Leveraging Structural Properties for Large-Scale Optimization

A Dissertation Presented
by

Armin Moharrer

to

The Department of Electrical and Computer Engineering

in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

Northeastern University
Boston, Massachusetts

April 2021

To Haleh and Mohammadreza.

Contents

List of Figures v

List of Tables vi

List of Acronyms vii

Acknowledgments viii

Abstract of the Dissertation ix

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Contributions . . . 3
  - 1.2.1 Distributing Frank-Wolfe via Map-Reduce . . . 3
  - 1.2.2 Design of Kelly Cache Networks via Submodular Maximization . . . 4
  - 1.2.3 Massively Distributed Graph Distances . . . 7
  - 1.2.4 Robust Regression via Model-Based Optimization . . . 9
- 1.3 Conclusion . . . 11

2 Technical Preliminary 13
- 2.1 Optimization . . . 13
  - 2.1.1 Constrained Optimization and Optimality Conditions . . . 16
- 2.2 Methods for Convex Optimization . . . 17
  - 2.2.1 Stochastic Gradient Descent . . . 17
  - 2.2.2 Frank-Wolfe . . . 18
  - 2.2.3 Alternating Directions Method of Multipliers . . . 19
  - 2.2.4 Distributed Stochastic Dual Coordinate Ascent . . . 21
  - 2.2.5 Bi-section Method . . . 22
- 2.3 Submodular Maximization . . . 24
  - 2.3.1 Set Functions and Submodularity. . . . 25
  - 2.3.2 Problem Statement . . . 26
  - 2.3.3 Continuous Greedy Algorithm . . . 26
  - 2.3.4 Continuous Greedy Algorithm and Frank-Wolfe . . . 27
  - 2.3.5 DR-submodular Functions . . . 28

2.4 Parallel and Distributed Computing Frameworks 29
2.4.1 Map-Reduce Framework 29
2.4.2 Message Passing Interface (MPI) 30
2.4.3 Open Multi-Processing (OpenMP) 31

3 Frank-Wolfe via Map-Reduce 32
3.1 Technical Background 34
3.1.1 Frank-Wolfe Algorithm 34
3.1.2 FW Variants 36
3.1.3 Distributed implementations 39
3.1.4 Frank-Wolfe Over the Simplex 40
3.2 Frank-Wolfe via Map-Reduce 41
3.2.1 Gradient Computation through Common Information 41
3.2.2 A Serial Algorithm 43
3.2.3 Parallelization Through Map-Reduce 44
3.2.4 Selecting the step size. 45
3.3 Examples 46
3.4 Extensions 50
3.5 Implementation 51
3.6 Experiments 52
3.6.1 Experiment Setup 52
3.6.2 Serial Execution 54
3.6.3 Effect of Parallelism 57
3.6.4 Subsampling the Gradient 59
3.6.5 LASSO Experiment 62
3.7 Conclusion 62

4 Design of Kelly Cache Networks via Submodular Maximization 65
4.1 Model 67
4.1.1 Kelly Cache Networks 67
4.1.2 Cache Optimization 71
4.2 Submodularity of Cache Optimization and Algorithms 73
4.2.1 Continuous-Greedy Algorithm 76
4.2.2 A Novel Estimator via Taylor Expansion 79
4.2.3 The Role of Sparsity in Efficient Computations 86
4.3 Beyond M/M/1 queues 86
4.4 Experiments 89
4.4.1 Caching Gain Across Different Topologies. 92
4.4.2 Varying Service Rates. 93
4.4.3 Effect of Congestion on Caching Gain. 94
4.4.4 Varying Caching Capacity. 94
4.5 Conclusion and Future Work 95

5 Massively Distributed Graph Distances **97**
5.1 Graph Distances . . . 100
5.1.1 The Weisfeiler-Lehman (WL) Algorithm . . . 101
5.1.2 Graph Distances. . . . 102
5.1.3 Metrics. . . . 103
5.1.4 Convex Relaxation . . . 103
5.1.5 Constraints and Node Features . . . 104
5.1.6 Importance of p-norms and Linear Term . . . 106
5.2 Proximal Operators and Consensus ADMM . . . 106
5.2.1 Proximal Operators. . . . 106
5.2.2 Consensus ADMM . . . 107
5.3 Distributed Graph Distances via ADMM . . . 109
5.3.1 Distributing Consensus ADMM for $p > 1$. . . . 111
5.3.2 Parallel p-norm Proximal Operator . . . 113
5.3.3 Proof of Theorem 5.3.2 . . . 116
5.3.4 Proof of Theorem 5.3.3 . . . 118
5.4 Parallel Complexity . . . 120
5.4.1 Characterizing the Sparsity of $\mathcal{G}$ . . . 121
5.5 Experiments . . . 123
5.5.1 Experimental Setup . . . 123
5.5.2 Linear Term . . . 124
5.5.3 p-norms . . . 125
5.5.4 Scalability . . . 128
5.5.5 Real Graph Pairs . . . 130
5.6 Conclusion . . . 131

6 Robust Regression via Model Based Optimization **132**
6.1 Robust Regression and Applications . . . 135
6.2 Robust Regression via MBO . . . 136
6.2.1 MBO . . . 137
6.3 Stochastic Alternating Direction Method of Multipliers . . . 139
6.3.1 OADM . . . 140
6.3.2 SADM . . . 140
6.3.3 Inner ADMM . . . 141
6.3.4 Convergence . . . 142
6.3.5 Proof of Theorem 6.3.1 . . . 144
6.4 Experiments . . . 150
6.4.1 Time and Objective Performance Comparison . . . 153
6.4.2 Robustness Analysis . . . 154
6.4.3 Classification Performance . . . 156
6.5 Conclusion . . . 158

Bibliography **159**

List of Figures

1.1 Illustrations of a cache network . . . 5
1.2 An example of two isomorphic graphs . . . 8
1.3 Chapter Dependencies. . . . 12

