[k]) - f(A_i^c[k])\right| \leq \lambda\alpha k$ with probability at least $(1-\delta)$.*

Proof. By the hypothesis, the α neighborhood of each element in $A^c[k]$ contains at least $km \log(k/\delta^{1/m})$ elements. For each $e_i \in A^c[k]$, let us take a set of $m \log(k/\delta^{1/m})$ elements from its α-neighborhood. These sets can be constructed to be mutually disjoint, since each α-neighborhood contains $m \log(k/\delta^{1/m})$ elements. We wish to show that at least one of the m partitions of V contains elements from α-neighborhoods of each element.

Each of the $m \log(k/\delta^{1/m})$ elements goes into a particular V_j with a probability $1/m$. The probability that a particular V_j does not contain an element α-close to $e_i \in A^c[k]$ is $\dfrac{\delta^{1/m}}{k}$. The probability that V_j does not contain elements α-close to one or more of the k elements is at most $\delta^{1/m}$ (by union bound). The probability that *each* $V_1, V_2, \ldots V_m$ does not contain elements from the α-neighborhood of one or more of the k elements is at most δ. Thus, with high probability of at least $(1-\delta)$, at least one of $V_1, V_2, \ldots V_m$ contains an $A^c_i[k]$ that is $\lambda\alpha k$-close to $A^c[k]$. $\square$

By lemma 37, for some V_i, $\left|f(A^c[k]) - f(A^c_i[k]|\right) \leq \lambda\alpha k$ with the given probability. Furthermore, $f(A^{gc}_i[\kappa]) \geq (1 - e^{-\kappa/k}) f(A^c_i[k])$ by Lemma 36. Therefore, the result follows using arguments analogous to the proof of Theorem 5.

Proof of Theorem 10

The following lemma says that in a sample drawn from distribution over an infinite dataset, a sufficiently large sample size guarantees a dense neighborhood near each element of $A^c[k]$ when the elements are from representative regions of the data.

Lemma 38. *A number of elements: $n \geq \dfrac{8km\log(k/\delta^{1/m})}{\beta g(\alpha)}$, where $\alpha \leq \alpha^*$, suffices to have at least $4km\log(k/\delta^{1/m})$ elements in the α-neighborhood of each $e_i \in A^c[k]$ with probability at least $(1-\delta)$, for small values of δ.*

Proof. The expected number of α-neighbors of an $e_i \in A^c[k]$, is $E[|N_\alpha(e_i)|] \geq 8km\log(k/\delta^{1/m})$. We now show that in a random set of samples, at least a half of this number of neighbors is realized with high probability near each element of $A^c[k]$.

This follows from a Chernoff bound:

$$P[|N_\alpha(e_i)| \leq 4km\log(k/\delta^{1/m})] \leq e^{-km\log(k/\delta^{1/m})} \leq (\delta^{1/m}/k)^{km}.$$

Therefore, the probability that some $e_i \in A^c[k]$ does not have a suitable sized neighborhood is at most $k(\delta^{1/m}/k)^{km}$. For $\delta \leq 1/k$, $k\delta^{km} \leq \delta^m$. Therefore, with probability at least $(1-\delta)$, the α-neighborhood of each element $e_i \in A^c[k]$ contains at least $4km\log(1/\delta)$ elements. $\square$

Lemma 39. *For $n \geq \dfrac{8km\log(k/\delta^{1/m})}{\beta g(\frac{\varepsilon}{\lambda k})}$, where $\frac{\varepsilon}{\lambda k} \leq \alpha^*$, if V is partitioned into sets $V_1, V_2, \ldots V_m$, where each element is randomly assigned to one set with equal probabilities, then for sufficiently small values of δ, there is at least one partition with a subset $A^c_i[k]$ such that $\left|f(A^c[k]) - f(A^c_i[k])\right| \leq \varepsilon$ with probability at least $(1-\delta)$.*

Proof. Follows directly by combining Lemma 38 and Lemma 37. The probability that some element does not have a sufficiently dense $\varepsilon/\lambda k$-neighborhood with $km\log(2k/\delta^{1/m})$ elements is at most $(\delta/2)$ for sufficiently small δ, and the probability that some partition does not contain elements from the one or more of the dense neighborhoods is at most $(\delta/2)$. Therefore, the result holds with probability at least $(1-\delta)$. □

By Lemma 39, there is at least one V_i such that $|f(A^c[k]) - f(A_i^c[k])| \leq \varepsilon$ with the given probability. And $f(A_i^{gd}[\kappa]) \geq (1-e^{-\kappa/k})f(A_i^c[k])$ using Lemma 36. The result follows using arguments analogous to the proof of Theorem 5.

Proof of Theorem 11

Note that each machine has on the average n/m elements. Let us define Π_i the event that $n/2m < |V_i| < 2n/m$. Then based on the Chernoff bound we know that $\Pr(\neg\Pi_i) \leq 2\exp(-n/8m)$. Let us also define $\xi_i(S)$ the event that $|f_{V_i}(S) - f(S)| < \epsilon$, for some fixed $\epsilon < 1$ and a fixed set S with $|S| \leq k$. Note that $\xi_i(S)$ denotes the event that the empirical mean is close to the true mean. Based on the Hoeffding inequality (without replacement) we have $\Pr(\neq \xi_i S| \leq 2\exp(-2n\epsilon^2/m)$. Hence,

$$\Pr(\xi_i(S) \wedge \Pi_i) \geq 1 - 2\exp(-2n\epsilon^2/m) - 2\exp(-n/8m).$$

Let ξ_i be an event that $|f_{V_i}(S) - f(S)| < \epsilon$, for any S such that $|S| \leq \kappa$. Note that there are at most n^κ sets of size at most κ. Hence,

$$\Pr(\xi_i \wedge \Pi_i) \geq 1 - 2n^\kappa(\exp(-2n\epsilon^2/m) - \exp(-n/8m)). \tag{A.1.6}$$

As a result, for $\epsilon < 1/4$ we have

$$\Pr(\xi_i \wedge \Pi_i) \geq 1 - 4n^\kappa \exp(-2n\epsilon^2/m).$$

Since there are m machines, by the union bound we can conclude that

$$\Pr((\xi_i \wedge \Pi_i) \text{ on all machines}) \geq 1 - 4mn^\kappa \exp(-2n\epsilon^2/m).$$

The above calculation implies that we need to choose $\delta \geq 4mn^\kappa \exp(-2n\epsilon^2/m)$. Let n_0 be chosen in a way that for any $n \geq n_0$ we have $\ln(n)/n \leq \epsilon^2/(mk)$. Then, we need to choose n as follows:

$$n = \max\left(n_0, \frac{m\log(\delta/4m)}{\epsilon^2}\right).$$

Hence for the above choice of n, there is at least one V_i such that $\left|f(A^c[k]) - f(A_i^c[\kappa])\right| \leq \varepsilon$ with probability $1-\delta$. Hence the solution is ϵ away from the optimum solution with probability $1-\delta$. Now if we confine the evaluation of $f(A_i^c)$ to data point only in machine i then under the assumption of Theorem 10 we lose another ϵ. Formally, the result at this point simply follows by combining Theorem 5 and Theorem 10.

Proof of Theorem 13

The proof is similar to the proof of Theorem 4 and Theorem 5 and follows from the following lemmas.

Lemma 40. $\max_i f(A_i^c[\zeta]) \geq \frac{1}{m} f(A^c[\zeta])$.

Proof. Let B_i be the elements in V_i that are contained in the optimal solution, $B_i = A^c[\zeta] \cap V_i$. Since $A^c[\zeta] \in \zeta$ and ζ is a set of hereditary constraints, we must have $B_i \in \zeta$ as well. Using submodularity of f and by the same argument as in the proof of Lemma 34, we have

$$\begin{aligned} f(A^c[\zeta]) = f(B_1 \cup \cdots \cup B_m) &= f(B_1) + f(B_2|B_1) + \cdots + f(B_m|B_{m-1}, \cdots, B_1) \\ &\leq f(B_1) + \cdots + f(B_m). \end{aligned}$$

Since $f(A_i^c[\zeta]) \geq f(B_i)$ we get

$$f(A^c[\zeta]) \leq f(A_1^c[\zeta]) + \cdots + f(A_m^c[\zeta]) \leq m \max_i f(A_i^c[\zeta]).$$

□

Lemma 41. $\max_i f(A_i^c[\zeta]) \geq \frac{1}{k} f(A^c[\zeta])$.

Proof. The proof follows the outline of the proof of Lemma 35.

Let $f(A^{\mathrm{c}}[\zeta]) = f(\{u_1, \cdots, u_{\rho([\zeta])}\})$. Since $A^{\mathrm{c}}[\zeta] \in \zeta$ and ζ is a set of hereditary constraints, we have $u_i \in \zeta$. Using submodularity of f, we have

$$f(A^{\mathrm{c}}[\zeta]) \leq \sum_{i=1}^{\rho([\zeta])} f(u_i) \leq \rho([\zeta]) f(u^*).$$

where $u^* = \arg\max_i f(u_i)$. Suppose that $u^* \in V_j$, we get

$$f(\max_i f(A_i^{\mathrm{c}}[\zeta])) \geq f(A_j^{\mathrm{c}}[\zeta]) \geq f(u^*) \geq \frac{1}{\rho([\zeta])} f(A^{\mathrm{c}}[\zeta]).$$

$\square$

Since $f(A^{\mathrm{d}}[m, \rho([\zeta])]) \geq \max_i f(A_i^{\mathrm{c}}[\zeta])$; from Lemma 41 and 40 we have

$$f(A^{\mathrm{d}}[m, \rho([\zeta])]) \geq \frac{1}{\min(m, \rho([\zeta]))} f(A^{\mathrm{c}}[\zeta]). \tag{A.1.7}$$

For the black box algorithm X with a τ-approximation guarantee, we have

$$f(A_i^X[\zeta]) \geq \tau f(A_i^{\mathrm{c}}[\zeta]).$$

Now, we generalize the definitions used in the proof of Theorem 5

$$\begin{aligned} B^{\mathrm{gc}} &= \cup_{i=1}^m A_i^{\mathrm{gc}}[\zeta], \\ A_{\max}^{\mathrm{gc}}[\zeta] &= \max_i f(A_i^{\mathrm{gc}}[\zeta]), \\ \tilde{A}[\zeta] &= \arg\max_{S \subseteq B^{\mathrm{gc}} \& |S| \leq \rho([\zeta])} f(S). \end{aligned}$$

Then using Eq. A.1.7 again, we obtain

$$\begin{aligned} f(A^{\mathrm{gd}}[m, \zeta]) &\geq \max\left\{f(A_{\max}^{\mathrm{gc}}[\zeta]),\ \tau f(\tilde{A}[\zeta])\right\} \\ &\geq \frac{\tau}{\min(m, \rho([\zeta]))} f(A^{\mathrm{c}}[\zeta]). \end{aligned}$$

Note that since we do not use monotonicity of the submodular function in any of the proofs, the results hold in general for constrained maximization of any non-negative submodular function.

Proof of Theorem 14

Lemma 42. *If for each $e_i \in A^X[\zeta]$, $|N_\alpha(e_i)| \geq \rho([\zeta])m \log(\rho([\zeta])/\delta^{1/m})$, and if V is partitioned into sets $V_1, V_2, \ldots V_m$, where each element is randomly assigned to one set with equal probabilities, then there is at least one partition with a subset $A_i^X[\zeta] \in \zeta$ such that $\left|f(A^c[\zeta]) - f(A_i^X[\zeta])\right| \leq \lambda\alpha\rho([\zeta])$ with probability at least $(1-\delta)$.*

The proof is similar to the proof of Lemma 37 by taking disjoint sets of size $\frac{m \log(\rho([\zeta]))}{\delta^{1/m}}$ in an α-neighborhood of each $e_i \in A^c[\zeta]$ and showing that with high probability, at least one of the m partitions of V contains elements from α-neighborhoods of each element in the optimal solution. Note that now the size of the optimal solution is at most $\rho([\zeta])$. Since ζ is locally replaceable with parameter α, as elements of $A^c[\zeta]$ gets replaced by nearby elements in their α-neighborhood, the resulting set is also a feasible solution.

By Lemma 42, for some V_i, $\left|f(A^c[\zeta]) - f(A_i^X[\zeta])\right| \leq \lambda\alpha\rho([\zeta])$ with the given probability. On the other hand, for the black box algorithm X, we have $f(A_i^X[\zeta]) \geq \tau f(A_i^c[\zeta])$. Therefore, the result follows using arguments analogous to the proof of Theorem 13.

Proof of Theorem 15

We use the following Lemmas to show that in a sample drawn from a ddistribution over an infinite dataset, a sufficiently large sample size guarantees a dense neighborhood near each element of the optimal solution.

Lemma 43. *A number of elements: $n \geq \dfrac{8\rho([\zeta])m\log(\rho([\zeta])/\delta^{1/m})}{\beta g(\alpha)}$, where $\alpha \leq \alpha^*$, suffices to have at least $4\rho([\zeta])m\log(\rho([\zeta])/\delta^{1/m})$ elements in the α-neighborhood of each $e_i \in A^c[\zeta]$ with probability at least $(1-\delta)$, for small values of δ.*

Lemma 44. *For $n \geq \dfrac{8\rho([\zeta])m\log(\rho([\zeta])/\delta^{1/m})}{\beta g(\frac{\varepsilon}{\lambda\rho([\zeta])})}$, where $\frac{\varepsilon}{\lambda\rho([\zeta])} \leq \alpha^*$, if V is partitioned into sets $V_1, V_2, \ldots V_m$, where each element is randomly assigned to one set with equal probabilities, then for sufficiently small values of δ, there is at least one partition with a subset $A_i^c[\zeta]$ such that $\left|f(A^c[\zeta]) - f(A_i^c[\zeta])\right| \leq \varepsilon$ with probability at least $(1-\delta)$.*

The proofs follows the same arguments as in the proof of Lemma 38 and 39. Recall that, by assumption ζ is locally replaceable with parameter α. Hence, for $\varepsilon \leq \alpha\lambda\rho([\zeta])$, any set ε-close to the optimal solution is also a feasible solution.

By Lemma 44, there is at least one V_i such that $\left|f(A^c[\zeta]) - f(A_i^c[\zeta])\right| \leq \varepsilon$ with the given probability. Furthermore, for the black box algorithm X, we have $f(A_i^{gd}[\zeta]) \geq \tau f(A_i^c[\zeta])$. Thus the result follows using arguments analogous to the proof of Theorem 13.

Proof of Theorem 16

Again the proof follows the same line of reasoning as the proof of Theorem 11, except that for a constraint set ζ with $\rho([\zeta]) = \max_{S\in\zeta} |S|$, there are at most $n^{\rho([\zeta])}$ feasible solutions. Using the same definitions for Π_i and $\mathcal{E}_i$ as in the proof of Theorem 11, instead of Eq. A.1.6 we get

$$\Pr(\xi_i \wedge \Pi_i) \geq 1 - 2n^{\rho([\zeta])}(\exp(-2n\epsilon^2/m) - \exp(-n/8m)).$$

As a result, for $\epsilon < 1/4$ and using union bound we conclude that

$$\Pr((\xi_i \wedge \Pi_i) \text{ on all machines}) \geq 1 - 4mn^{\rho([\zeta])}\exp(-2n\epsilon^2/m).$$

which implies that we need to choose $\delta \geq 4mn^{\rho}([\zeta])\exp(-2n\epsilon^2/m)$. Now if n_0 be chosen in a way that for any $n \geq n_0$ we have $\ln(n)/n \leq \epsilon^2/(mk)$, we get $n \geq \max(n_0, m\log(\delta/4m)/\epsilon^2)$.