3.1 Objective Trajectories . . . 56
3.2 Speedups for Different Datasets . . . 59
3.3 Objective Trajectories Across Different Problems . . . 60
3.4 Experiments on Real Data . . . 60
3.5 Results for Stochastic Algorithms . . . 61
3.6 Comparison Between ADMM and our Distributed FW . . . 63

4.1 Illustration of a classic Kelly Network and a Kelly cache Network . . . 68
4.2 Counter-example . . . 76
4.3 A simple network with finite-capacity queues. . . . 88
4.4 The `abilene` topology. . . . 90
4.5 Caching gains for different topologies and different arrival distributions . . . 91
4.6 Running time for different topologies and power-law arrival distribution . . . 91
4.7 Caching gain vs. M . . . 93
4.8 Caching gain vs. arrival rate . . . 94
4.9 Caching gain vs. cache capacity . . . 95

5.1 Clustering experiment using metrics and non-metrics (from Bento and Ioannidis [1]). 98
5.2 A bipartite graph showing showing dependencies in consensus ADMM . . . 108
5.3 Effects of adding the linear term on the convergence. . . . 125
5.4 Effects of adding linear term on solutions in the case of `BRN` . . . 127
5.5 Traces of our ADMM algorithm for $p = 2$ and real graphs pairs. . . . 130

6.1 Robustness of ℓ_p norms vs. MSE to outliers introduced to `MNIST` when training an autoencoder. . . . 133
6.2 A comparison of scalability of the non-outliers loss $F_{\texttt{NOUT}}$ for different p-norms . . . 154
6.3 A comparison of scalability of the test loss $F_{\texttt{TEST}}$ for different p-norms . . . 155
6.4 Classification performance for different methods and datasets . . . 156

List of Tables

1.1 Summary of problems and the role of sparsity in obtaining efficient algorithms. . . 12

2.1 Properties of DR-submodular functions 28

3.1 Examples of problems satisfying Prop. 1–3 46
3.2 Dataset A 55
3.3 Dataset B 55
3.4 Dataset C 55
3.5 Real Datasets 55
3.6 A summary of speedups in comparison with three serial implementations 58

4.1 Results of $\rho_{u,v}(\mathbf{x})$'s for different caching configurations. 88
4.2 Graph Topologies and Experiment Parameters 90

5.1 A summary of real graph pairs along with the preprocessing method for generating $\mathcal{Q}$. 124
5.2 Comparison of different p-norms and λ coefficients for BRN, OUT1, and OUT2 . . 126
5.3 Comparison of different p-norms and λ coefficients for GSS LPC, and MIX 126
5.4 Scaling results for Alg. 8 129
5.5 Scaling results for Alg. 8 129

6.1 Time and Objective Performance 151
6.2 Accuracy of the classifiers for MNIST dataset. 157
6.3 Accuracy of the classifiers for Fashion-MNIST dataset. 158

List of Acronyms

FW The Frank-Wolfe algorithm. The optimization algorithm explained in Section 3.1.

ADMM Alternating Directions Method of Multipliers. An optimization algorithm for convex optimization problems (see Section 2.2.3).

SGD Stochastic Gradient Descent. A stochastic optimization algorithm for convex optimization problems, well-suited for problems with separable objectives (see Section 2.2.1).

DSDCA Distributed Stochastic Dual Coordinate Ascent. A distributed stochastic optimization algorithm for convex optimization problems, which solves problems in their dual domain (see Section 2.2.4).

MPI Message Passing Interface. A distributed framework for expressing parallel algorithms.

WL The Weisfeiler-Lehman Algorithm. An algorithm for assigning labels to nodes of a graph based on their neighbor size (see Sec. 5.1.1)

MBO Model-Based Optimization. A class of optimization algorithms for non-smooth non-convex problems (see Sec. 6.2)

OADM Online Alternating Direction Method. An online variant of ADMM (see Sec. 6.3.1)

SADM Stochastic Alternating Direction Method. A stochastic variant of ADMM (see Sec. 6.3.2)

Acknowledgments

Here I wish to express gratitude toward those who supported me, in any way, during this long journey. First and foremost, I would like to thank my parents for always believing in me and supporting me. I thank them for inspiring me to follow my ambitions and work hard, and-above all-to be kind.

I also like to deeply thank Prof. Stratis Ioannidis for believing in me, and teaching me so much about academic research. I started my Ph.D. fresh with an undergraduate degree without any serious research background. However, Stratis was extremely patient with my progress and made anything possible to make my transition from undergraduate to the Ph.D. program smooth. I have always found his level of dedication, attention to details, and obsession with high quality work-though overwhelming at times-quite inspiring.

I also thank my committee members Prof. David Kaeli and Prof. Jennifer Dy. I spent one of my first classes, Introduction to Machine Learning, with Prof. Dy and she made the class accessible for all levels. She also warmly welcomed me into her reading groups and has always been supportive. The same is true for Prof. Kaeli who has been a mentor for me. I had one of my best coursework experiences when taking his High Performance Computing class in Fall 2017; I truly enjoyed the class, and despite the fact that the course content was different from my background, Dave was quite helpful and made following the course comfortable.

I thank my collaborators Gözde Özcan, Jasmin Gao, Milad Mahdian, and Khashayar Kamran. I had a great experience working with them that resulted in publications in competitive conferences and journals. Gözde led most of the efforts for our work that appeared in SDM 2021. Jasmin was only a second-year undergraduate, when joined our group for a short period; however, she made considerable contributions to the work presented in Chapter 5. Milad developed most of the theoretical material presented in Chapter 4. I also thank the faculty members, Prof. Edmund Yeh and Prof. Jose Bento for contributing to the results presented in chapters 4 and 5, respectively.

I also would like to thank dear friends who made my over 5 year journey as a Ph.D. student enjoyable. I thank Kimia Shayestehfard and Khashayar Kamran for being great friends and I will definitely miss our amazing tea breaks. I also thank my friends outside the academic circle. I am grateful to Andrea Babbs, Misha D'Andre, Galen Hench, and Jennifer Dyment for welcoming me into their circle and amazing game nights. I thank Siavash Kayal for being a great friend and my roommate for over 4 years.

Abstract of the Dissertation

Leveraging Structural Properties for Large-Scale Optimization

by

Armin Moharrer

Doctor of Philosophy in Electrical and Computer Engineering

Northeastern University, April 2021

Dr. Stratis Ioannidis, Advisor

Large scale optimization problems abound in data mining, machine learning, and system design. We address the challenges posed by such large scale optimization problems by providing efficient optimization algorithms. The scope of studied problems is quite broad; it includes applications such as experimental design, computing graph distances (dissimilarity scores), training auto-encoders, multi-target regression, and the design of cache networks. We leverage the structural properties present in these problems, e.g., sparsity or separability. In particular, we introduce some structural properties under which the Frank-Wolfe algorithm (FW) can be distributed over a cluster of computers. We show that the distributed FW running over 350 workers (CPUs) solves an instance of experimental design problem with 20M variables in 79 minutes, while the serial implementation takes 48 hours. Furthermore, we study a variant of FW for the design of cache networks. The problem is NP-hard, but we achieve a $1 - 1/e$ approximation ratio, by optimizing a non-convex relaxation via FW. We also propose a distributed Alternating Direction Method of Multipliers (ADMM) algorithm for computing graph distances. We observe speedups of 153 times when running over a cluster with 448 CPUs, in comparison with running over 1 CPU, for graphs with 2.4K nodes. Finally, we study applications of ADMM in solving robust variants of risk minimization problems; in these variants we replace the typically chosen mean squared error loss with a general ℓ_p norm. We combine model based optimization with ADMM to minimize the resulting non-smooth and non-convex objectives. We show that a stochastic variant of ADMM converges with the rate $O(\log T/T)$ and is highly efficient for optimizing the corresponding model functions.

Chapter 1

Introduction

1.1 Motivation

Many interesting problems in machine learning, data mining, and system design can be formulated as (convex or non-convex) optimization problems. The goal in these problems is to minimize/maximize a design objective, such as the total error over a training dataset in regression and classification problems [2, 3], informativeness in experimental design [4, 5], or the total expected delay in the design of cache networks [6, 7].

As a result of drastic growth in the size of data, these optimization problems in practice are *large-scale*. For example, the input data used for these problems can be in the order of terabytes, resulting in problems with millions of terms in their objectives or millions of parameters, e.g., weights in neural networks or decision variables in design problems. Moreover, regardless of the input data, in some cases the underlying models are highly complex and have a large number of parameters themselves; quintessential examples are deep learning architectures, such as AlexNet [8] and ResNet [9] with about 61M and 23M parameters, respectively. Classic optimization algorithms, such as gradient descent, the Alternating Direction Method of Multipliers (ADMM), or the Frank-Wolfe algorithm, are inefficient and extremely slow for solving such large-scale problems.

The existence of these large-scale optimization problems have motivated the study of parallel or distributed optimization algorithms. They are implemented in distributed frameworks that leverage the massively parallel computational power of computer clusters. For example, the input data is distributed over a cluster of workers, e.g., CPUs or GPUs, where workers run local algorithms on smaller subsets of input data in parallel and communicate sporadically to synchronize or exchange information. Some promising works have been done in this direction, such as distributed gradient

methods [10, 11, 12], or the distributed alternating direction method of multipliers [13, 14, 15], just to name a few. In fact, in this thesis we also present two such distributed algorithms and show their scalability in Chapters 3 and 5. However, it is not clear how to parallelize classic optimization algorithms for general problems. Moreover, though parallel algorithms offer *computational* advantages through parallelism they also incur *communication* costs. Therefore, even in cases that parallel or distributed algorithms exist, one needs to assess their communication costs carefully for problems at hand, as the performance of these parallel algorithms highly depends on communication patterns implied by input data or problems. For instance, in order to achieve speedups via parallelism, these methods often require that the input data to be large only along one dimension but of moderate size along the other dimensions, as is the case for our distributed algorithms in Sec. 3. As a result, parallel or distributed algorithms are only interesting if the computational advantages overcome communications overheads.

Fortunately, many interesting applications have structural properties that give rise to efficient algorithms. One such property is *sparsity*. There are different ways that sparsity can arise: for example, the quantities, e.g., vectors or matrices, that are used to describe the problems or are computed at intermediate steps have only a few non-zero elements [16, 17, 18]. One immediate advantage of sparse vectors is that they can be denoted by parsimonious representations [19], e.g., the indices of non-zero elements and the corresponding values [20]. In addition, they often offer further advantages, for instance, matrix computations [21], e.g., matrix inversion [22, 23, 24] or matrix-vector multiplication [25, 26], that involve sparse vectors can be done efficiently. In addition, sparsity can arise in the support of functions [13, 20], too; often problems have objectives that are a summation over a number of terms, where each term is a function of a small subset of the input data or parameters. As a result, these functions with sparse supports can be computed while having access to only small subsets of data. Besides computational advantages, these sparsity patterns offer advantages in terms of communication costs in the case of distributed algorithms. For example, only a small portion of data needs to be communicated across workers, so that they can evaluate a function or update their data, or each worker only needs to communicate with a small subset of workers [17, 20, 27, 28].

Motivated by the potential advantages that sparsity patterns offer for dealing with large-scale problems, in this thesis we explore sparse structures and their advantages across different problems in machine learning, data mining, and systems design. In some cases, we propose distributed algorithms that run in parallel, where we determine computational and communication complexities in terms of the sparsity parameters. We summarize the the problems that we consider,

the role of sparsity in deriving efficient algorithms, and whether the algorithms are parallel in Table 1.1. We next briefly describe these problems and explain how sparsity patterns (mentioned in Table 1.1) arise.