Bearing in mind that ζ is locally replaceable, there is at least one V_i such that the solution $A_i^c[\zeta]$ is feasible and ϵ away from the optimum solution with probability $1-\delta$. Now under the assumption of Theorem 15, if we evaluate $f(A_i^c)$ only on machine i, then we lose another ϵ. Now by combining Theorem 13 and Theorem 15 we get the desired result.

A.2 Proofs from Chapter 6

This section presents the complete proofs of theorems presented in the article.

Proof of Theorem 17

This theorem is a special case of theorem 18 with $\alpha = 1$.

Proof of Theorem 18

We first prove a lemma which will be used in the proof of the theorem.

Lemma 45. *Let ℓ^* be the final value of ℓ at the end of the* DisCover *Algorithm. Then,*

$$\ell^* < 2\alpha k.$$

Proof. We consider it in two cases. In first case $\ell^* < \alpha k$ in which case the lemma is trivially true. Else at some iteration of DisCover we have that $\alpha k \le \ell < 2\alpha k$. We then show that for such a value of ℓ step 6 always holds and hence ℓ is never incremented, proving the lemma. For the rest of the proof consider such a value of ℓ.

Let $A^{\text{gd}}[m, \ell]$ be the set returned by GreeDi when requested to return a solution of size ℓ when the current solution is $A^{\text{dc}}[m]$. Let S_p^* be a set of size p which maximizes $f(S_p^* \cup A^{\text{dc}}[m])$. Let $S_k^* = \{e_1, e_2, \dots, e_k\}$ be the elements when then are sorted according to their marginal contributions when they are added one by one to $A^{\text{dc}}[m]$. Let $S_k^*(l) = \{e_1, e_2, \dots, e_l\}$. By definition of sorting according to decreasing marginal values we have the following inequality

$$\begin{aligned} &\forall p, f(S_k^*(p+1) \cup A^{\text{dc}}[m]) - f(S_k^*(p) \cup A^{\text{dc}}[m]) \\ &\geq f(S_k^*(p+2) \cup A^{\text{dc}}[m]) - f(S_k^*(p+1) \cup A^{\text{dc}}[m]) \end{aligned} \tag{A.2.1}$$

Now we have the following set of inequalities.

$$\begin{aligned}
& f(A^{\mathrm{gd}}[m,\ell] \cup A^{\mathrm{dc}}[m]) - f(A^{\mathrm{dc}}[m]) && \\
& \geq \lambda(f(S^*_\ell \cup A^{\mathrm{dc}}[m]) - f(A^{\mathrm{dc}}[m])) && \text{By definition of } \lambda \\
& \geq \lambda(f(S^*_{\alpha k} \cup A^{\mathrm{dc}}[m]) - f(A^{\mathrm{dc}}[m])) && \text{By monotonicity} \\
& \geq \lambda(f(S^*_k(\alpha k) \cup A^{\mathrm{dc}}[m]) - f(A^{\mathrm{dc}}[m])) && \text{By definition of } S^*_{\alpha k} \\
& = \lambda \sum_{i=0}^{\alpha k-1} (f(S^*_k(i+1) \cup A^{\mathrm{dc}}[m]) - f(S^*_k(i) \cup A^{\mathrm{dc}}[m])) && \\
& \geq \lambda\alpha \sum_{i=0}^{k-1} (f(S^*_k(i+1) \cup A^{\mathrm{dc}}[m]) - f(S^*_k(i) \cup A^{\mathrm{dc}}[m])) && \text{Elements sorted by marginals} \\
& = \lambda\alpha(f(S^*_k \cup A^{\mathrm{dc}}[m]) - f(A^{\mathrm{dc}}[m])) && \\
& \geq \lambda\alpha(f(S^*_k) - f(A^{\mathrm{dc}}[m])) && \text{By monotonicity} \\
& \geq \lambda\alpha(Q - f(A^{\mathrm{dc}}[m])) && \text{By monotonicity}
\end{aligned}$$

The above set of inequalities prove that step 6 always holds and hence ℓ is never incremented once it reaches a value such that $\alpha k \leq \ell < 2\alpha k$, proving the lemma. □

Now armed with Lemma 45 we complete the proof of theorem 18. Consider step 6 of DisCover. This step can get executed at most $\log(2\alpha k)$ times because of lemma 45. Let T_i be the solution set S returned when step 6 of DisCover is satisfied for the i^{th} time. Let $T^* = T_z$ be the solution before condition on step 6 is satisfied for the last time and condition on step 6 be satisfied $z+1$ times. Then by the inequality in step 6 we have the inequality below.

$$\begin{aligned}
f(T_{i+1}) - f(T_i) \geq &\quad \alpha\lambda(Q - f(T_i)) && \\
\Rightarrow Q - f(T_{i+1}) \leq &\quad (1-\alpha\lambda)(Q - f(T_i)) && \\
\Rightarrow Q - f(T^*) \leq &\quad (1-\alpha\lambda)^z Q \quad < 1 && \\
\Rightarrow (1-\alpha\lambda)^z < &\quad 1/Q && \\
\Rightarrow z\log(1-\alpha\lambda) < &\quad -\log(Q) && \\
\Rightarrow z\alpha\lambda > &\quad \log(Q) && \text{Since } \log(1-x) \leq -x \\
\Rightarrow z > &\quad \tfrac{1}{\alpha\lambda}\log(Q) &&
\end{aligned}$$

Hence the total number of rounds of the algorithm is $\log(\alpha k) + \frac{1}{\alpha\lambda}\log(Q) + 1$. Additionally since we pick at most $2\alpha k$ elements in each round satisfying step 6 of DisCover we get the solution is of size at most $2\alpha k \cdot (\frac{1}{\alpha\lambda}\log(Q)+1) = 2\alpha k + \frac{2k}{\lambda}\log(Q)$.

Proof of Theorem 19

Let $\mathcal{G}(T,k)$ be the set returned by running greedy to choose k elements on T.

Fact 1. *Let Ω be any ground set and $S = \mathcal{G}(\Omega,k)$ and O be any set such that $|O| \le k$. Then we have the following inequality*

$$f(S) \ge \frac{k}{|O|}\left(f(S \cup O) - f(S)\right) \tag{A.2.2}$$

Proof. Let $S = \{e_1, e_2, \ldots, e_k\}$ sorted by the order in which greedy algorithm picks the elements. Let for each i, $S_i = \{e_1, e_2, \ldots, e_i\}$. Let $O = \{o_1, o_2, \ldots o_{|O|}\}$ and $O_i = \{o_1, o_2, \ldots, o_i\}$. Follows easily from the following set of equations

$$\begin{aligned}
f(S) &= \sum_{i=1}^{k} f(S_i) - f(S_{i-1}) \\
&\ge \sum_{i=1}^{k} \frac{1}{|O|}\sum_{j=1}^{|O|} f(S_{i=1} \cup \{o_j\}) - f(S_{i-1}) && \text{By property of greedy algorithm} \\
&\ge \sum_{i=1}^{k} \frac{1}{|O|}\sum_{j=1}^{|O|} f(S_{i=1} \cup O_j) - f(S_{i-1} \cup O_{j-1}) && \text{By submodularity} \\
&= \sum_{i=1}^{k} \frac{1}{|O|}(f(S_{i=1} \cup O) - f(S_{i-1})) \\
&\ge \sum_{i=1}^{k} \frac{1}{|O|}(f(S \cup O) - f(S)) && \text{By submodularity} \\
&= \frac{k}{|O|}(f(S \cup O) - f(S))
\end{aligned}$$

□

Now we complete proof of theorem 19. Let V_i be the set of elements on machine i. Let $S_i = \mathcal{G}(V_i, k)$. Then we will show a set $S \subseteq \cup_{i=1}^{m} S_i$ such that $|S| \le k$ and $f(S) \ge \frac{1}{18\sqrt{\min(k,m)}} f(\mathsf{OPT})$. Then the theorem follows because $\mathcal{G}(\cup_{i=1}^{m} S_i, k)$ produces at least a $1 - 1/e$ approximation to S.

Let $S_1, S_2, \ldots, S_q$ be the sets such that $S_i \cap \mathsf{OPT} \ne \emptyset$. Then note that $q \le \min(k,m)$. We will restrict the analysis to these sets. Let $\mathsf{OPT}_i = V_i \cap \mathsf{OPT}$. We construct the set S by iteratively processing each S_i and adding elements S. Let L_i be the set S after we process S_i. We start with $L_0 = \emptyset$. Let us say we have processed till S_{j-1}, we will show what happens when we process S_j.

Let $S_j = \{s_j^1, s_j^2, \dots, s_j^k\}$ in the order they are taken by the greedy algorithm. Let S_j^p be the set of first p elements in S_j. Then there are three cases.

1. Assume for each $p \leq k$ the following inequality is satisfied.

$$f(L_{j-1} \cup S_j^p \cup \mathsf{OPT}_j) - f(L_{j-1} \cup S_j^p) \geq \frac{1}{2}\left(f(L_{j-1} \cup \mathsf{OPT}_j) - f(L_{j-1})\right) \tag{A.2.3}$$

   Then choose $|\mathsf{OPT}_j|$ elements from S_j greedily and add them to L_{j-1}. Then we prove a lower bound on the improvement.

$$\begin{aligned}
&f(L_j) - f(L_{j-1}) && \text{(A.2.4)}\\
&\geq \frac{|\mathsf{OPT}_j|}{k}\left(f(L_j \cup S_j) - f(L_j)\right) && \text{Because of greedy algorithm}\\
&\geq \frac{|\mathsf{OPT}_j|}{k} f(S_j) - \frac{|\mathsf{OPT}_j|}{k} f(L_j) && \text{Because of monotonicity}\\
&\geq f(S_j \cup \mathsf{OPT}_j) - f(S_j) - \frac{|\mathsf{OPT}_j|}{k} f(L_j) && \text{Because of fact 1}\\
&\geq f(L_{j-1} \cup S_j \cup \mathsf{OPT}_j) - f(L_{j-1} \cup S_j) - \frac{|\mathsf{OPT}_j|}{k} f(L_j) && \text{Because of submodularity}\\
&\geq \frac{1}{2}\left(f(L_{j-1} \cup \mathsf{OPT}_j) - f(L_{j-1})\right) - \frac{|\mathsf{OPT}_j|}{k} f(L_j) && \text{Because of assumption A.2.3}\\
&\geq \frac{1}{2}\left(f(L_l \cup \mathsf{OPT}_j) - f(L_l)\right) - \frac{|\mathsf{OPT}_j|}{k} f(L_j) && \text{by submodularity}\\
&\geq \frac{1}{2}\left(f(L_l \cup \mathsf{OPT}_j) - f(L_l)\right) - \frac{|\mathsf{OPT}_j|}{k} f(L_l) && \text{by monotonicity}\\
& && \text{(A.2.5)}
\end{aligned}$$

2. The next two cases are folded into this case. Let $c(j)$ be the minimum index such that the following inequality is satisfied.

$$f(L_{j-1} \cup S_j^{c(j)} \cup \mathsf{OPT}_j) - f(L_{j-1} \cup S_j^{c(j)}) \leq \frac{1}{2}\left(f(L_{j-1} \cup \mathsf{OPT}_j) - f(L_{j-1})\right) \tag{A.2.6}$$

Then we get the following sequence of inequalities

$$
\begin{aligned}
& f(L_{j-1} \cup S_j^{c(j)}) - f(L_{j-1}) && \text{(A.2.7)}\\
&= f(L_{j-1} \cup S_j^{c(j)} \cup \mathsf{OPT}_j) - f(L_{j-1}) && \text{(A.2.8)}\\
&- \left(f(L_{j-1} \cup S_j^{c(j)} \cup \mathsf{OPT}_j) - f(L_{j-1} \cup S_j^{c(j)}) \right) \\
&\geq f(L_{j-1} \cup \mathsf{OPT}_j) - f(L_{j-1}) - \left(f(L_{j-1} \cup S_j^{c(j)} \cup \mathsf{OPT}_j) - f(L_{j-1} \cup S_j^{c(j)}) \right) \\
& \qquad \text{Because of monotonicity} \\
&\geq f(L_{j-1} \cup \mathsf{OPT}_j) - f(L_{j-1}) - \frac{1}{2} \left(f(L_{j-1} \cup \mathsf{OPT}_j) - f(L_{j-1}) \right) \\
& \qquad \text{From equation A.2.6} \\
&= \frac{1}{2} \left(f(L_{j-1} \cup \mathsf{OPT}_j) - f(L_{j-1}) \right) && \text{(A.2.9)}
\end{aligned}
$$

3. Consider when $c(j) > 1$. Then note that for any $p < c(j)$ we have that $f(L_{j-1} \cup S_j^p \cup \mathsf{OPT}_j) - f(L_{-1}j \cup S_j^p) \geq \frac{1}{2}\left(f(L_{j-1} \cup \mathsf{OPT}_j) - f(L_{j-1}) \right)$. Hence we also get the following inequality

$$
\begin{aligned}
f(S_j^{p+1}) - f(S_j^p) &\geq \frac{1}{|\mathsf{OPT}_j|} \sum_{e \in \mathsf{OPT}_j} f(S_j^p \cup \{e\}) - f(S_j^p) && \text{By choice of greedy}\\
&\geq \frac{1}{|\mathsf{OPT}_j|} \left(f(S_j^p \cup \mathsf{OPT}_j) - f(S_j^p) \right) && \text{By submodularity}\\
&\geq \frac{1}{|\mathsf{OPT}_j|} \left(f(L_{j-1} \cup S_j^p \cup \mathsf{OPT}_j) - f(L_{j-1} \cup S_j^p) \right) \\
&\geq \left(\frac{1}{2|\mathsf{OPT}_j|} \right) \left(f(L_{j-1} \cup \mathsf{OPT}_j) - f(L_{j-1}) \right) && \text{By assumption}
\end{aligned}
\tag{A.2.10}
$$

Summing above inequalities, we get

$$
f(S_j^{c(j)}) \geq \left(\frac{c(j)}{2|\mathsf{OPT}_j|} \right) \left(f(L_{j-1} \cup \mathsf{OPT}_j) - f(L_{j-1}) \right) \tag{A.2.11}
$$

Then we have three different situations

(a) $c(j) <= \sqrt{q} \cdot |\mathsf{OPT}_j|$ and $c(j) > \mathsf{OPT}_j$ then add $|\mathsf{OPT}_j|$ elements to L_{j-1}

greedily from S_j. Then we get

$$\begin{aligned}
& f(L_j) - f(L_{j-1}) && \text{(A.2.12)}\\
&\geq \frac{|\mathsf{OPT}_j|}{c(j)} \left(f(L_j \cup S_j^{c(j)}) - f(L_j)\right) && \text{from property of greedy algorithm}\\
&\geq \frac{|\mathsf{OPT}_j|}{2c(j)} \left(f(L_{j-1} \cup \mathsf{OPT}_j) - f(L_{j-1})\right) && \text{from equation A.2.9}\\
&\geq \frac{1}{2\sqrt{q}} \left(f(L_{j-1} \cup \mathsf{OPT}_j) - f(L_{j-1})\right) && \text{because } c(j) <= \sqrt{q} \cdot |\mathsf{OPT}_j|\\
&\geq \frac{1}{2\sqrt{q}} \left(f(L_q \cup \mathsf{OPT}_j) - f(L_q)\right) && \text{from submodularity} \quad \text{(A.2.13)}
\end{aligned}$$

(b) $c(j) > \sqrt{q} \cdot |\mathsf{OPT}_j|$ then add $|\mathsf{OPT}_j|$ elements to L_{j-1} greedily from S_j. Then we get

$$\begin{aligned}
& f(L_j) - f(L_{j-1}) && \text{(A.2.14)}\\
&\geq \frac{|\mathsf{OPT}_j|}{c(j)} \left(f(L_j \cup S_j^{c(j)}) - f(L_j)\right) && \text{from property of greedy algorithm}\\
&\geq \frac{|\mathsf{OPT}_j|}{c(j)} \left(f(S_j^{c(j)}) - f(L_j)\right) && \text{from monotonicity}\\
&\geq \frac{1}{2} \left(f(L_{j-1} \cup \mathsf{OPT}_j) - f(L_{j-1})\right) - \frac{|\mathsf{OPT}_j|}{c(j)} f(L_j) && \text{from equation A.2.10}\\
&\geq \frac{1}{2} \left(f(L_{j-1} \cup \mathsf{OPT}_j) - f(L_{j-1})\right) - \frac{1}{\sqrt{q}} f(L_j) && \text{because } c(j) > \sqrt{q} \cdot |\mathsf{OPT}_j|\\
&\geq \frac{1}{2} \left(f(L_q \cup \mathsf{OPT}_j) - f(L_q)\right) - \frac{1}{\sqrt{q}} f(L_j) && \text{from submodularity}\\
&\geq \frac{1}{2} \left(f(L_l \cup \mathsf{OPT}_j) - f(L_l)\right) - \frac{1}{\sqrt{q}} f(L_l) && \text{from monotonicity}\\
& && \text{(A.2.15)}
\end{aligned}$$

(c) If $c(j) <= |\mathsf{OPT}_j|$ then $L_j = L_{j-1} \cup S_j^{c(j)}$ and we get $f(L_j) - f(L_{j-1}) \geq \frac{1}{2}\left(f(L_{j-1} \cup \mathsf{OPT}_j) - f(L_{j-1})\right)$ from equation A.2.9

We complete the proof by considering three different cases and proving the theorem in each of the three cases. For simplicity let the indices which satisfy condition one be I_1, condition 2a) (or condition 2c) be I_{2a} and condition 2b) be I_{2b}. Let $\mathsf{OPT}' = \cup_{i \in I_1} \mathsf{OPT}_i$, $\mathsf{OPT}'' = \cup_{i \in I_{2a}} \mathsf{OPT}_i$ and $\mathsf{OPT}''' = \cup_{i \in I_{2b}} \mathsf{OPT}_i$. Then by simply submodularity we know that $\max(f(\mathsf{OPT}'), f(\mathsf{OPT}''), f(\mathsf{OPT}''')) \geq f(\mathsf{OPT})/3$. We deal with each case separately.