1.2 Contributions

1.2.1 Distributing Frank-Wolfe via Map-Reduce

The Frank-Wolfe [29] algorithm (FW) is a convex optimization method. It solves constrained optimization problems, where the constraint is a compact convex set. It is an iterative algorithm, where at each iteration it minimizes a linear objective subject to the constraint set. For this reason, FW is known to be highly efficient [30], as the problems with linear objectives can often be solved efficiently and in some cases the solutions are sparse. One example is the simplex, i.e., the set $\mathcal{D}_0 \equiv \left\{\theta \in \mathbb{R}^n_+ : \sum_{i=1}^n \theta_i = 1\right\}$. In fact, as we explain in Chapter 3 the solutions that FW generates in each iteration for the simplex are highly sparse, i.e., they have only one non-zero element. This enables us to develop distributed and highly scalable algorithms. Here to give the reader intuitions on how sparsity is leveraged, we briefly describe the experimental design problem, as an application that can be solved in parallel via our distributed algorithm. In experimental design, a learner wishes to regress a model from an input data comprising feature vectors $x_i \in \mathbb{R}^d$ and labels $y_i \in \mathbb{R}$ for n data points ($i = 1, \ldots, n$). The learner has only access to the feature vectors and needs to decide which labels y_i to collect. One formulation of this problem is the D-optimal design problem [31, 32], which minimizes the negative log entropy of a linear regression model under Gaussian. Formally, D-optimal design amounts to the following optimization problem: noise:

$$\min_{\theta \in D_0} \operatorname{logdet} \left(\sum_{i=1}^{N} \theta_i x_i x_i^\top \right)^{-1}, \tag{1.1}$$

where θ_i denotes the portion of experiments for the feature x_i. Solving (1.1) via gradient methods, e.g., FW, requires computing the inverse of the matrix $\sum_{i=1}^N \theta_i x_i x_i^\top$, which is computationally expensive, in general. However, considering the sparsity of the solutions generated via FW for the simplex, the matrix in an iteration is a rank-one update of its value in the previous iteration. We thusly leverage the Sherman-Morrison formula [22] and we can compute the inverse matrix efficiently through matrix-vector products.

In Chapter 3 we describe how more problems can be solved efficiently via FW, and how manipulating the sparsity of FW solutions we can develop distributed algorithms. In particular, we

make the following contributions

- We identify two sparsity-induced properties of the objective under which FW can be parallelized through map-reduce operations.
- We show that several important optimization problems, including experimental design, Adaboost, and projection to a convex hull satisfy the aforementioned properties.
- We implement our distributed FW algorithm on Spark [33], an engine for large-scale distributed data processing. Our implementation is generic: a developer using our code needs to only implement a few problem-specific computational primitives; our code handles execution over a cluster.
- We extensively evaluate our Spark implementation over large synthetic and real-life datasets, illustrating the speedup and scalability properties of our algorithm. For example, using 350 compute cores, we can solve problems of 10 million variables in 44 minutes, an operation that would take 133 hours when executed serially.
- We introduce two stochastic variants of distributed FW, in which we only compute a subsample of the elements of the gradient. We implement these algorithms on Spark and compare their performance with distributed FW.

On a high level, sparsity here arises as FW solutions for the subproblems with linear objectives subject to the simplex have only one non-zero element. This allows us to have highly salable distributed implementations. Moreover, as we show with experimental studies in Chapter 3, leveraging these sparsity patterns leads to significant speedups, even for serial implementations.

1.2.2 Design of Kelly Cache Networks via Submodular Maximization

Submodular maximization is a class of combinatorial optimization problems that are known to be NP-hard in general [34]. Intuitively, submodularity captures the notion of *diminishing return*, i.e., for a set function, the marginal gain due to adding one element to the set diminishes as more elements added to the set. Many applications in machine learning, data mining, viral marketing, and system design, that can be cast as submodular maximization problems; some examples are facility location [35], influence maximization [36], and data summarization [37]. Due to the hardness of submodular maximization problems, there has been an active area of research on the development of approximate algorithms for these problems [38, 39]. In particular, a variant of FW,

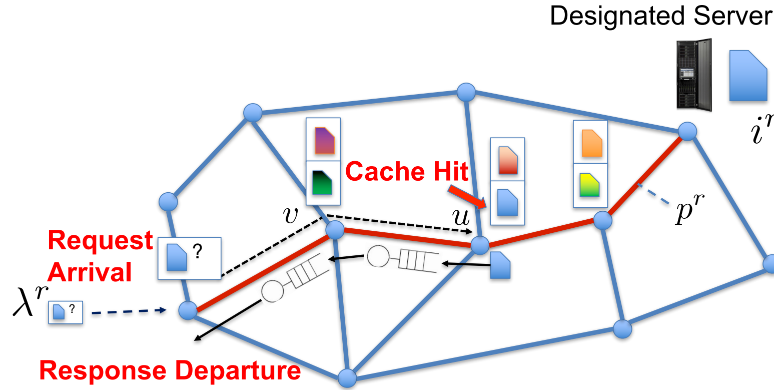


Figure 1.1: Illustrations of a cache network. The graph represents a network of entities (nodes) capable of storing data, performing computations, and making decisions. Requests r for an item i^r arrive on the nodes of the network with arrival rates λ^r. The requests are routed over the predetermined path p^r towards the designated server; however, they may terminate upon reaching a node in the path p^r that stores the requested item. At that point, responses are generated that carry the item, they traverse the path in reverse order and exit the network. As a result, by caching items earlier in the predetermined paths we can decrease the total traffic over the network. For the design of such cache networks, we need to find optimal items to store at each cache. Therefore, the problem has $|V| \times |\mathcal{R}|$ optimization variables in general, where V and $\mathcal{R}$ are the sets of nodes and requests, respectively. Nonetheless, sparsity naturally arises in this problem; for example, note that each request r corresponds to a single item i^r and travels over a path (e.g., denoted by red line) that includes only a subset of nodes and edges in the graph. Therefore, the considered cost functions corresponding to each request only depends on the optimization variables corresponding to the item i^r and the nodes along the path. By leveraging the sparsity in the support of this cost functions we propose an efficient variant of the Frank-Wolfe algorithm in Chapter 4.

i.e., the *continuous-greedy* algorithm [40] attains a $1 - 1/e$ approximation ratio to the optimal for submodular maximization problems over matroid constraints. In Chapter 4, we focus on a system design application, i.e., cache networks optimization. We show that it can be cast as a submodular maximization problem. We show that the continuous greedy algorithm similar to FW attains efficient implementations due to the structure of these problems. Before explaining further, let us briefly introduce *Cache networks*.