- Case 1 when $f(\mathsf{OPT}') \geq f(\mathsf{OPT})/3$

$$\begin{aligned}
f(L_l) - f(\emptyset) &\geq \sum_{i \in I_1} f(L_i) - f(L_{i-1}) \\
&\geq \sum_{i \in I_1} \left(\frac{1}{2} \left(f(L_l \cup \mathsf{OPT}_j) - f(L_l) \right) - \frac{|\mathsf{OPT}_j|}{k} f(L_l) \right) \\
\Rightarrow 2f(L_l) &\geq \sum_{i \in I_1} \frac{1}{2} \left(f(L_l \cup \mathsf{OPT}_j) - f(L_l) \right) && \text{Rearranging terms} \\
&\geq \frac{1}{2} \left(f(L_l \cup \mathsf{OPT}') - f(L_l) \right) && \text{By submodularity} \\
&\geq \frac{1}{2} \left(f(\mathsf{OPT}') - f(L_l) \right) && \text{By monotonicity} \\
\Rightarrow f(L_l) &\geq \frac{1}{5} f(\mathsf{OPT}') && \text{Rearranging terms} \\
&\geq \frac{1}{15} f(\mathsf{OPT}) && \text{By assumption} \qquad \text{(A.2.16)}
\end{aligned}$$

- Case 2 when $f(\mathsf{OPT}'') \geq f(\mathsf{OPT})/3$

$$\begin{aligned}
f(L_l) - f(\emptyset) &\geq \sum_{i \in I_{2a}} f(L_i) - f(L_{i-1}) \\
&\geq \sum_{i \in I_{2a}} \frac{1}{2\sqrt{q}} \left(f(L_l \cup \mathsf{OPT}_j) - f(L_l) \right) \\
&\geq \frac{1}{2\sqrt{q}} \left(f(L_l \cup \mathsf{OPT}'') - f(L_l) \right) && \text{By submodularity} \\
&\geq \frac{1}{2\sqrt{q}} O \left(f(\mathsf{OPT}'') - f(L_l) \right) && \text{By monotonicity} \\
\Rightarrow f(L_l) &\geq \frac{1}{4\sqrt{q}} f(\mathsf{OPT}'') && \text{Rearranging terms} \\
&\geq \frac{1}{12\sqrt{q}} f(\mathsf{OPT}) && \text{By assumption} \qquad \text{(A.2.17)}
\end{aligned}$$

- Case 3 when $f(\mathsf{OPT}''') \geq f(\mathsf{OPT})/3$

$$\begin{aligned}
f(L_l) - f(\emptyset) &\geq \sum_{i\in I_{2b}} f(L_i) - f(L_{i-1}) \\
&\geq \sum_{i\in I_{2b}} \left(\frac{1}{2}\left(f(L_l \cup \mathsf{OPT}_j) - f(L_l)\right) - \frac{1}{\sqrt{q}} f(L_l)\right) \\
\Rightarrow (\sqrt{q}+1) f(L_l) &\geq \sum_{I_{2b}} \frac{1}{2}\left(f(L_l \cup \mathsf{OPT}_j) - f(L_l)\right) && \text{Rearranging terms} \\
&\geq \frac{1}{2}\left(f(L_l \cup \mathsf{OPT}''') - f(L_l)\right) && \text{By submodularity} \\
&\geq \frac{1}{2}\left(f(\mathsf{OPT}''') - f(L_l)\right) && \text{By monotonicity} \\
\Rightarrow f(L_l) &\geq \frac{1}{2(\sqrt{q}+2)} f(\mathsf{OPT}''') && \text{Rearranging terms} \\
&\geq \frac{1}{6(\sqrt{q}+2)} f(\mathsf{OPT}) && \text{By assumption} \\
&\geq \frac{1}{18\sqrt{q}} f(\mathsf{OPT}) && \text{(A.2.18)}
\end{aligned}$$

Remember that $q = \min(k, m)$ which completes the proof.

A.3 Proofs from Chapter 7

Before proving Theorem 21, we need to prove a few Lemmas to upper bound the size of final solution and the number of rounds separately.

Lemma 46. FastCover *returns a solution S with at most $|OPT| \ln(L)/(1-\epsilon)$ items and $f(S) \geq L$.*

Proof. We remind that an epoch ends when the if condition of line 14 holds, and therefore we update τ. We prove that at this point all items with marginal value at least τ have been added to S. Therefore the marginal value of every item to S is less than τ at the end of an epoch. Since $Full_i = False$ for every i, all items with marginal value at least τ are in selected sets $\{S_i\}_{i=1}^m$. In lines $10-12$, FastCover makes sure that every item in $\{S_i\}_{i=1}^m$ with marginal value at least τ is added to S. So at the end of each epoch all marginal values are less than τ. We should also note that by submodularity the marginal values to set S can only decrease, and τ is unchanged during an epoch, therefore all marginal values are still less than τ at the end of the epoch.

We now prove that $f(S) \geq L$. The algorithm terminates either at the Break operation of lines 13 in which $f(S) \geq L$ or line 16. At line 16, we are at the end of an epoch, and the else condition holds only if $\tau = 1$. Since all marginal values are less than $\tau = 1$ in this stage, and f only takes integer values, we conclude that all marginal values should be equal to zero. Using submodularity, we conclude that $f(S)$ in this case is equal to $f(\mathbb{V}) \geq L$ because $f(\mathbb{V}) - f(S) \leq \sum_{x \in \mathbb{V}} \Delta(x|S) = 0$ where $\Delta(x|S)$ is $f(S \cup \{x\}) - f(S)$.

We are ready to upper bound $|S|$. Since every time we update τ, it is at least $1-\epsilon$ times its old value, and at the end of each epoch τ is greater than maximum marginal value to S, we can say that throughout the entire algorithm (not just the end of epochs), τ is always at least $(1-\epsilon)\max_{x \in \mathbb{V}} \Delta(x|S)$. This is in particular true at the beginning of the algorithm that we set $\tau = \max_{x \in \mathbb{V}} f(\{x\})$. Using submodularity of f, we know that $\sum_{x \in \mathsf{OPT}} \Delta(x|S) \geq f(\mathsf{OPT}) - f(S)$. So $\max_{x \in \mathbb{V}} \Delta(x|S)$ should be at least $(f(\mathsf{OPT}) - f(S))/|\mathsf{OPT}|$. Since every item we add has marginal value at least $\tau \geq (1-\epsilon)\max_{x \in \mathbb{V}} \Delta(x|S)$, we conclude that each item adds at least a value of $\frac{(1-\epsilon)(f(\mathsf{OPT})-f(S))}{|\mathsf{OPT}|}$. After adding t items, the gap $f(\mathsf{OPT}) - f(S)$ becomes at most $(f(\mathsf{OPT}) - f(\emptyset))(1 - \frac{1-\epsilon}{|\mathsf{OPT}|})^t$. With $t = |\mathsf{OPT}|\ln(L)/(1-\epsilon)$, this gap becomes less than 1, and since f is integral, $f(S)$ should be at least $f(\mathsf{OPT}) = L$ with $|S| = |\mathsf{OPT}|\ln(L)/(1-\epsilon)$ items. □

To upper bound the number of rounds, we categorize all rounds into two groups. We say a round is *good* if the algorithm adds at least $\frac{k}{2}$ items to S. Otherwise we call it a *bad* round. We upper bound the number of good and bad rounds separately to reach a unified bound on the total number of rounds of FastCover.

Lemma 47. *The number of good rounds in all epochs is at most* $\log_2 L$.

Proof. In a good round, at least $k/2$ items are added to S, and each addition increases the value of $f(S)$ by τ. So in a good round, $f(S)$ is increased by at least $k\tau/2$. On the other hand, we define k to be $\lceil (L - f(S^{before}))/\tau \rceil$ where S^{before} is set S just before starting this round. So $f(S)$ is increased by at least $L - f(S^{before})/2$ in this good round. In other words, the difference $L - f(S)$ is reduced by at least a multiplicative factor of 2 in each good round. Once this difference goes below 1, we know $f(S) \geq L$, and the algorithm terminates. Therefore there are at most $\log_2 L$ good rounds in total. □

Next we bound the total number of bad rounds. Since in each epoch, we reduce τ by a factor of $(1-\epsilon)$ until it becomes at most 1, the number of epochs is upper bounded by $1+\log_{\frac{1}{1-\epsilon}}(M) \leq \frac{\log(M)}{\log(1/(1-\epsilon))} \leq \frac{\log(M)}{\epsilon}$. Therefore we need to upper bound the number of bad rounds in each epoch.

Lemma 48. *The number of bad rounds in each epoch is at most* $\log_{3/2}(n/km)$ *with high probability.*

Proof. In an epoch, the value of τ is unchanged, and we keep adding items to S. So the set of items that each machine could potentially send to the central machine (items with marginal value at least τ to set S) only shrinks. We call these items candidate items. At the beginning of an epoch, there are at most n/m such candidate items in each machine since T_i has n/m items. The epoch ends when each machine has at most k candidate items. We show that in each bad round, this set of candidate items shrinks by at least a factor of $2/3$ with high probability, and therefore the number of bad rounds in each epoch is no more than $\log_{3/2}(n/km)$.

Now we focus on a bad round, and how it changes the set of candidate items in a machine i. Let S^{before} and S^{after} be the values of set S before and after a bad round. We note that $S^{after} \setminus S^{before}$ has less than $k/2$ items. We define S_i^{before} to be $\{x | x \in T_i \;\&\; f(S^{before} \cup \{x\}) - f(S^{before}) \geq \tau\}$ which is the set of candidate items of machine i before this round. We note that set S_i is a random subset of S_i^{before} with size at most k. We similarly define S_i^{after} to be $\{x | x \in T_i \;\&\; f(S^{after} \cup \{x\}) - f(S^{after}) \geq \tau\}$ which is the set of candidate items in machine i in the next round. We prove that with high probability the size of $|S_i^{after}| \leq 2|S_i^{before}|/3$.

If there are at most k items in S_i^{before}, the whole set S_i^{before} is sent to the central machine, and each item in it is either added to S or its marginal value to S becomes less than τ after this round. So S_i^{after} is empty in this case. In the other case, k random items in S_i^{before} are selected to be sent to the central machine. For the sake of analysis, we define an intermediary hypothetical set S_i^{hyp} which is a set sandwiched between S_i^{after} and S_i^{before}. Let S_i^{hyp} be the set $\{x | x \in S_i^{before} \; AND \; f(S^{after} \cup \{x\}) - f(S^{after} \setminus \{x\}) \geq \tau\}$. This is the set of items that either they are candidate items in the next round (part of S_i^{after}) or they were added to S, and if we remove them from S, they become a candidate item in the next round. By definition, we have $S_i^{after} \subseteq S_i^{hyp} \subseteq S_i^{before}$. The significance of S_i^{hyp} is that any item machine i chooses from it to send to the central machine will be chosen by definition. So if S_i^{hyp} has p fraction of S_i^{before} for some

$0 \leq p \leq 1$, in expectation pk items out of k items of S_i will be selected in this round. Using concentration bounds, we know that if $\frac{|S_i^{hyp}|}{S_i^{before}}$ is at least $\frac{2}{3}$, with high probability at least $k/2$ selected items in S_i are in S_i^{hyp}, and consequently will be added to S. However we know that we are in a bad round, and less than $k/2$ items are added to S. Therefore with high probability $|S_i^{hyp}|$ is less than $2|S_i^{before}|/3$, and consequently $|S_i^{after}|$ is also less than $2|S_i^{before}|/3$ which completes the proof. □

Next we summarize all lemmas and prove our main guarantees for the number of rounds of Algorithm FastCover.

Proof of Theorem 21

Proof. Lemma 46 provides the desired upper bound on size of the solution. Using Lemmas 47, and 48, we know that there are at most $\log_2(L)$ good rounds in total, and $\log_{3/2}(n/km)$ bad rounds in each epoch with high probability. We have also proved that there are not more than $\log(M)/\epsilon$ epochs. Therefore the total number of rounds is upper bounded by $\log_{3/2}(n/km)\log(M)/\epsilon + \log_2(L)$. □

Remark 49. Each machine sends back at most k items. In proof of Theorem 21, we showed that τ is always at least $(1-\epsilon)$ times the maximum marginal value to set S. Using submodularity, we know that $L - f(S) \leq f(\mathsf{OPT}) - f(S) \leq \sum_{x \in \mathsf{OPT}} \Delta(x|S)$. So $k = (L - f(S))/\tau$ cannot be more than $|\mathsf{OPT}|/(1-\epsilon)$. The space requirement for the central machine is $km \leq m|\mathsf{OPT}|/(1-\epsilon)$, and for each distributed machine is n/m. Therefore our overall space requirement is no more than $\max\{n/m, m|\mathsf{OPT}|/(1-\epsilon)\}$.

A.4 Proofs from Chapter 9

Proof of theorem 22

Proof. Consider a chain of r instances of our streaming algorithm, i.e. $\{\text{IndStream}_1, \cdots, \text{IndStream}_r\}$. For each $i \in [1, r]$, IndStream_i provides an α-approximation guarantee on the ground set V_i of items it has received. Therefore we have:

$$f(S_i) \geq \alpha f(S_i \cup C_i), \tag{A.4.1}$$

where $C_i = C^* \cap V_i$ for all $i \in [1, r]$, and C^* is the optimal solution. Moreover, for each i, S'_i is the solution of the unconstrained maximization algorithm on ground set S_i. Therefore, we have:

$$f(S'_i) \geq \beta f(S_i \cap C_i), \tag{A.4.2}$$

where β is the approximation guarantee of the unconstrained submodular maximization algorithm (Unconstrained-Max).