Cache networks [41, 42, 7] represent a network of entities capable of storing data, performing computations, and making decisions. The entities are represented as nodes of an undirected graph. For example consider the cache network illustrated in Fig. 1.1, where requests r for an item i^r arrive

on the nodes of the network with arrival rates λ^r. The requests are routed over the predetermined path p^r towards the designated server; however, they may terminate upon reaching a node in the path p^r that stores the requested item. At that point, responses are generated that carry the item, they traverse the path in reverse order and exit the network. As a result, by caching items earlier in the predetermined paths we can decrease the total traffic over the network. Moreover, we assume that the nodes have a certain capacity on the number of items that they can store. In Chapter 4, we propose a model for cache networks, in which the goal is to minimize a class of design objectives (e.g., the overall expected delay), by determining which items to store at each node, while satisfying the constraints on node capacities. Formally, the constraints can be written as

$$\sum_i x_{vi} \leq c_v, \quad \text{for all } v \in V, \tag{1.2}$$

where the set V represent the graph nodes, $c_v \in \mathbb{N}$ are the node capacities, and $x_{vi} \in [0,1]$ are decision variables determining the probability of caching the item i on node $v \in V$. Note that the constraints (1.2) are similar to the simplex, with only difference that the 1 on the right hand-side of the inequality is now replaced with c_v. The so-called continuous greedy algorithm, which is similar to FW, optimizes linear objectives subject to (1.2); this can again be done efficiently. Moreover, the objective in cache networks problems is usually the total expected delay, which is a summation over some terms, where each is a function of the total load on an edge in the network, i.e., the overall rates for requests passing through that edge. As a result, terms have sparse supports, as the load on each edge only depends on a subset of decision variables x_{vi}. Leveraging the sparsity offered by the continuous greedy algorithm and the sparsity in the support of the functions in the objective we obtain efficient algorithms for this problem (see Sec 4.2.2). As an additional means of improving efficiency, we also develop a method for reducing sampling in the algorithm.

In summary, we make the following contributions in Chapter 4

- We study the problem of optimizing the placement of objects in caches in Kelly cache networks of M/M/1 queues, with the objective of minimizing a cost function of the system state. We show that, for a broad class of cost functions, including packet delay, system size, and server occupancy rate, *this optimization amounts to a submodular maximization problem with matroid constraints*. This result applies to general Kelly networks with fixed service rates; in particular, it holds for FIFO, LIFO, and processor sharing disciplines at each queue.

- We leverage this connection to submodular optimization to study approximation algorithms for the resulting (NP-hard) problems. It is known that the classic greedy algorithm [38] produces a

solution with a 0.5 approximation ratio, while the so-called continuous-greedy algorithm [40] yields a ratio of $1 - 1/e \approx 0.63$. We prove that *the 0.5-approximation ratio of the greedy algorithm is tight*,

- The so-called continuous greedy algorithm [40] attains a $1 - 1/e$ approximation for this NP-hard problem. However, it does so by computing an expectation over a random variable with exponential support via randomized sampling. The number of samples required to attain the $1 - 1/e$ approximation guarantee can be prohibitively large in realistic settings. Our second contribution is to show that, for Kelly networks of M/M/1 queues, *this randomization can be entirely avoided*: a closed-form solution can be computed using the Taylor expansion of our problem's objective. In particular, we leverage the *sparsity* of the support of the terms in the design objective along with their product-form structure to efficiently compute the Taylor expansion of the objective. To the best of our knowledge, we are the first to identify a submodular maximization problem that exhibits this structure, and to exploit it to eschew sampling.

- Finally, we extend our results to *networks of M/M/k and symmetric M/D/1 queues*, and prove a negative result: submodularity does *not* arise in networks of M/M/1/k queues. We extensively evaluate our proposed algorithms over several synthetic and real-life topologies.

To sum up, in the cache networks problems that we consider sparsity patterns arise in the support of the functions, which correspond to total traffic over edges. Moreover, the decision variables are also sparse, as the result of sparsity in graphs that represent the networks and the fact that the predetermined paths (e.g., the shortest path between two nodes) for requests only include few nodes in the graphs. The sparsity patterns along with the product form of the design objectives allow us to compute the Taylor approximations of the objective efficiently. Therefore, we obtain an efficient algorithm with the $1 - 1/e$ approximation guarantee for the NP-hrad cache design problem.