We now use the following lemma from [Buc+14] to bound the total value of the solutions provided by the r instances of IndStream.

Lemma 50 (*Lemma 2.2. of* [Buc+14]). *Let $f' : 2^V \to R$ be submodular. Denote by $A(p)$ a random subset of A where each element appears with probability at most p (not necessarily independently). Then, $\mathbb{E}[f'(A(p))] \geq (1-p)f'(\emptyset)$.*

Let S be a random set which is equal to every one of the sets $\{S_1, \cdots, S_r\}$ with probability $p = 1/r$. For $f' : 2^V \to R$, and $f'(S) = f(S \cup \mathsf{OPT})$, from Lemma 50 we get:

$$\mathbb{E}[f'(S)] = \mathbb{E}[f(S \cup C^*)] = \frac{1}{r}\sum_{i=1}^{r} f(S_i \cup C^*) \overset{\text{Lemma 50}}{\geq} (1-p)f'(\emptyset) = (1-\frac{1}{r})f(C^*) \tag{A.4.3}$$

Also, note that each instance i of IndStream in the chain has processed all the elements of the ground set V except those that are in the solution of the previous instances of IndStream in the chain. As a result, $V_i = V \setminus \cup_{j=1}^{i-1} S_i$, and for every $i \in [1, r]$, we can write:

$$f(C_i) + f(C^* \cap (\cup_{j=1}^{i-1} S_j)) = f(C_i) + f(\cup_{j=1}^{i-1}(C^* \cap S_j)) = f(C^*). \tag{A.4.4}$$

Now, using Eq. A.4.3, and via a similar argument as used in [FHK17], we can write:

$$\begin{aligned}
(r-1)f(C^*) &\leq \sum_{i=1}^{r} f(S_i \cup C^*) && \text{By Eq. A.4.3} \\
&\leq \sum_{i=1}^{r} \Big[f(S_i \cup C_i) + f\big(\cup_{j=1}^{i-1} (C^* \cap S_j)\big)\Big] && \text{By Eq. A.4.4} \\
& && \text{(A.4.5)} \\
&\leq \sum_{i=1}^{r} \left[f(S_i \cup C_i) + \sum_{j=1}^{i-1} f(C^* \cap S_j)\right] && \text{(A.4.6)} \\
&\leq \sum_{i=1}^{r} \left[\frac{1}{\alpha} f(S_i) + \frac{1}{\beta} \sum_{j=1}^{i-1} f(S'_j)\right] && \text{By Eq. A.7.3, Eq. A.7.4} \\
&\leq \sum_{i=1}^{r} \left[\frac{1}{\alpha} f(S) + \frac{1}{\beta} \sum_{j=1}^{i-1} f(S)\right] && \text{By definition of } S \text{ in Algorithm 9} \\
&= \left(\frac{r}{\alpha} + \frac{r(r-1)}{2\beta}\right) f(S).
\end{aligned}$$

Hence, we get:

$$f(S) \geq \frac{r-1}{r/\alpha + r(r-1)/2\beta} f(C^*) \tag{A.4.7}$$

Taking the derivative w.r.t. r, we get that the ratio is maximized for $r = \left\lceil \sqrt{\frac{2\beta}{\alpha}} + 1 \right\rceil$. Plugging this value into Eq. A.4.7, we have:

$$\begin{aligned}
f(S) &\geq \frac{1 - \frac{1}{\sqrt{2\beta/\alpha + 1}}}{\frac{1}{\alpha} + \frac{\sqrt{2\beta/\alpha}}{2\beta}} f(C^*) \\
&= \frac{\sqrt{2\beta/\alpha}}{(\sqrt{\frac{2\beta}{\alpha}} + 1)(\frac{1}{\alpha} + \frac{\sqrt{2\beta/\alpha}}{2\beta})} f(C^*) \\
&= \frac{\sqrt{2\beta}}{(\sqrt{2\beta} + 1/\sqrt{\alpha})(1/\sqrt{\alpha} + 1/\sqrt{2\beta})} f(C^*) \\
&= \frac{\sqrt{2\beta}}{(1/\sqrt{\alpha} + 1/\sqrt{2\beta})^2} f(C^*)
\end{aligned}$$

Using $\beta = 1/2$ from [Buc+15], we get the desired result:

$$f(S) \geq \frac{1}{(1/\sqrt{\alpha}+1)^2} f(C^*)$$

Finally, Corollary 23 follows by replacing $\alpha = 1/4p$ from [CGQ15] and $\beta = 1/2$ from [Buc+15]:

$$f(S) \geq \frac{1}{(2\sqrt{p}+1)^2} f(C^*)$$

□

For calculating the average update time, we consider the worst case scenario, where every element can go through the entire chain of r instances of IndStream at some point during the run of Streaming Local Search. Here the total running time of the algorithm is $O(nrT)$, where n is the size of the stream, and T is the update time of IndStream. Hence the average update time per element for Streaming Local Search is $O(nrT/n) = O(rT)$.

Proof of theorem 24

Proof. Here, a (fixed) density threshold ρ is used to restrict the IndStream to only pick elements if $\frac{f_{S_i}(e)}{\sum_{j=1}^{d} c_{je}} \geq \rho$. We first bound the approximation guarantee of this new algorithm IndStreamDensity, and then use a similar argument as in the proof ot Theorem 22 to provide the guarantee for Streaming Local Search. Consider an optimal solution C^* and set:

$$\rho^* = \frac{2}{\left(\frac{1}{\sqrt{\alpha}} + \frac{1}{\sqrt{\beta}}\right)\left(\frac{1}{\sqrt{\alpha}} + 2d\sqrt{\alpha} + \frac{1}{\sqrt{\beta}}\right)} f(C^*). \tag{A.4.8}$$

By submodularity we know that $m \leq f(C^*) \leq mk$, where k is an upper bound on the cardinality of the largest feasible solution, and m is the maximum value of any singleton element. Hence:

$$\frac{2m}{\left(\frac{1}{\sqrt{\alpha}} + \frac{1}{\sqrt{\beta}}\right)\left(\frac{1}{\sqrt{\alpha}} + 2d\sqrt{\alpha} + \frac{1}{\sqrt{\beta}}\right)} \leq \rho^* \leq \frac{2mk}{\left(\frac{1}{\sqrt{\alpha}} + \frac{1}{\sqrt{\beta}}\right)\left(\frac{1}{\sqrt{\alpha}} + 2d\sqrt{\alpha} + \frac{1}{\sqrt{\beta}}\right)}.$$

Thus there is a run of the algorithm with density threshold $\rho \in R$ such that:

$$\rho \leq \rho^* \leq (1+\epsilon)\rho. \tag{A.4.9}$$

For the run of the algorithm corresponding to ρ, we call the solution of the first instance $\textsc{IndStreamDensity}_1$, S_ρ. If $\textsc{IndStreamDensity}_1$ terminates by exceeding some knapsack capacity, we know that for one of the knapsacks $j \in [d]$, we have $c_j(S_\rho) > 1$, and hence also $\sum_{j=1}^d c_j(S_\rho) > 1$ (W.l.o.g. we assumed the knapsack capacities are 1). On the other hand, the extra density threshold we used for selecting the elements tells us that for any $e \in S_\rho$, we have $\frac{f_{S_\rho}(e)}{\sum_{j=1}^d c_{je}} \geq \rho$. I.e., the marginal gain of every element added to the solution S_ρ was greater than or equal to $\rho \sum_{j=1}^d c_{je}$. Therefore, we get:

$$f(S_\rho) \geq \sum_{e \in S_\rho} (\rho \sum_{j=1}^{d} c_{je}) > \rho.$$

Note that S_ρ is not a feasible solution, as it exceeds the j-th knapsack capacity. However, the solution before adding the last element e to S_ρ, i.e. $T_\rho = S_\rho - \{e\}$, and the last element itself are both feasible solutions, and by submodularity, the best of them provide us with the value of at least

$$\max\{f(T_\rho), f(\{e_f\})\} \geq \frac{\rho}{2}.$$

On the other hand, if $\textsc{IndStreamDensity}_1$ terminates without exceeding any knapsack capacity, we divide the elements in $C^* \setminus S_\rho$ into two sets. Let $C^*_{<\rho}$ be the set of elements from C^* which cannot be added to S_ρ because their density is below the threshold, i.e., $\frac{f_{S_\rho}(e)}{\sum_{i=1}^d c_{je}} < \rho$ and $C^*_{\geq\rho}$ be the set of elements from C^* which cannot be added to S_ρ due to independence system constraints. For the elements of the optimal solution C^* which cannot be added to S_ρ because their density is below the threshold, we have:

$$f_{S_\rho}(C^*_{<\rho}) \leq \sum_{e \in C_{<\rho}} \rho \sum_{j=1}^{d} c_{je} = \rho \sum_{j=1}^{d} \sum_{e \in C_{<\rho}} c_{je}$$

Since $C_{<\rho}$ is a feasible solution, we know that $\sum_{e \in C_{<\rho}} c_{je} \leq 1$, and therefore:

$$f_{S_\rho}(C^*_{<\rho}) \leq d\rho \leq \rho \sum_{j=1}^{d} \sum_{e \in C_{<\rho}} c_{je} \leq d\rho \leq d\rho^* \tag{A.4.10}$$

On the other hand, if the ground set was restricted to elements that pass the density threshold, then S_ρ would be a subset of that ground set, and the approximation guarantee of IndStream$_1$ still holds; hence from Eq. A.7.3 we know that:

$$f(S_\rho) \geq \alpha f(S_\rho \cup C^*_{\geq \rho}),$$

and thus we obtain:

$$f_{S_\rho}(C^*_{\geq\rho}) = f(S_\rho \cup C^*_{\geq\rho}) - f(S_\rho) \leq (\frac{1}{\alpha} - 1) f(S_\rho). \tag{A.4.11}$$

Adding Eq A.7.1 and A.7.2, and using submodularity we get:

$$f(S_\rho \cup C^*) - f(S_\rho) \leq f_{S_\rho}(C^*_{<\rho}) + f_{S_\rho}(C^*_{\geq\rho}) \leq (\frac{1}{\alpha} - 1) f(S_\rho) + d\rho$$

Therefore,

$$f(S_\rho) \geq \alpha f(S_\rho \cup C^*) - \alpha d\rho. \tag{A.4.12}$$

Now, using a similar argument as in the proof of Theorem 22, we have:

$$\begin{aligned}
(r-1) f(C^*) &\leq \sum_{i=1}^{r} f(S_i \cup C^*) && \text{By Eq. A.4.3} \\
&\leq \sum_{i=1}^{r} f(S_i \cup C_i) + \sum_{i=1}^{r} \sum_{j=1}^{i-1} f(C^* \cap S_j) && \text{By Eq. A.4.6} \\
&\leq \frac{1}{\alpha} \sum_{i=1}^{r} [f(S_i) + \alpha d\rho] + \frac{1}{\beta} \sum_{i=1}^{r} \sum_{j=1}^{i-1} f(S'_j) && \text{By Eq. A.4.12} \\
&\leq \frac{1}{\alpha} \sum_{i=1}^{r} [f(S) + \alpha d\rho] + \frac{1}{\beta} \sum_{i=1}^{r} \sum_{j=1}^{i-1} f(S) && \text{By definition of } S \text{ in Algorithm 10} \\
&= \left(\frac{r}{\alpha} + \frac{r(r-1)}{2\beta} \right) f(S) + rd\rho
\end{aligned}$$

□

Hence, we have:

$$f(S) \geq \frac{r-1}{r/\alpha + r(r-1)/2\beta} f(C^*) - \frac{rd\rho}{r/\alpha + r(r-1)/2\beta} f(C^*)$$

From Eq. A.4.9, we know that $\rho \geq (1-\varepsilon)\rho^*$. Using Eq. A.4.8, we get:

$$f(S) \geq \frac{r-1}{r/\alpha + r(r-1)/2\beta} f(C^*) - \frac{\frac{2rd(1-\varepsilon)}{(1/\sqrt{\alpha}+1/\sqrt{\beta})(1/\sqrt{\alpha}+2d\sqrt{\alpha}+1/\sqrt{\beta})}}{r/\alpha + r(r-1)/2\beta} f(C^*)$$

Plugging in $r = \left\lceil \sqrt{\frac{2\beta}{\alpha}} + 1 \right\rceil$ and simplifying, we get the desired result:

$$\begin{aligned}
f(S) &\geq \frac{\sqrt{\frac{2\beta}{\alpha}} - \frac{2d\left(\sqrt{\frac{2\beta}{\alpha}}+1\right)(1-\varepsilon)}{\left(\frac{1}{\sqrt{\alpha}}+\frac{1}{\sqrt{\beta}}\right)\left(\frac{1}{\sqrt{\alpha}}+2d\sqrt{\alpha}+\frac{1}{\sqrt{\beta}}\right)}}{\frac{1}{\alpha}\sqrt{\frac{2\beta}{\alpha}} + \frac{2}{\alpha} + \sqrt{\frac{1}{2\beta\alpha}}} f(C^*) \\
&= \frac{\sqrt{2\beta}\left(\frac{1}{\sqrt{\alpha}}+\frac{1}{\sqrt{\beta}}\right)\left(\frac{1}{\sqrt{\alpha}}+2d\sqrt{\alpha}+\frac{1}{\sqrt{\beta}}\right) - 2d(1-\varepsilon)\left(\sqrt{2\beta}+\sqrt{\alpha}\right)}{\left(\frac{\sqrt{2\beta}}{\alpha}+\frac{2}{\sqrt{\alpha}}+\sqrt{\frac{1}{2\beta}}\right)\left(\frac{1}{\sqrt{\alpha}}+\frac{1}{\sqrt{\beta}}\right)\left(\frac{1}{\sqrt{\alpha}}+2d\sqrt{\alpha}+\frac{1}{\sqrt{\beta}}\right)} f(C^*) \\
&\geq \frac{1-\varepsilon}{(1/\sqrt{\alpha}+1/\sqrt{\beta})(1/\sqrt{\alpha}+2d\sqrt{\alpha}+1/\sqrt{\beta})} f(C^*)
\end{aligned}$$

For $\beta = 1/2$ from [Buc+15], we get the desired result:

$$f(S) \geq \frac{1-\epsilon}{(1+1/\sqrt{\alpha})(1+2d\sqrt{\alpha}+1/\sqrt{\alpha})} f(C^*)$$

Corollary 25 follows by replacing $\alpha = 1/4p$ from [CGQ15] and $\beta = 1/2$ from [Buc+15]:

$$f(S) \geq \frac{1-\varepsilon}{1+4p+4\sqrt{p}+d(2+1/\sqrt{p})} f(C^*)$$

The average update time for one run of the algorithm corresponding to a $\rho \in R$ can be calculated as in the proof of Theorem 22. We run the algorithm for $\log(k)/\varepsilon$ different values of ρ, and hence the average update time of STREAMING LOCAL SEARCH per element is $O(rT\log(k)/\varepsilon)$. However, the algorithm can be run in parallel for the $\log(k)/\varepsilon$ values of ρ (line 7 of Algorithm 10), and hence using parallel processing, the average update time per element is $O(rT)$.