1.2.3 Massively Distributed Graph Distances

Graph distance (or similarity) scores are used in several graph mining tasks, including anomaly detection, nearest neighbor and similarity search, pattern recognition, transfer learning, and clustering. For example, consider two graphs G_A and G_B in Fig. 1.2, in the graph distances problem we are interested in computing a value, which shows structural dissimilarities between the two graphs. In fact, the graphs in Fig. 1.2 are isomorphic, i.e., there is a one-to-one correspondence between the

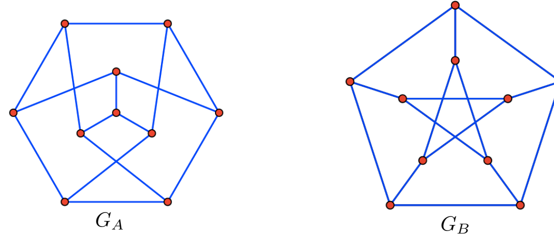


Figure 1.2: An example of two isomorphic graphs. Two graphs are isomorphic, if there is a one-to-one correspondence between the nodes in G_A and G_B, s.t., two nodes are adjacent in G_A if and only if their corresponding nodes in G_B are adjacent. Note that the distance between two isomorphic graphs is zero. In particular, for two isomorphic graphs there is a permutation matrix $P \in \{P \in \{0,1\}^{n\times n} | P\mathbf{1} = \mathbf{1}, P^\top \mathbf{1} = \mathbf{1}\}$, s.t., $BP^\top = AP$, where $A, B \in \mathbb{R}^{n\times n}$ are the adjacency matrices for the n-node graphs G_A and G_B. We consider a formulation that involves minimizing $\|AP - PB\|$, over the set of doubly stochastic matrices $\{P \in [0,1]^{n\times n} | P\mathbf{1} = \mathbf{1}, P^\top \mathbf{1} = \mathbf{1}\}$. Sparsity naturally arises here, as graphs usually have only few edges, in practice. Moreover, as explained in Chapter 5, we introduce sparsity on the support of the matrix P. As a result, each element in the matrix $AP - PB$ depends only on a small subset of the coordinates of the variable P. We leverage the sparsity to design an efficient distributed optimization algorithm.

nodes in G_A and G_B, s.t., two nodes are adjacent in G_A if and only if their corresponding nodes in G_B are adjacent. Therefore, the distance between G_A and G_B is zero, as there is no structural differences between them. Unfortunately, determining whether such one-to-one correspondence exists between nodes of two given graphs is a combinatorial problem and computationally hard. In this thesis, we focus on a convex relaxation that was proposed by Bento and Ioannidis [1]. In a nutshell, they propose to solve the following problem

$$\min_{P \in \mathbb{R}^{n\times n}_{\geq 0} : P\mathbf{1}=\mathbf{1}, P^\top \mathbf{1} = \mathbf{1}} \|AP - PB\|_p, \tag{1.3}$$

where A, B are the adjacency matrices of G_A, G_B, respectively, n is the number of nodes in the graph, $\|\cdot\|_p$ is an element-wise p-norm, and P is a matrix that captures the one-to-one correspondence. For instance, if the (i, j)-th element of P is 1, then the node i in G_A corresponds to the node j in G_B. The goal of (1.3) is to find a one-to-one correspondence between nodes of the graphs, s.t., the edge discrepancy between them is minimized. Consider the case, where $p = 1$, then we see that the objective is a summation over n^2 terms, i.e., the absolute value of the elements of $AP - PB$. Moreover, under the assumption that the graphs G_A, G_B are sparse, i.e., each node is only connected to few other nodes, the (i, j)-th element of the matrix $AP - PB$ only depends on few elements of P. As we explain in Chapter 5, we can further introduce sparsity patterns on the support of P. In Chapter 5 we propose a distributed ADMM-based algorithm for the graph distances problem (1.3),

where the sparsity is key in the development; it is also essential for the performance of the distributed algorithm.

In particular, for the graph distances problem we make the following contributions in Chapter 5.

- We propose an ADMM-based distributed algorithm for solving (1.3) *for all* $p \geq 1$. Our solution for the case $p > 1$ uses a nested-ADMM (Alg. 8 and 9) in combination with a novel distributed bisection algorithm (Alg. 10) as building blocks. The bisection algorithm computes the proximal operator of a p-norm via repeated map-reduce operations, and is therefore of interest in its own right.
- We implement our algorithm in OpenMP [43] and Spark [33]. Our publicly available implementation scales to hundreds of CPUs. Over a 448 CPU cluster, we attain speedups of as much as 154×, in comparison to using a single CPU.

In summary, for the problem of graph distances in (1.3), sparsity arises in graphs G_A, G_B, and the support of the matrix P. This enables us to propose distributed algorithms. In particular, in Chapter 5, we explicitly bound the total of number of exchanged messages for our proposed algorithm in terms of these these sparsity parameters.

1.2.4 Robust Regression via Model-Based Optimization

Mean Squared Error (MSE) loss problems are ubiquitous in machine learning and data mining. Such problems have the following form:

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \|F(\theta; \mathbf{x}_i)\|_2^2 + g(\theta), \tag{1.4}$$

where function $F : \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}^N$ captures the contribution of a sample $\mathbf{x}_i \in \mathbb{R}^m, i = 1, \ldots, n$, to the objective under the parameter $\theta \in \mathbb{R}^d$ and $g : \mathbb{R}^d \to \mathbb{R}$ is a regularizer. Example applications include training auto-encoders [44, 45], matrix factorization [46], and multi-target regression [47]..

For instance, in training auto-encoders [48, 44], we are given n data points $\mathbf{x}_i \in \mathbb{R}^m$, $i \in [n]$. Auto-encoders embed these datapoints in a $m'-$dimensional space, $m' \ll m$, as follows. The mapping to $\mathbb{R}^{m'}$ is done by a possibly non-linear function (e.g., a neural network) with $d_{\texttt{enc}}$ parameters $F_{\texttt{enc}} : \mathbb{R}^{d_{\texttt{enc}}} \times \mathbb{R}^m \to \mathbb{R}^{m'}$, called the *encoder*. An inverse mapping, the *decoder* $F_{\texttt{dec}} : \mathbb{R}^{d_{\texttt{dec}}} \times \mathbb{R}^{m'} \to \mathbb{R}^m$ with $d_{\texttt{dec}}$ parameters re-constructs the original points given latent embeddings.