A.5 Proofs from Chapter 10

Proof of Theorem 27

Approximation guarantee. First, we proof that ROBUST-STREAMING provides an α-approximation guarantee, using $O(r.M)$ memory. Note that since we experience at most m adversarial deletions, for $r = m+1$, one of the $(m+1)$ instances of STREAMINGALG has never experienced any deletion. Hence, there is at least one instance left. We show that among all instances left (i.e., did not face any deletions from their memory), the one with the lowest index provides the claimed approximation guarantee. Let us assume that at time t the instance i of STREAMINGALG is the remaining one with lowest index. As a result, the content of all $M_t^{(1)}, M_t^{(2)}, \ldots, M_t^{(i-1)}$ memories (in some order) has been passed to $M_t^{(i)}$. This implies that $M_t^{(i)}$ has seen all the elements in the data stream V_t except the ones deleted D_t by time t. Since STREAMINGALG provides an α approximation for reading any sequence (*independent of the order* of reading its elements), the instance i of STREAMINGALG provides an α approximation for reading $V_t \setminus D_t$. Hence $f(S_t^{(i)}) \geq \alpha \mathsf{OPT}_t$ where OPT_t is the optimum solution for the constrained optimization problem (10.1.2) when we have m deletions. Clearly, in order to run ROBUST-STREAMING we need $O(r \cdot M)$ memory as we have at most r instances of STREAMINGALG running simultaneously, each requiring memory of size M.

Update time. We now calculate an upper bound on the update time of the algorithm. The new element received from the stream is first processed by STREAMINGALG $^{(1)}$. If it is accepted, it may result in discarding at most M elements from the memory $M_t^{(1)}$ of STREAMINGALG $^{(1)}$. In the worst case, the discarded elements from $M_t^{(1)}$ result in discarding all the M elements from the memory $M_t^{(2)}$ of STREAMINGALG $^{(2)}$, which in turn may result in discarding all the M elements from the memory $M_t^{(3)}$ of

StreamingAlg $^{(3)}$. This process continues until all the elements int the memory of all the StreamingAlgs are discarded. Taking a sum over the number of discarded elements, the update time will be scaled by

$$Mr + M(r-1) + M(r-2) + M = O(M.r^2) \tag{A.5.1}$$

The update time in case of a deletion can be calculated in a similar manner. For calculating the average update time, we note that in the worst case, every element can go through the entire chain of r instances of StreamingAlgs. Therefore, if all the elements gets discarded by all the instances of StreamingAlg at some point during the run of Robust-Streaming, we have a total running time of $O(n.r.T)$, where n is the size of the stream. Hence, the average update time per element is $O(r.T)$.

Proof of Theorem 28

If each element has probability p of being deleted, the probability that no elements is deleted from the solution set of at least one of the streaming algorithms is

$$1 - \left(1 - (1-p)^k\right)^r$$

and we want this probability to be greater than $1 - \delta$. Hence, we have

$$\begin{aligned}
&1 - \left(1 - (1-p)^k\right)^r \geq 1 - \delta \\
&-\left(1 - (1-p)^k\right)^r \geq -\delta \\
&r \log \left(1 - (1-p)^k\right)^{-1} \geq \log(1/\delta) \\
&r(1-p)^k \geq \log(1/\delta) \\
&r \geq (1-p)^{-k} \log(1/\delta)
\end{aligned}$$

Having $p = \frac{\alpha}{n} \leq 1$, r is inversly proportional to $(1 - \frac{\alpha}{n})^k$. Therefore, for fixed k, δ and p, a *constant* number r is sufficient to support $m = pn$ *independently* of n.

A.6 Proofs from Chapter 12

The following lemma gives us the approximation guarantee.

Lemma 51. *Given a current solution A, the expected gain of* Stochastic-Greedy *in one step is at least $\frac{1-\epsilon}{k}\sum_{a\in A^*\setminus A}\Delta(a|A)$.*

Proof. Let us estimate the probability that $R\cap(A^*\setminus A)\neq\emptyset$. The set R consists of $s=\frac{n}{k}\log\frac{1}{\epsilon}$ random samples from $V\setminus A$ (w.l.o.g. with repetition), and hence

$$\begin{aligned}\Pr[R\cap(A^*\setminus A)=\emptyset] &= \left(1-\frac{|A^*\setminus A|}{|V\setminus A|}\right)^s\\ &\leq e^{-s\frac{|A^*\setminus A|}{|V\setminus A|}}\\ &\leq e^{-\frac{s}{n}|A^*\setminus A|}.\end{aligned}$$

Therefore, by using the concavity of $1-e^{-\frac{s}{n}x}$ as a function of x and the fact that $x=|A^*\setminus A|\in[0,k]$, we have

$$Pr[R\cap(A^*\setminus A)\neq\emptyset]\geq 1-e^{-\frac{s}{n}|A^*\setminus A|}\geq(1-e^{-\frac{sk}{n}})\frac{|A^*\setminus A|}{k}.$$

Recall that we chose $s=\frac{n}{k}\log\frac{1}{\epsilon}$, which gives

$$\Pr[R\cap(A^*\setminus A)\neq\emptyset]\geq(1-\epsilon)\frac{|A^*\setminus A|}{k}. \tag{A.6.1}$$

Now consider Stochastic-Greedy: it picks an element $a\in R$ maximizing the marginal value $\Delta(a|A)$. This is clearly as much as the marginal value of an element randomly chosen from $R\cap(A^*\setminus A)$ (if nonempty). Overall, R is equally likely to contain each element of $A^*\setminus A$, so a uniformly random element of $R\cap(A^*\setminus A)$ is actually a uniformly random element of $A^*\setminus A$. Thus, we obtain

$$\mathbf{E}[\Delta(a|A)]\geq\Pr[R\cap(A^*\setminus A)\neq\emptyset]\times\frac{1}{|A^*\setminus A|}\sum_{a\in A^*\setminus A}\Delta(a|A).$$

Using (A.6.1), we conclude that $\mathbf{E}[\Delta(a|A)]\geq\frac{1-\epsilon}{k}\sum_{a\in A^*\setminus A}\Delta(a|A)$. □

Now it is straightforward to finish the proof of Theorem 29. Let $A_i=\{a_1,\dots,a_i\}$ denote the solution returned by STOCHASTIC-GREEDY after i steps. From Lemma 51,

$$\mathbf{E}[\Delta(a_{i+1}|A_i) \mid A_i] \geq \frac{1-\epsilon}{k} \sum_{a \in A^* \setminus A_i} \Delta(a|A_i).$$

By submodularity,

$$\sum_{a \in A^* \setminus A_i} \Delta(a|A_i) \geq \Delta(A^*|A_i) \geq f(A^*) - f(A_i).$$

Therefore,

$$\begin{aligned}\mathbf{E}[f(A_{i+1}) - f(A_i) \mid A_i] &= \mathbf{E}[\Delta(a_{i+1}|A_i) \mid A_i] \\ &\geq \frac{1-\epsilon}{k}(f(A^*) - f(A_i)).\end{aligned}$$

By taking expectation over A_i,

$$\mathbf{E}[f(A_{i+1}) - f(A_i)] \geq \frac{1-\epsilon}{k}\mathbf{E}[f(A^*) - f(A_i)].$$

By induction, this implies that

$$\begin{aligned}\mathbf{E}[f(A_k)] &\geq \left(1 - \left(1 - \frac{1-\epsilon}{k}\right)^k\right) f(A^*) \\ &\geq \left(1 - e^{-(1-\epsilon)}\right) f(A^*) \geq (1 - 1/e - \epsilon) f(A^*).\end{aligned}$$

A.7 Proofs from Chapter 13

Proof of theorem 30

The proof is divided into two cases.

Case One: This is when FANTOM terminates by exceeding some knapsack capacity. Let S_ρ be the set when FANTOM exceeds capacity for some $i \in [d]$ for the first time, i.e., $c_i(S_\rho) > 1$. Then we also have that $\sum_{i=1}^{d} c_i(S_\rho) > 1$. Since every element that we include satisfies $\frac{f_S(j)}{\sum_{i=1}^{d} c_{ij}} \geq \rho$, with respect to the current solution S we obtain:

$$f(S_\rho) \geq \rho \sum_{i=1}^{d} \sum_{j \in S_\rho} c_{ij} > \rho.$$

However, S_ρ is infeasible as it exceeds the capacity. But if j is the last added item, both j and $T_\rho = S_\rho - \{j\}$ should be feasible. Then we get

$$\begin{aligned}
\max\{f(T_\rho), f(\{j\})\} &\geq \frac{1}{2}\left(f(T_\rho) + f(\{j\})\right) \\
&\geq \frac{1}{2}\left(f(T_\rho \cup \{j\}) + f(\phi)\right) && \text{By submodularity} \\
&= \frac{1}{2} f(S_\rho) \\
&\geq \frac{1}{2}\rho.
\end{aligned}$$

Case Two: This is when FANTOM terminates as it cannot add items due to p-system constraint and never exceeds any knapsack capacity. Let S_ρ be the set returned. Consider set C as described in the statement of the theorem and divide it into two sets. Let $C_{<\rho}$ be the set of elements from C which cannot be added because their density is below the threshold, i.e., $\frac{f_{S_\rho}(j)}{\sum_{i=1}^{d} c_{ij}} < \rho$ and $C_{\geq \rho}$ be the set of elements from C which cannot be added due to p-system constraint. Then we first derive inequalities with respect to each of these sets. First consider the set $C_{<\rho}$. We obtain

$$\begin{aligned}
f_{S_\rho}(C_{<\rho}) &\leq \sum_{e \in C_{<\rho}} f_{S_\rho}(e) \text{ (By submodularity)} \\
&\leq \sum_{e \in C_{<\rho}} \rho \sum_{i=1}^{d} c_{ie} \text{ (By definition } C_{<\rho}) \\
&= \rho \sum_{i=1}^{d} \sum_{e \in C_{<\rho}} c_{ie} \\
&\leq \rho \sum_{i=1}^{d} 1 \text{ (As } C_{<\rho} \text{ is a feasible solution)} \\
&= \rho d.
\end{aligned} \tag{A.7.1}$$

Now consider the set $T = S_\rho \cup C_{\geq \rho}$. This time if we run greedy algorithm on T with

p-system constraints and no knapsack constraints then we still get S_ρ as the solution. Hence, by lemma 3.2 of [Gup+10b] we get that

$$f(S_\rho) \geq \frac{1}{p+1} f(S_\rho \cup C_{\geq\rho}).$$

By rewriting we obtain

$$f_{S_\rho}(C_{\geq\rho}) = f(S \cup C_{\geq\rho}) - f(S_\rho) \leq p f(S_\rho). \tag{A.7.2}$$

Adding Eq A.7.1 and A.7.2, and using submodularity we get

$$f(S_\rho \cup C) - f(S_\rho) \leq f_{S_\rho}(C_{<\rho}) + f_{S_\rho}(C_{\geq\rho}) \leq \rho d + p f(S_\rho).$$

Hence,

$$f(S_\rho) \geq \frac{1}{p+1} f(S_\rho \cup C) - \frac{\rho d}{p+1}.$$

Proof of theorem 31

Let $T_i = \cup_{j=1}^{i} S_j$. For the remaining part of the proof we assume that for each i, $f(S_i) \geq \frac{f(S_i \cup (C - T_{i-1}))}{p+1} - \frac{d\rho}{p+1}$ (or else by the proof of Theorem 30 the proof is simple). Now we obtain a list of inequalities. For each i by assumption we have

$$f(S_i) \geq \frac{f(S_i \cup (C - T_{i-1}))}{p+1} - \frac{d\rho}{p+1}. \tag{A.7.3}$$

Also, for each i by the fact about unconstrained maximization from [Buc+15] we have

$$f(S_i') \geq \frac{1}{2} f(S_i \cap C). \tag{A.7.4}$$

Now by induction we prove that $\forall t \in [2, p+1]$ we have

$$\sum_{i=1}^{t} f(S_i \cup (C - T_{i-1})) + (t-i) f(S_i \cap C) \geq (t-1) f(C) + f(T_i \cup C).$$

First, consider the base case $t = 2$:

$$
\begin{aligned}
&f(S_1 \cup C) + f(S_2 \cup (C - S_1)) + f(S_1 \cap C)\\
&\geq f(S_1 \cup S_2 \cup C) + f(C - S_1) + f(S_1 \cap C) \text{ (By submodularity)}\\
&\geq f(S_1 \cup S_2 \cup C) + f(C) \text{ (By submodularity)}\\
&= f(T_2 \cup C) + f(C).
\end{aligned}
$$

Now, we prove the inductive case:

$$
\begin{aligned}
&\sum_{i=1}^{t} [f(S_i \cup (C - T_{i-1})) + (t-i) f(S_i \cap C)]\\
&\geq \sum_{i=1}^{t-1} [f(S_i \cup (C - T_{i-1})) + (t-1-i) f(S_i \cap C)]\\
&+ f(S_t \cup (C - T_{t-1})) + \sum_{i=1}^{t-1} f(S_i \cap C)\\
&\geq (t-2) f(C) + f(T_{t-1} \cup C)\\
&+ f(S_t \cup (C - T_{t-1})) + \sum_{i=1}^{t-1} f(S_i \cap C) \text{ (By induction)}\\
&\geq (t-2) f(C) + f(T_t \cup C)\\
&+ f(C - T_{t-1}) + \sum_{i=1}^{t-1} f(S_i \cap C) \text{ (By submodularity)}\\
&\geq (t-2) f(C) + f(T_t \cup C) + f(C) \text{ (By submodularity)}\\
&= (t-1) f(C) + f(T_t \cup C).
\end{aligned} \tag{A.7.5}
$$

By multiplying Eq. A.7.3 by $p+1$ and Eq. A.7.4 by $2(p+1-i)$ we obtain

$$
\begin{aligned}
(p+1)\sum_{i=1}^{p+1} f(S_j) + \sum_{i=1}^{p+1} 2(p+1-i) f(S'_j) &\geq \sum_{i=1}^{p+1} \{f(S_i \cup (C - T_{i-1})) - d\rho\}\\
&+ \sum_{i=1}^{p+1} (p+1-i) f(S_i \cap C)\\
&\geq p f(C) - (p+1) d\rho. \text{ (from equation A.7.5)}
\end{aligned}
$$

Taking a max over the left hand side of the equation we get the following inequality

$$
\left((p+1)^2 + 2\sum_{i=1}^{p+1} (p+1-i)\right) \max_i (f(S_i), f(S'_i)) \geq p f(C) - (p+1) d.
$$

Hence,

$$\max_i(f(S_i), f(S'_i)) \geq \frac{p}{(p+1)(2p+1)} f(C) - \frac{d\rho}{2p+1}.$$

Proof of theorem 32

Let $r = \frac{2 \cdot p \cdot f(C)}{(p+1)(2p+2d+1)}$. Consider the value of $\rho \in R$ such that $r \leq \rho \leq (1+\epsilon)r$. Substituting this value in Theorem 31 we complete the proof.

Name of the recommended movies in table 13.1.

1- Serenity (2005)
2- Presto (2008)
3- How to Train Your Dragon (2010)
4- Snow White: A Tale of Terror (1997)
5- Tom and Jerry: The Movie (1992)
6- Spirited Away (Sen to Chihiro no kamikakushi) (2001)
7- Raiders of the Lost Ark (1981)
8- Three Musketeers, The (1948)
9- Pirate Movie, The (1982)
10- Spy Kids 3-D: Game Over (2003)
11- Pokemon Heroes (2003)
12- Barney's Great Adventure (1998)
13- Son of the Mask (2005)
14- Pokemon 4 Ever (a.k.a. Pokemon 4) (2002)
15- Turbo: A Power Rangers Movie (1997)
16- Porco Rosso (Crimson Pig) (Kurenai no buta) (1992)
17- Watership Down (1978)
18- Sholay (1975)
19- Bolt (2008)
20- Mummy, The (1999)
21- Twelve Tasks of Asterix (1976)
22- The Lego Movie (2014)
23- Laputa: Castle in the Sky (1986)

24- Interstate 60 (2002)
25- Super Mario Bros. (1993)

List of Tables

1.1 Summary of key contributions. n is the size of the dataset, m is the number of muchines, k is the size of the summary, r is the size of the largest feasible solution, Q is the desired utility, α is the approximation guarantee for streaming monotone submodular maximization algorithms under a collection of independence systems, d is the number of knapsack constraints, p is the ratio of the size of the largest vs. smallest independent sets in a p-system. 14

2.1 Approximation guarantees (τ) for monotone and non-monotone submodular maximization under different constraints. 27

2.2 Approximation guarantees for distributed monotone and non-monotone submodular maximization under different constraints. $\rho[\zeta]$ is the maximum size of the desired solution, and bounds for randomized algorithms that hold in expectation are marked (R). 31

2.3 Approximation guarantees, memory and update time for single-pass monotone and non-monotone streaming submodular maximization under different constraints. Bounds for randomized algorithms that hold in expectation are marked (R). α, M, T is the approximation guarantee, memory, and update time of the streaming monotone submodular maximization under a collection of independent systems and d knapsack constraints. 34

2.4 Comparison of running times and approximation ratios for non-monotone submodular maximization under different constraints. 36

9.1 Performance of various video summarization methods with segment size 10 on YouTube and OVP datasets, measured by F-Score (F), Precision (P), and Recall (R). The methods marked by (c) are centralized. 125

10.1 Robust-Streaming can be combined with the existing one-pass streaming algorithms in order to make them robust against m deletions. α, M is the approximation guarantee, and memory for streaming monotone submodular maximization under a collection of independence systems and d knapsack constraints. 141

13.1 Movies recommended by Fantom vs. Greedy and Density Greedy using Eq. 3.6.1, and Eq. 3.6.2 for $m = 5$ and $c = 1$. There are 19 genres in total: Action(1), Adventure(2), Animation (3), Children (4), Comedy (5), Crime (6), Documentary (7), Drama (8), Fantasy (9), Film-Noir (10), Horror (11), Musical (12), Mystery (13), Romance (14), Sci-Fi (15), Thriller (16), War (17), Western (18), IMAX (19). The user is interested in adventure, animation, and fantasy movies (genres **1,2,9**). See the Appendix for a complete list of movie names. 175

List of Figures

1.1 Illustration of a distributed framework. 5

1.2 Illustration of an streaming framework. 6

1.3 Cluster exemplars (left column) discovered by our distributed algorithm GreeDi described in Chapter 5 applied to the Tiny Images dataset [TFF08], and a set of representatives from each cluster. 9

1.4 (a) Example of an active set selected from a subset of the Yahoo! Webscope dataset [Yah12], and the corresponding decision boundary of the kernelized SVM classifier trained on the selected subset. For the sake of presentation, the data is projected onto its 2 largest principal components. (b) Example of a dominating set (marked as red) in a graph. The coverage of u, v is the set of their neighbors and is shown by $\varrho(\{u, v\})$. 10

1.5 (a) A summary obtained by our Robust-Streaming algorithm described in Chapter 10 from the geo-location trace of a bike route around Zurich [Fat15], (b) Example sensor placement obtained by our algorithm Stochastic-Greedy introduced in Chapter 12, in a realistic water network [Kra+08a]. 11

1.6 (a) Revenue maximization with 2 products. The shaded blue and red regions indicate the influence spread from nodes s_1 and s_2 to buy product 1, 2 respectively. Here, node u has some value to buy both of the products. (b) Personalized movies recommendation using our algorithm FANTOM described in Chapter 13 from MovieLens dataset [Mov]. Movies are from 19 genres: Action(1), Adventure(2), Animation (3), Children (4), Comedy (5), Crime (6), Documentary (7), Drama (8), Fantasy (9), Film-Noir (10), Horror (11), Musical (12), Mystery (13), Romance (14), Sci-Fi (15), Thriller (16), War (17), Western (18), IMAX (19). The user is interested in adventure, animation, and fantasy movies (genres 1,2,9). 13

5.1 Cluster exemplars (left column) discovered by our distributed algorithm GreeDi described in Section 5.2 applied to the Tiny Images dataset [TFF08], and a set of representatives from each cluster. 57

5.2 Illustration of our two-round algorithm GREEDI 63

5.3 Illustration of our two-round algorithm GREEDI for decomposable functions 67

5.4 Performance of GREEDI compared to the other benchmarks. a) and b) show the mean and standard deviation of the ratio of distributed vs. centralized solution for global and local objective functions with budget $k = 50$ and varying the number m of partitions. c) and d) show the same ratio for global and local objective functions for $m = 5$ partitions and varying budget k, for a set of 10,000 *Tiny Images*. 73

5.5 Performance of GREEDI compared to the other benchmarks. a) shows the distributed solution with $m = 8000$ and varying k for local objective functions on the whole dataset of 80,000,000 *Tiny Images*. b) shows a set of cluster exemplars discovered by GREEDI, and each column in c) shows 100 images nearest to exemplars 26 and d) shows 100 images nearest to exemplars 63 in b). 75

5.6 Performance of GREEDI compared to the other benchmarks. a) shows the ratio of distributed vs. centralized solution with $k = 50$ and varying m for *Parkinsons Telemonitoring*. b) shows the same ratio with $m = 10$ and varying k on the same dataset. 76

5.7 Performance of GreeDi with $m = 32$ and varying budget k compared to the other benchmarks on *Yahoo! Webscope data*. 76

5.8 Running time of GreeDi compared to the centralized greedy algorithm. a) shows the ratio of centralized vs. distributed solution with $k = 64, 128, 256$ and up to $m = 32$ machines for *Yahoo Webscope* data. b) shows the same ratio with $k = 64, 128, 256$ and up to $m = 512$ machines on the same dataset. Both experiments are performed on a cluster of 8 quad core machines. 77

5.9 Performance of GreeDi compared to the other benchmarks. a) shows the mean and standard deviation of the ratio of distributed to centralized solution for budget $k = 20$ with varying number of machines m and b) shows the same ratio for varying budget k with $m = 10$ on *Facebook-like social network*. 78

5.10 Performance of GreeDi compared to the GreedyScaling algorithm of [Kum+13] (as reported in their paper). a) shows the ratio of distributed to centralized solution on *Accidents* dataset with 340,183 elements and b) shows the same ratio for *Kosarak* dataset with 990,002 elements. The results are reported for varying budget k and varying number of machines $m = n/\mu$ where $\mu = O(kn^{\delta} \log n)$ and n is the size of the dataset. The results are reported for $\delta = 1/2$. Note that the results presented by [Kum+13] indicate that GreedyScaling generally requires a substantially larger number of MapReduce rounds compared to GreeDi. 79

6.1 Illustration of our multi-round algorithm DisCover, assuming it terminates in two rounds (without doubling search for ℓ). 89

6.2 Performance of DisCover compared to the centralized solution. a, b) show the solution set size vs. the number of rounds for various α, for a set of 10,000 *Tiny Images* and *Parkinsons Telemonitoring*. c) shows the same quantities for the Friendster network with 65,608,366 vertices. 91

7.1 Performance of FastCover vs. other baselines. a), b), c) solution set size vs. number of rounds for personalized location recommendation on a set of 3,056 GPS measurements, for covering 60%, 80%, 90% of the maximum utility of each user. 102

7.2 Performance of FastCover vs. other baselines. Solution set size vs. number of rounds for personalized movie recommendation on a set of 1000 movies, 138,493 users and 20,000,263 ratings, for covering a) 10%, b) 20%, c) 30% of the maximum utility of each user. 103

7.3 Performance of FastCover vs. other baselines. a) solution set size vs. coverage for simultaneously covering all users vs. covering users one by one and taking the union. The recommendation is on a set of 1000 movies for 1000 users. b) Exponential speedup of FastCover over DisCover on a subgraph of 14M nodes. 104

7.4 Performance of FastCover vs. other baselines. Solution set size vs. the number of rounds for covering a) 30%, b) 40%, c) 50% of the Friendster network with 65,608,366 vertices. d) solution set size vs. the number of rounds for FastCover and DisCover for covering 50% of the Friendster network. 106

9.1 Performance of Streaming Local Search compared to the other benchmarks. a) shows the ratio of the F-score obtained by Streaming Local Search and Fantom vs. the F-score obtained by the method of [Gon+14], using the sequential DPP objective and linear embeddings on YouTube dataset. b) shows the relative F-scores for non-linear features from a one-hidden-layer neural network. c), d) show the speed up of Streaming Local Search and Fantom over the method of [Gon+14]. 126

9.2 Performance of Streaming Local Search compared to the other benchmarks. a) shows the ratio of the F-score obtained by Streaming Local Search and Fantom vs. the F-score obtained by the method of [Gon+14], using the sequential DPP objective and linear embeddings on OVP dataset. b) shows the relative F-scores for non-linear features from a one-hidden-layer neural network. c), d) show the speedup of Streaming Local Search and Fantom over the method of [Gon+14]. 127

9.3 Performance of Streaming Local Search compared to the other benchmarks. a), c) show the utility and running time for Streaming Local Search and random selection vs. the utility and running time of Fantom, using the original DPP objective and linear embeddings on YouTube and OVP datasets. b), d) show similar qualities using non-linear features from a one-hidden-layer neural network. 128

9.4 Summary focused on judges, and singer for YouTube video 106. 129

9.5 Summary produced by method of [Gon+14] (top row), vs. Streaming Local Search (middle row), and a user selected summary (bottom row), for YouTube video 105. 129

10.1 Robust-Streaming uses r instances of a generic StreamingAlg to construct r non-overlapping memories at any given time t, i.e., $M_t^{(1)}$, $M_t^{(2)}, \ldots, M_t^{(r)}$. Each instance produces a solution $S_t^{(i)}$ and the solution returned by Robust-Streaming is the first valid solution $S_t = \{S_t^{(i)} | i = \min j \in [1 \cdots r], M_t^{(i)} \neq \text{null}\}$. 138

10.2 Performance of Robust-Streaming vs Sieve-Streaming for different deletion strategies (SG, RG) on a collection of 100 images. Here we fix $k = 5$ and $r = 3$. a) performance of Robust-Streaming and Sieve-Streaming normalized by the utility obtained by greedy that knows the deleted elements beforehand. b) updated solution of size $k = 5$ returned by Robust-Streaming after deleting the 1 image from the summary. 143

10.3 Robust-Streaming vs Sieve-Streaming and Stream-Greedy for different deletion strategies (SG, RG) on geolocation data. We fix $k = 20$ and $r = 5$. a) and c) show the performance of robustified Sieve-Streaming, whereas b) and d) show performance for robustified Stream-Greedy. a) and b) consider the performance after deletions at the end of the stream, while c) and d) consider average performance while deletions happen during the stream. 145

10.4 Robust-Streaming vs Sieve-Streaming and Stream-Greedy for different deletion strategies (SG, RG) on geolocation data. We fix $k=20$ and $r=5$. a) red and green triangles show a set of size 10 found by Sieve-Streaming and the updated solution found by Robust-Streaming where 70% of the points are deleted. b) set found by Streaming-Greedy, constrained to pick at most 1 point per grid cell (matroid constraint). Here $r = 5$, and we deleted the shaded area. . . . 146

10.5 Robust-Streaming vs random unbalanced and balanced selection and Sieve-Streaming selecting equal numbers of clicked and not-clicked data points, on 45,811,883 feature vectors from *Yahoo! Webscope* data. We fix $k = 10,000$ and delete 99% of the data points. 147

12.1 Performance comparisons. a), and b) show the performance of all the algorithms for different values of k on *Parkinsons Telemonitoring*, and a set of 10,000 *Tiny Images* respectively. c), and d) show the cost of all the algorithms for different values of k on the same datasets. 161

12.2 Performance comparisons. a), and b) show the performance of all the algorithms for different values of k on *Water Network*, and a set of 50,000 *Tiny Images* respectively. c) and d) show the cost of all the algorithms for different values of k on the same datasets. 162

12.3 Performance comparisons. The utility obtained versus cost for a fixed $k = 200$ on a) *Parkinsons Telemonitoring*, b) a set of 10,000 *Tiny Images*, c) *Water Network*, d) and a set of 50,000 *Tiny Images* respectively. 163

13.1 Performance of Fantom compared to the benchmarks for movie recommendation from a set of 10,437 movies from MovieLens. a) shows the performance of Fantom based on Eq. 3.6.1 for recommending movies from three genres: adventure, animation, and fantasy for $m = 3$, and varying knapsack limit c. b) shows the same quantity for $c = 1$, and varying the matroid limits m. c) shows the solution value based on Eq. 3.6.2 with $m = 3$, and varying c. d) shows the same quantity for $c = 1$, and varying m. 174

13.2 Performance of Fantom compared to the benchmarks for revenue maximization on top 5000 communities of YouTube with 39,841 nodes and 224,235 edges. a) shows the performance of Fantom for selling $q = 10$ product types, with $x = 50$ available items per product, matroid limit $m = 5$ for all communities, user constraint $u = 3$, and varying knapsack limit c. b) shows the same quantity for $c = 0.1$, $q = 10$, $x = 50$, $u = 3$ and varying m. c) shows the solution value for $c = 0.2$, $q = 10$, $x = 50$, $m = 3$, and varying u. d) shows the same quantity for $c = 0.2$, $q = 10$, $m = 5$, $u = 3$ and varying x. 177

13.3 Performance of Fantom compared to the benchmarks for personalized image summarization: a) shows the solution value for summarizing three categories airplane, automobile, and bird for $m = 3$, and varying the knapsack limit c. b) shows the same quantity for $c = 0.1$, and varying the matroid limits m. 178

List of Algorithms

1 Inefficient Distributed Submodular Maximization 61

2 Greedy Distributed Submodular Maximization (GreeDi) 62

3 GreeDi under General Constraints 69

4 Approximate Submodular Cover 85

5 Approximate OptCard 85

6 DisCover 87

7 FastCover 99

8 ThresholdSample 100

9 Streaming Local Search for independence systems 119

10 Streaming Local Search for independence systems $\mathcal{I}$ and d knapsacks 122

11 Robust-Streaming 140

12 Stochastic-Greedy 158

13 GDT - Greedy with density threshold 170

14 IGDT: Iterated greedy with density threshold 171

15 Fantom 171

Bibliography

[Aga+13] Pankaj K Agarwal, Graham Cormode, Zengfeng Huang, Jeff M Phillips, Zhewei Wei, and Ke Yi. “Mergeable summaries”. In: *ACM Transactions on Database Systems (TODS)* 38.4 (2013), p. 26 (cit. on p. 20).

[AMT13] Zeinab Abbassi, Vahab S Mirrokni, and Mayur Thakur. “Diversity maximization under matroid constraints”. In: *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2013, pp. 32–40 (cit. on p. 24).

[Bab+13] Mahmoudreza Babaei, Baharan Mirzasoleiman, Mahdi Jalili, and Mohammad Ali Safari. “Revenue maximization in social networks through discounting”. In: *Social Network Analysis and Mining* 3.4 (2013), pp. 1249–1262 (cit. on p. 46).

[Bad+14] Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. “Streaming submodular maximization: Massive data summarization on the fly”. In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2014, pp. 671–680 (cit. on pp. 17, 33, 34, 115, 121, 123, 141, 142, 166).

[Bal+16] Eric Balkanski, Baharan Mirzasoleiman, ETHZ CH, and Yaron Singer. “Learning sparse combinatorial representations via two-stage submodular maximization”. In: *Proceedings of The 33rd International Conference on Machine Learning*. 2016, pp. 2207–2216 (cit. on p. 17).

[Bar+15] Rafael Barbosa, Alina Ene, Huy L Nguyen, and Justin Ward. “The power of randomization: Distributed submodular maximization on massive datasets”. In: *International Conference on Machine Learning*. 2015, pp. 1236–1244 (cit. on pp. 30, 31, 67, 84, 88).