Both the encoder and the decoder are trained jointly over a dataset $\{\mathbf{x}_i\}_{i=1}^n$ by minimizing the reconstruction error; cast in our robust setting, this amounts to minimizing (1.4) with

$$F(\theta;\mathbf{x}_i) = \mathbf{x}_i - F_{\texttt{dec}}\left(\theta_{\texttt{dec}}; F_{\texttt{enc}}(\theta_{\texttt{enc}};\mathbf{x}_i)\right), \tag{1.5}$$

where $\theta = [\theta_{\texttt{dec}};\theta_{\texttt{enc}}] \in \mathbb{R}^{d_{\text{enc}}+d_{\text{dec}}}$ comprises the parameters of the encoder and the decoder.

The MSE loss in (1.4) is computationally convenient, as the resulting problem is smooth and can thus be optimized efficiently via gradient methods, such as stochastic gradient descent (SGD). However, it is well-known that the MSE loss is not *robust to outliers* [2, 49, 50, 51, 52], i.e., samples far from the dataset mean. Intuitively, when squaring the error, outliers tend to dominate the objective. To mitigate the effect of outliers, a classic approach is to introduce robustness by replacing the squared error with either the ℓ_2 norm [49, 50, 53, 44, 54, 45] or the ℓ_1 norm [55, 56, 52, 57, 58, 59, 60, 51]. This has been applied to several applications, including feature selection [50, 53], PCA [57, 58, 59, 49], K-means clustering [54], training autoencoders [44, 45], matrix factorization [55, 56, 52, 51], and regression [60]. Motivated by this approach, we study the following robust variant of Problem (1.4):

$$\min_{\theta} \frac{1}{n}\sum_{i=1}^{n} \|F(\theta;\mathbf{x}_i)\|_p + g(\theta), \tag{1.6}$$

where $\|\cdot\|_p$ denotes an ℓ_p norm ($p \geq 1$). We are particularly interested in cases where F is not affine and, in general, Problem (1.6) is non-convex. This includes, e.g., feature selection [50], matrix factorization [51, 55], auto-encoders [44], and deep multi-target regression [60, 47].

A significant challenge behind solving Prob. (1.6) is that its objective is not smooth, precisely because the ℓ_p norm is not differentiable at $\mathbf{0} \in \mathbb{R}^N$. For non-convex and non-smooth problems of the form (1.6), where the objective contains a composite function, *Model-Based Optimization* (MBO) methods [61, 62, 63, 64, 65, 66] come with good experimental performance as well as theoretical guarantees. In particular, these MBO methods define a convex (but non-smooth) approximation of the main objective, called the *model function*. They then iteratively optimize this model function plus a proximal quadratic term. Under certain conditions, MBO converges to a stationary point of the non-convex problem [61].

In Chapter 6, we use MBO to solve Problem (1.6) for arbitrary ℓ_p norms. In particular, each MBO iteration results in a convex optimization problem. We solve these sub-problems using a novel stochastic variant of the Online Alternating Direction Method (OADM) [67], which we call *Stochastic Alternating Direction Method* (SADM). Using SADM is appealing, as its resulting

steps have efficient gradient-free solutions; in particular, we exploit a bi-section method [68] (c.f. Sec. 5.3.2 in Chapter 5) for finding the proximal operator of ℓ_p norms. We provide theoretical guarantees for SADM. As an additional benefit, SADM comes with a stopping criterion, which is hard to obtain for gradient methods when the objective is non-smooth [69].

Overall, we make the following contributions in Chapter 6:

- We study a general outlier-robust optimization that replaces the MSE with ℓ_p norms. We show that such problems can be solved via Model-Based Optimization (MBO) methods.
- We propose SADM, i.e., a stochastic version of OADM, and show that under strong convexity of the regularizer g, it converges with a $O(\log T/T)$ rate when solving the sub-problems arising at each MBO iteration.
- We conduct extensive experiments on training auto-encoders and multi-target regression. We show (a) the higher robustness of ℓ_p norms in comparison with MSE and (b) the superior performance of MBO, against stochastic gradient methods, both in terms of minimizing the objective and performing down-stream classification tasks. In some cases, we see that the MBO variant using SADM obtains objectives that are 29.6$\times$ smaller than the ones achieved by the competitors.

In summary, we provide robust alternative formulations to MSE minimization in Chapter 6. Nonetheless, this results in non-smooth non-convex problems for which gradient methods are not efficient. However, we show that MBO methods jointly with a novel variant of ADMM provide efficient methods. The key in our approach is the efficient proximal operators for ℓ_p norms, which we introduce in Chapter 5.

1.3 Conclusion

We introduced a diverse set of applications that can be cast as optimization problems. We further briefly explained that these problems have structural properties that can be leveraged to develop efficient optimization methods. The remainder of this thesis is organized as follows: we introduce classic topics related to convex optimization in Chapter 2. We present our distributed Frank-Wolfe algorithm in Chapter 3. We show an application of cache networks that can be solved via a variant of the Frank-Wolfe algorithm in Chapter 4. We explain our results for computing graph distances via ADMM in Chapter 5. Finally, we present a general robust formulation for regression

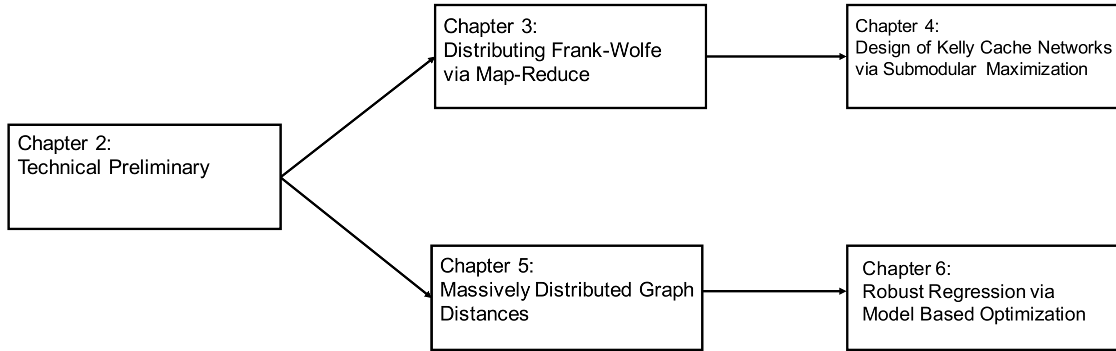


Figure 1.3: Chapter Dependencies.