[BEM16] MohammadHossein Bateni, Hossein Esfandiari, and Vahab Mirrokni. "Distributed Coverage Maximization via Sketching". In: *arXiv preprint arXiv:1612.02327* (2016) (cit. on p. 20).

[BHZ10] MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Morteza Zadimoghaddam. "Submodular secretary problem and extensions". In: *Proceedings of the 13th international conference on Approximation, and 14 the International conference on Randomization, and combinatorial optimization: algorithms and techniques*. Berlin, Heidelberg, 2010, pp. 39–52 (cit. on p. 35).

[Bia+17] Yatao Bian, Baharan Mirzasoleiman, Joachim M Buhmann, and Andreas Krause. "Guaranteed non-convex optimization: Submodular maximization over continuous domains". In: (2017) (cit. on p. 17).

[Bod12] Ferenc Bodon. *Kosarak Dataset*. 2012. URL: http://fimi.ua.ac.be/data/ (cit. on p. 78).

[Bog+17] Ilija Bogunovic, Slobodan Mitrovic, Jonathan Scarlett, and Volkan Cevher. "Robust Submodular Maximization: A Non-Uniform Partitioning Approach". In: *The 34th International Conference on Machine Learning (ICML)*. EPFL-CONF-229153. 2017 (cit. on p. 28).

[BPT11] Guy E. Blelloch, Richard Peng, and Kanat Tangwongsan. "Linear-work greedy parallel approximate set cover and variants". In: *SPAA*. 2011 (cit. on p. 31).

[BRS89] Bonnie Berger, John Rompel, and Peter W Shor. "Efficient NC algorithms for set cover with applications to learning and geometry". In: *Foundations of Computer Science, 1989., 30th Annual Symposium on*. IEEE. 1989, pp. 54–59 (cit. on pp. 31, 32).

[Buc+] Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. "Submodular Maximization with Cardinality Constraints". In: *SODA 2014* (cit. on p. 36).

[Buc+12] Niv Buchbinder, Michael Feldman, Joseph Naor, and Roy Schwartz. "A tight linear time (1/2)-approximation for unconstrained submodular maximization". In: *53rd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 2012, pp. 649–658 (cit. on p. 26).

[Buc+14] Niv Buchbinder, Moran Feldman, Joseph Seffi Naor, and Roy Schwartz. "Submodular maximization with cardinality constraints". In: *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics. 2014, pp. 1433–1452 (cit. on pp. 24, 27, 78, 142, 218).

[Buc+15] Niv Buchbinder, Moran Feldman, Joseph Seffi, and Roy Schwartz. "A tight linear time (1/2)-approximation for unconstrained submodular maximization". In: *SIAM Journal on Computing* (2015) (cit. on pp. 36, 120, 220, 223, 229).

[BV14] Ashwinkumar Badanidiyuru and Jan Vondrák. "Fast algorithms for maximizing submodular functions". In: *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics. 2014, pp. 1497–1514 (cit. on pp. 5, 27, 35, 115, 121, 159, 166, 169).

[Cal+11] Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. "Maximizing a monotone submodular function subject to a matroid constraint". In: *SIAM Journal on Computing* 40.6 (2011), pp. 1740–1766 (cit. on pp. 24, 25, 27).

[CC84] 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". In: *Discrete Applied Mathematics* 7.3 (1984), pp. 251–274 (cit. on p. 25).

[CGQ15] Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. "Streaming Algorithms for Submodular Function Maximization". In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2015, pp. 318–330 (cit. on pp. 33, 34, 115, 120–123, 141, 142, 220, 223).

[Chi+15] Flavio Chierichetti, Alessandro Epasto, Ravi Kumar, Silvio Lattanzi, and Vahab Mirrokni. "Efficient algorithms for public-private social networks". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2015, pp. 139–148 (cit. on pp. 93, 94).

[Chu+07] Cheng Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Andrew Y Ng, and Kunle Olukotun. "Map-reduce for machine learning on multicore". In: *Advances in Neural Information Processing Systems* 19 (2007), p. 281 (cit. on p. 29).

[CJV15] Chandra Chekuri, TS Jayram, and Jan Vondrák. "On multiplicative weight updates for concave and submodular function maximization". In: *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*. ACM. 2015, pp. 201–210 (cit. on p. 37).

[CK14] Amit Chakrabarti and Sagar Kale. "Submodular maximization meets streaming: Matchings, matroids, and more". In: *Mathematical Programming* 154.1-2 (2014), pp. 225–247 (cit. on pp. 33, 34, 115, 141).

[CKT10] Flavio Chierichetti, Ravi Kumar, and Andrew Tomkins. "Max-cover in map-reduce". In: *Proceedings of the 19th International Conference on World Wide Web*. ACM. 2010, pp. 231–240 (cit. on pp. 32, 47).

[CNZ16] Jiecao Chen, Huy L. Nguyen, and Qin Zhang. "Submodular Maximization over Sliding Windows". In: *arXiv preprint arXiv:1611.00129* (2016) (cit. on p. 34).

[CR09] Emmanuel J Candès and Benjamin Recht. "Exact matrix completion via convex optimization". In: *Foundations of Computational mathematics* (2009) (cit. on p. 96).

[CR12] Emmanuel Candes and Benjamin Recht. "Exact matrix completion via convex optimization". In: *Communications of the ACM* 55.6 (2012), pp. 111–119 (cit. on p. 45).

[CVZ] Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. "Submodular Function Maximization via the Multilinear Relaxation and Contention Resolution Schemes". In: *SIAM J. Comput. 2014* () (cit. on pp. 36, 37).

[DA+11] Sandra Eliza Fontes De Avila, Ana Paula Brandão Lopes, Antonio da Luz, and Arnaldo de Albuquerque Araújo. "VSUMM: A mechanism designed to produce static video summaries and a novel evaluation method". In: *Pattern Recognition Letters* 32.1 (2011), pp. 56–68 (cit. on p. 124).

[DF07a] Delbert Dueck and Brendan J. Frey. "Non-metric affinity propagation for unsupervised image categorization". In: *ICCV*. 2007 (cit. on p. 3).

[DF07b] Delbert Dueck and Brendan J Frey. “Non-metric affinity propagation for unsupervised image categorization”. In: *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*. IEEE. 2007, pp. 1–8 (cit. on pp. 46, 166).

[DG08] Jeffrey Dean and Sanjay Ghemawat. “MapReduce: simplified data processing on large clusters”. In: *Communications of the ACM* 51.1 (2008), pp. 107–113 (cit. on pp. 5, 29, 55, 82).

[DK11] Abhimanyu Das and David Kempe. “Submodular meets Spectral: Greedy Algorithms for Subset Selection, Sparse Approximation and Dictionary Selection”. In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 2011, pp. 1057–1064 (cit. on p. 81).

[DKR13a] Anirban Dasgupta, Ravi Kumar, and Sujith Ravi. “Summarization Through Submodularity and Dispersion”. In: *ACL*. 2013 (cit. on p. 3).

[DKR13b] Anirban Dasgupta, Ravi Kumar, and Sujith Ravi. “Summarization Through Submodularity and Dispersion”. In: *ACL*. 2013 (cit. on pp. 165, 166).

[Du+13] Nan Du, Yingyu Liang, Maria Florina Balcan, and Le Song. “Budgeted influence maximization for multiple products”. In: *arXiv preprint arXiv:1312.2164* (2013) (cit. on pp. 21, 25, 27).

[DVV03] Sven De Vries and Rakesh V Vohra. “Combinatorial auctions: A survey”. In: *INFORMS Journal on Computing* 15.3 (2003), pp. 284–309 (cit. on p. 47).

[EA+09] Khalid El-Arini, Gaurav Veda, Dafna Shahaf, and Carlos Guestrin. “Turning Down the Noise in the Blogosphere”. In: *KDD*. Paris, France, 2009 (cit. on pp. 3, 21).

[EAG11] Khalid El-Arini and Carlos Guestrin. “Beyond Keyword Search: Discovering Relevant Scientific Literature”. In: *KDD*. 2011 (cit. on pp. 3, 166).

[Epa+16] Alessandro Epasto, Silvio Lattanzi, Sergei Vassilvitskii, and Morteza Zadimoghaddam. “Submodular Optimization over Sliding Windows”. In: *arXiv preprint arXiv:1610.09984* (2016) (cit. on p. 33).

[EPF08] Jaliya Ekanayake, Shrideep Pallickara, and Geoffrey Fox. “Mapreduce for data intensive scientific analyses”. In: *IEEE Fourth International Conference on eScience*. IEEE. 2008, pp. 277–284 (cit. on p. 29).

[EU15] Council of the European Union. http://data.consilium.europa.eu/doc/document/ST-9565-2015-INIT/en/pdf. 2015 (cit. on p. 131).

[Fat15] Philipe Fatio. https://refind.com/fphilipe/topics/open-data. 2015 (cit. on pp. 11, 12, 144).

[Fei98] Uriel Feige. "A threshold of ln n for approximating set cover". In: *Journal of the ACM* (1998) (cit. on pp. 4, 22, 23, 82, 83).

[FHK17] Moran Feldman, Christopher Harshaw, and Amin Karbasi. "Greed is Good: Near-Optimal Submodular Maximization via Greedy Optimization". In: *arXiv preprint arXiv:1704.01652* (2017) (cit. on pp. 130, 219).

[FMV11] Uriel Feige, Vahab S Mirrokni, and Jan Vondrak. "Maximizing non-monotone submodular functions". In: *SIAM Journal on Computing* (2011) (cit. on pp. 36, 118).

[FNS11a] Moran Feldman, Joseph Naor, and Roy Schwartz. "A unified continuous greedy algorithm for submodular maximization". In: *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*. IEEE. 2011, pp. 570–579 (cit. on pp. 24, 25, 27, 33).

[FNS11b] Moran Feldman, Joseph Naor, and Roy Schwartz. "Nonmonotone Submodular Maximization via a Structural Continuous Greedy Algorithm - (Extended Abstract)". In: *ICALP (1)*. 2011, pp. 342–353 (cit. on p. 36).

[FNW78] Marshall L. Fisher, George L. Nemhauser, and Laurence A. Wolsey. "An analysis of approximations for maximizing submodular set functions - II". In: *Mathematical Programming Study* 8 (1978), pp. 73–87 (cit. on pp. 24, 25, 27).

[FPZ03] Robert Fergus, Pietro Perona, and Andrew Zisserman. "Object class recognition by unsupervised scale-invariant learning". In: *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*. Vol. 2. IEEE. 2003, pp. II–II (cit. on p. 44).

[Geu+03] Karolien Geurts, Geert Wets, Tom Brijs, and Koen Vanhoof. "Profiling of high-frequency accident locations by use of association rules". In: *Transportation Research Record: Journal of the Transportation Research Board* 1840 (2003), pp. 123–130 (cit. on p. 78).

[GK04] Michael B Greenwald and Sanjeev Khanna. “Power-conserving computation of order-statistics over sensor networks”. In: *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM. 2004, pp. 275–285 (cit. on p. 20).

[GK10] Ryan Gomes and Andreas Krause. “Budgeted Nonparametric Learning from Data Streams”. In: *ICML*. 2010 (cit. on pp. 3, 32, 46, 142, 166).

[GKP01] Rahul Garg, Vijay Kumar, and Vinayaka Pandit. “Approximation algorithms for budget-constrained auctions”. In: *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques*. Springer, 2001, pp. 102–113 (cit. on p. 26).

[GKT12a] Jennifer Gillenwater, Alex Kulesza, and Ben Taskar. “Discovering diverse and salient threads in document collections”. In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics. 2012, pp. 710–720 (cit. on p. 41).

[GKT12b] Jennifer Gillenwater, Alex Kulesza, and Ben Taskar. “Near-optimal map inference for determinantal point processes”. In: *Advances in Neural Information Processing Systems*. 2012, pp. 2735–2743 (cit. on p. 21).

[GNR15] Anupam Gupta, Viswanath Nagarajan, and R Ravi. “Robust and MaxMin Optimization under Matroid and Knapsack Uncertainty Sets”. In: *ACM Transactions on Algorithms (TALG)* 12.1 (2015), p. 10 (cit. on p. 36).

[Gon+14] Boqing Gong, Wei-Lun Chao, Kristen Grauman, and Fei Sha. “Diverse sequential subset selection for supervised video summarization”. In: *Advances in Neural Information Processing Systems*. 2014, pp. 2069–2077 (cit. on pp. 41, 123–127, 129).

[GRLK10] Manuel Gomez Rodriguez, Jure Leskovec, and Andreas Krause. “Inferring networks of diffusion and influence”. In: *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2010, pp. 1019–1028 (cit. on p. 47).

[Gup+10a] Anupam Gupta, Aaron Roth, Grant Schoenebeck, and Kunal Talwar. “Constrained non-monotone submodular maximization: Offline and secretary algorithms”. In: *Internet and Network Economics*. Springer, 2010, pp. 246–257 (cit. on pp. 26, 27, 30, 35, 118, 119).

[Gup+10b] Anupam Gupta, Aaron Roth, Grant Schoenebeck, and Kunal Talwar. "Constrained Non-monotone Submodular Maximization: Offline and Secretary Algorithms". In: *WINE*. 2010 (cit. on pp. 36, 37, 169, 171, 229).

[HMS08] Jason Hartline, Vahab Mirrokni, and Mukund Sundararajan. "Optimal marketing strategies over social networks". In: *Proceedings of the 17th International Conference on World Wide Web*. ACM. 2008, pp. 189–198 (cit. on pp. 14, 46).

[HPM04] Sariel Har-Peled and Soham Mazumdar. "On coresets for k-means and k-median clustering". In: *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*. ACM. 2004, pp. 291–300 (cit. on p. 20).

[IB13] Rishabh K Iyer and Jeff A Bilmes. "Submodular optimization with submodular cover and submodular knapsack constraints". In: *Advances in Neural Information Processing Systems*. 2013, pp. 2436–2444 (cit. on pp. 84, 93, 96).

[KG05a] Andreas Krause and Carlos Guestrin. *A Note on the Budgeted Maximization on Submodular Functions*. Tech. rep. CMU-CALD-05-103. Carnegie Mellon University, 2005 (cit. on p. 26).

[KG05b] Andreas Krause and Carlos Guestrin. "Near-optimal Nonmyopic Value of Information in Graphical Models". In: *Proceedings of Uncertainty in Artificial Intelligence (UAI)*. 2005, p. 5 (cit. on pp. 23, 26, 40).

[KG10] Andreas Krause and Ryan G Gomes. "Budgeted nonparametric learning from data streams". In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 2010, pp. 391–398 (cit. on pp. 9, 21, 42, 66).

[KG11] Andreas Krause and Carlos Guestrin. "Submodularity and its applications in optimized information gathering". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 2.4 (2011), p. 32 (cit. on p. 21).

[KG13] Andreas Krause and Daniel Golovin. "Submodular Function Maximization". In: *Tractability: Practical Approaches to Hard Problems*. Cambridge University Press, 2013 (cit. on pp. 3, 22, 58, 135).

[KH09] Alex Krizhevsky and Geoffrey Hinton. "Learning multiple layers of features from tiny images". In: (2009) (cit. on p. 177).

[KKT03] David Kempe, Jon Kleinberg, and Éva Tardos. "Maximizing the spread of influence through a social network". In: *Proceedings of the ninth ACM SIGKDD*. 2003 (cit. on pp. 20, 81).

[KLQ95] Chun-Wa Ko, Jon Lee, and Maurice Queyranne. "An exact algorithm for maximum entropy sampling". In: *Operations Research* 43.4 (1995), pp. 684–691 (cit. on p. 41).

[KR09] Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. Vol. 344. Wiley-Interscience, 2009 (cit. on pp. 9, 41).

[Kra+08a] Andreas Krause, Jure Leskovec, Carlos Guestrin, Jeanne VanBriesen, and Christos Faloutsos. "Efficient Sensor Placement Optimization for Securing Large Water Distribution Networks". In: *Journal of Water Resources Planning and Management* 134.6 (2008), pp. 516–526 (cit. on pp. 11, 12, 43, 81).

[Kra+08b] Andreas Krause, H Brendan McMahan, Carlos Guestrin, and Anupam Gupta. "Robust submodular observation selection". In: *Journal of Machine Learning Research* 9.Dec (2008), pp. 2761–2801 (cit. on pp. 28, 93, 96, 97).

[KST09] Ariel Kulik, Hadas Shachnai, and Tami Tamir. "Maximizing submodular set functions subject to multiple linear constraints". In: *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics. 2009, pp. 545–554 (cit. on pp. 26, 27).

[KSV10] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. "A model of computation for MapReduce". In: *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics. 2010, pp. 938–948 (cit. on p. 29).

[KT+12] Alex Kulesza, Ben Taskar, et al. "Determinantal point processes for machine learning". In: *Foundations and Trends® in Machine Learning* 5.2–3 (2012), pp. 123–286 (cit. on pp. 41, 118).

[Kum+13] Ravi Kumar, Benjamin Moseley, Sergei Vassilvitskii, and Andrea Vattani. "Fast greedy algorithms in MapReduce and streaming". In: *Proceedings of the 25th ACM Symposium on Parallelism in Algorithms and Architectures*. ACM. 2013, pp. 1–10 (cit. on pp. 29–34, 78, 79, 166).

[Lat+11] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. "Filtering: a method for solving graph problems in mapreduce". In: *Proceedings of the Twenty-Third Annual ACM Symposium on Parallelism in Algorithms and Architectures*. ACM. 2011, pp. 85–94 (cit. on p. 42).

[LB11a] Hui Lin and Jeff Bilmes. "A class of submodular functions for document summarization". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics. 2011, pp. 510–520 (cit. on pp. 3, 21, 35, 156).

[LB11b] Hui Lin and Jeff Bilmes. "A class of submodular functions for document summarization". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics. 2011, pp. 510–520 (cit. on pp. 26, 45, 47, 166).

[LBK16] Mario Lucic, Olivier Bachem, and Andreas Krause. "Strong coresets for hard and soft Bregman clustering with applications to exponential family mixtures". In: *International Conference on Artificial Intelligence and Statistics*. 2016 (cit. on p. 20).

[Lee+09] Jon Lee, Vahab S Mirrokni, Viswanath Nagarajan, and Maxim Sviridenko. "Non-monotone submodular maximization under matroid and knapsack constraints". In: *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*. ACM. 2009, pp. 323–332 (cit. on pp. 26, 27, 130).

[Les+07] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. "Cost-effective outbreak detection in networks". In: *KDD '07: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. San Jose, California, USA: ACM, 2007, pp. 420–429 (cit. on pp. 20, 23, 47, 55, 58, 156).

[LGG12] Yong Jae Lee, Joydeep Ghosh, and Kristen Grauman. "Discovering important people and objects for egocentric video summarization". In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE. 2012, pp. 1346–1353 (cit. on p. 129).

[LK06] Tiecheng Liu and John Kender. “Optimization algorithms for the selection of key frame sequences of variable length”. In: *Computer Vision—ECCV 2002* (2006), pp. 301–305 (cit. on p. 129).

[LLS07] John Langford, Lihong Li, and Alex Strehl. *Vowpal wabbit online learning project*. 2007 (cit. on p. 146).

[Low04] David G Lowe. “Distinctive image features from scale-invariant keypoints”. In: *International journal of computer vision* 60.2 (2004), pp. 91–110 (cit. on p. 124).

[Low99] David G Lowe. “Object recognition from local scale-invariant features”. In: *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*. Vol. 2. Ieee. 1999, pp. 1150–1157 (cit. on p. 44).

[LSV09] Jon Lee, Maxim Sviridenko, and Jan Vondrák. “Submodular maximization over multiple matroids via generalized exchange properties”. In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer, 2009, pp. 244–257 (cit. on pp. 25, 27).

[LSV10] Jon Lee, Maxim Sviridenko, and Jan Vondrák. “Submodular Maximization over Multiple Matroids via Generalized Exchange Properties”. In: *Math. Oper. Res.* (2010) (cit. on pp. 36, 37).

[LWD15] Erik M Lindgren, Shanshan Wu, and Alexandros G Dimakis. “Sparse and Greedy: Sparsifying Submodular Facility Location Problems”. In: *NIPS* (2015) (cit. on pp. 104, 173).

[Mac75] Odile Macchi. “The coincidence approach to stochastic point processes”. In: *Advances in Applied Probability* 7.01 (1975), pp. 83–122 (cit. on p. 40).

[MBK16] Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, and Amin Karbasi. “Fast constrained submodular maximization: Personalized data summarization”. In: *ICLM'16: Proceedings of the 33rd International Conference on Machine Learning (ICML)*. 2016 (cit. on p. 16).

[Min78] M. Minoux. “Accelerated greedy algorithms for maximizing submodular set functions”. In: *Optimization Techniques, LNCS* (1978), pp. 234–243 (cit. on pp. 5, 8, 23, 55, 58, 115, 156, 158, 159).

[Mir+13] Baharan Mirzasoleiman, Amin Karbasi, Rik Sarkar, and Andreas Krause. "Distributed submodular maximization: Identifying representative elements in massive data". In: *Advances in Neural Information Processing Systems*. 2013, pp. 2049–2057 (cit. on pp. 16, 166).

[Mir+15a] Baharan Mirzasoleiman, Amin Karbasi, Ashwinkumar Badanidiyuru, and Andreas Krause. "Distributed submodular cover: Succinctly summarizing massive data". In: *Advances in Neural Information Processing Systems*. 2015, pp. 2881–2889 (cit. on p. 16).

[Mir+15b] Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrak, and Andreas Krause. "Lazier Than Lazy Greedy". In: *Twenty-Ninth AAAI Conference on Artificial Intelligence*. 2015 (cit. on p. 16).

[Mir+16] Baharan Mirzasoleiman, Amin Karbasi, Rik Sarkar, and Andreas Krause. "Distributed submodular maximization". In: *Journal of Machine Learning Research* 17.238 (2016), pp. 1–44 (cit. on p. 16).

[MJK18] Baharan Mirzasoleiman, Stefanie Jegelka, and Andreas Krause. "Streaming Non-monotone Submodular Maximization: Personalized Video Summarization on the Fly". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018 (cit. on p. 16).

[MKK17] Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. "Deletion Robust Submodular Maximization: Data Summarization with the Right to be Forgotten". In: *Proceedings of The 33rd International Conference on Machine Learning*. 2017 (cit. on p. 16).

[Mov] *GroupLens. MovieLens 20M dataset*. `http://grouplens.org/datasets/movielens/20m/`. 2015 (cit. on pp. 13, 104, 173).

[MZ15] Vahab Mirrokni and Morteza Zadimoghaddam. "Randomized Composable Core-sets for Distributed Submodular Maximization". In: *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*. STOC '15. Portland, Oregon, USA: ACM, 2015, pp. 153–162 (cit. on pp. 30, 31, 67, 84, 88).

[MZK16] Baharan Mirzasoleiman, Morteza Zadimoghaddam, and Amin Karbasi. "Fast Distributed Submodular Cover: Public-Private Data Summarization". In: *Advances in Neural Information Processing Systems*. 2016, pp. 3594–3602 (cit. on pp. 16, 130).

[NMZ03] Chong-Wah Ngo, Yu-Fei Ma, and Hong-Jiang Zhang. "Automatic video summarization by graph modeling". In: *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*. IEEE. 2003, pp. 104–109 (cit. on p. 129).

[NN12] Ramasuri Narayanam and Amit A Nanavati. "Viral marketing for product cross-sell through social networks". In: *Machine Learning and Knowledge Discovery in Databases*. Springer, 2012, pp. 581–596 (cit. on p. 24).

[NW78] G. L. Nemhauser and L. A. Wolsey. "Best algorithms for approximating the maximum of a submodular set function". In: *Math. Oper. Research* (1978) (cit. on pp. 4, 23).

[NWF78a] George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. "An analysis of approximations for maximizing submodular set functions - I". In: *Mathematical Programming* (1978) (cit. on pp. 3, 21, 22, 55, 115, 193).

[NWF78b] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. "An analysis of approximations for maximizing submodular set functions—I". In: *Mathematical Programming* 14.1 (1978), pp. 265–294 (cit. on pp. 33, 130).

[OP09] Tore Opsahl and Pietro Panzarasa. "Clustering in weighted networks". In: *Social networks* 31.2 (2009), pp. 155–163 (cit. on p. 77).

[Ost+08] Avi Ostfeld, James G Uber, Elad Salomons, Jonathan W Berry, William E Hart, Cindy A Phillips, Jean-Paul Watson, Gianluca Dorini, Philip Jonkergouw, Zoran Kapelan, et al. "The battle of the water sensor networks (BWSN): A design challenge for engineers and algorithms". In: *Journal of Water Resources Planning and Management* 134.6 (2008), pp. 556–568 (cit. on pp. 12, 163).

[OSU16] James B Orlin, Andreas S Schulz, and Rajan Udwani. "Robust Monotone Submodular Function Maximization". In: *Proceedings of the 18th International Conference on Integer Programming and Combinatorial Optimization-Volume 9682*. Springer-Verlag New York, Inc. 2016, pp. 312–324 (cit. on pp. 28, 137).

[PD07] Florent Perronnin and Christopher Dance. "Fisher kernels on visual vocabularies for image categorization". In: *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE. 2007, pp. 1–8 (cit. on p. 124).

[Rah+10] Esa Rahtu, Juho Kannala, Mikko Salo, and Janne Heikkilä. "Segmenting salient objects from images and videos". In: *Computer Vision–ECCV 2010* (2010), pp. 366–379 (cit. on p. 124).

[Reg12] European Data Protection Regulation. `http://ec.europa.eu/justice/data-protection/document/review2012/com_2012_11_en.pdf`. 2012 (cit. on p. 131).

[RG13] Colorado Reed and Zoubin Ghahramini. "Scaling the Indian Buffet Process via Submodular Maximization". In: *ICML*. 2013 (cit. on p. 21).

[RLK12] Manuel Gomez Rodriguez, Jure Leskovec, and Andreas Krause. "Inferring Networks of Diffusion and Influence". In: *ACM Transactions on Knowledge Discovery from Data* 5.4 (2012), 21:1–21:37 (cit. on p. 156).

[RW06] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. 2006 (cit. on p. 10).

[See04] Matthias Seeger. *Greedy Forward Selection in the Informative Vector Machine*. Tech. rep. University of California, Berkeley, 2004 (cit. on p. 10).

[SGK09] Matthew Streeter, Daniel Golovin, and Andreas Krause. "Online learning of assignments". In: *Advances in Neural Information Processing Systems*. 2009, pp. 1794–1802 (cit. on p. 24).

[Sin+14] Adish Singla, Ilija Bogunovic, Gabor Bartok, Amin Karbasi, and Andreas Krause. "Near-Optimally Teaching the Crowd to Classify." In: *ICML*. 2014, pp. 154–162 (cit. on pp. 21, 166).

[Sip+12a] Ruben Sipos, Adith Swaminathan, Pannaga Shivaswamy, and Thorsten Joachims. "Temporal Corpus Summarization Using Submodular Word Coverage". In: *CIKM*. Maui, Hawaii, USA, 2012 (cit. on pp. 3, 21).

[Sip+12b] Ruben Sipos, Adith Swaminathan, Pannaga Shivaswamy, and Thorsten Joachims. "Temporal corpus summarization using submodular word coverage". In: *CIKM*. 2012 (cit. on p. 166).

[SS01] Bernhard Scholkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001 (cit. on p. 166).

[SSS07] Ian Simon, Noah Snavely, and Steven M Seitz. "Scene summarization for online image collections". In: *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*. IEEE. 2007, pp. 1–8 (cit. on pp. 45, 166).

[ST15] Stergios Stergiou and Kostas Tsioutsiouliklis. "Set cover at web scale". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2015, pp. 1125–1133 (cit. on p. 31).

[Svi04] Maxim Sviridenko. "A Note on Maximizing a Submodular Set Function Subject to Knapsack Constraint". In: *Operations Research Letters* 32 (2004) (cit. on pp. 26, 27, 121).

[TFF08] Antonio Torralba, Rob Fergus, and William T Freeman. "80 million tiny images: A large data set for nonparametric object and scene recognition". In: *IEEE Trans. Pattern Anal. Mach. Intell.* (2008) (cit. on pp. 9, 57, 72, 74, 89, 160).

[TKC05] Ivor W Tsang, James T Kwok, and Pak-Ming Cheung. "Core vector machines: Fast SVM training on very large data sets". In: *Journal of Machine Learning Research* 6.Apr (2005), pp. 363–392 (cit. on p. 20).

[Tsa+10] Athanasios Tsanas, Max A Little, Patrick E McSharry, and Lorraine O Ramig. "Enhanced classical dysphonia measures and sparse regression for telemonitoring of Parkinson's disease progression". In: *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*. IEEE. 2010, pp. 594–597 (cit. on pp. 74, 90, 102, 160).

[Tsc+14] Sebastian Tschiatschek, Rishabh K Iyer, Haochen Wei, and Jeff A Bilmes. "Learning mixtures of submodular functions for image collection summarization". In: *Advances in neural information processing systems*. 2014, pp. 1413–1421 (cit. on pp. 43, 143, 165, 166).

[Uij+13] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. "Selective search for object recognition". In: *International journal of computer vision* 104.2 (2013), pp. 154–171 (cit. on p. 129).

[Vit85] Jeffrey S Vitter. "Random sampling with a reservoir". In: *ACM Transactions on Mathematical Software (TOMS)* 11.1 (1985), pp. 37–57 (cit. on p. 123).

[VJ04] Paul Viola and Michael J Jones. "Robust real-time face detection". In: *International journal of computer vision* 57.2 (2004), pp. 137–154 (cit. on p. 128).

[WDJ11] Christian Wengert, Matthijs Douze, and Hervé Jégou. "Bag-of-colors for improved image search". In: *Proceedings of the 19th ACM international conference on Multimedia*. ACM. 2011, pp. 1437–1440 (cit. on p. 44).

[Web11] Rolf H Weber. "The right to be forgotten". In: *More than a Pandora's Box* 2 (2011) (cit. on p. 132).

[Wei+13] Kai Wei, Yuzong Liu, Katrin Kirchhoff, and Jeff Bilmes. "Using document summarization techniques for speech data subset selection". In: *Proceedings of NAACL-HLT*. 2013, pp. 721–726 (cit. on pp. 35, 47, 156).

[WIB14] Kai Wei, Rishabh K Iyer, and Jeff A Bilmes. "Fast Multi-stage Submodular Maximization". In: *ICML*. 2014, pp. 1494–1502 (cit. on pp. 35, 166).

[Wol82] Laurence A. Wolsey. "An analysis of the greedy algorithm for the submodular set covering problem". In: *Combinatorica* (1982) (cit. on pp. 22, 82, 83).

[Yah12] Yahoo. *Yahoo! Academic Relations. R6A, Yahoo! Front Page Today Module User Click Log Dataset, Version 1.0*. 2012. URL: http://Webscope.sandbox.yahoo.com (cit. on pp. 10, 11, 74, 144).

[YL15] Jaewon Yang and Jure Leskovec. "Defining and evaluating network communities based on ground-truth". In: *Knowledge and Information Systems* 42.1 (2015), pp. 181–213 (cit. on pp. 10, 14, 90, 105, 176).

[Zah+10] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. "Spark: Cluster Computing with Working Sets." In: *HotCloud* 10.10-10 (2010), p. 95 (cit. on pp. 74, 82, 90, 94, 101).