Problem	Sparsity Pattern	Parallel	Chapter
Distributing Frank-Wolfe via Map-Reduce	Sparsity in intermediate solutions	YES	3
Design of Kelly Cache Networks via Submodular Maximization	Sparsity in the support of functions	NO	4
Massively Distributed Graph Distances	Sparsity in the support of functions + sparse vector representation	YES	5
Robust Regression via Model-Based Optimization	Separability of objective	NO	6

Table 1.1: Summary of problems and the role of sparsity in obtaining efficient algorithms.

problems along with algorithms to solve it in Chapter 6. For a summary of chapters, see Table 1.1 and Fig. 1.3.

Chapter 2

Technical Preliminary

The focus of this thesis is on developing efficient optimization algorithms for a variety of problems. To that end, we review basic definitions related to convex optimization in Sec. 2.1; we introduce the notion of convexity in sets and functions along with formal definition of convex optimization problems. We then review some well-known convex optimization algorithms that are key for methods we develop in this thesis, in Sec. 2.2. We then review a class of combinatorial optimization problems known as submodular maximization in Sec. 2.3. Finally, we briefly introduce distributed frameworks that can be used for implementing distributed algorithms, in Sec 2.4.

2.1 Optimization

The following topics are classic in the convex optimization literature, the following definitions and results are mostly adopted from Boyd and Vandenberg [31] and Bertsekas [70]; we refer interested readers to these sources for more information.

Convex set. A set $\mathcal{D} \in \mathbb{R}^N$ is called *convex* if the line segment defined by any two points in the set lies within the set. Formally, for every $\theta_1, \theta_2 \in \mathcal{D}$ and $\alpha \in [0, 1]$ we have:

$$(1 - \alpha)\theta_1 + \alpha\theta_2 \in \mathcal{D}.$$

Given m points $\theta_1, \dots, \theta_m \in \mathbb{R}^N$, a point $\theta \in \mathbb{R}^N$ is called a *convex combination* of $\theta_1, \dots, \theta_m$ if there exist $\alpha_1, \dots \alpha_m \in \mathbb{R}_{\geq 0}$, s.t., $\sum_{i=1}^m \alpha_i = 1$ and

$$\theta = \sum_{i=1}^m \alpha_i \theta_i.$$

The set of all convex combinations of a set $\mathcal{D}$ is called the *convex hull* of $\mathcal{D}$, and is denoted by $\textbf{conv}(\mathcal{D})$. It is straightforward to show that the convex hull of every set is a convex set. Moreover, the convex hull of a set is also the smallest convex set that contains it.

Convex function. A function $F : \mathbb{R}^N \to \mathbb{R}$ is *convex* if (a) its domain $\textbf{dom}F$ is a convex set and (b) for all $\theta_1, \theta_2 \in \textbf{dom}F$ and $\alpha \in [0, 1]$ it holds that

$$F((1-\alpha)\theta_1 + \alpha\theta_2) \leq (1-\alpha)F(\theta_1) + \alpha F(\theta_2). \tag{2.1}$$

Eq. (2.1) is known as Jensen's inequality [71]. In other words, the line segment between the points $(\theta_1, F(\theta_1)), (\theta_2, F(\theta_2))$ lies above the graph of the function F. A function F is called strongly convex if the above inequality holds with strong inequality for $\alpha \neq 0, 1$.

A convex function is continuous but not necessarily differentiable. When F is differentiable, convexity is equivalent to the following first-order condition [72]:

$$F(\theta_2) \geq F(\theta_1) + \nabla F(\theta_1)^\top(\theta_2 - \theta_1) \;\; \forall \theta_1, \theta_2 \in \textbf{dom}F. \tag{2.2}$$

Moreover, if F is differentiable and strongly convex there exists a constant $\beta \in \mathbb{R}_+$, s.t.,

$$F(\theta_2) \geq F(\theta_1) + \nabla F(\theta_1)^\top(\theta_2 - \theta_1) + \beta\|\theta_2 - \theta_1\|_2^2 \quad \forall \theta_1, \theta_2 \in \textbf{dom}F. \tag{2.3}$$

In other words, there is a quadratic lower-bound for F. In this case, we call F *β-strongly convex.*

Lipschitz continuity. For a subset $\mathcal{S} \subset \mathbb{R}^N$ a function F is *Lipschitz continuous* on $\mathcal{S}$ w.r.t. the Euclidean norm $\|\cdot\|$ if there exists a constant L such that

$$\|F(\theta_1) - F(\theta_2)\| \leq L\|\theta_1 - \theta_2\|, \;\; \forall \theta_1, \theta_2 \in \mathcal{S}. \tag{2.4}$$

Lipschitz continuity measures smoothness of the function F.

Curvature. A concept closely related to Lipschitz continuity is the *curvature*, which is a measure of nonlinearity of the convex function F over the convex domain $\mathcal{D}$ [30]. Formally, the *curvature C_f* of a convex and differentiable function F is defined as:

$$C_F \equiv \sup_{\theta_1, s \in \mathcal{D}, \alpha \in [0,1], \theta_2 = (1-\alpha)\theta_1 + \alpha s} 2/\alpha^2 \left(F(\theta_2) - F(\theta_1) - (\theta_2 - \theta_1)^\top \nabla F(\theta_1)\right). \tag{2.5}$$

From (2.2) we know that if F is convex and differentiable, then $F(\theta_2)$ lies above its first-order Taylor approximation $F(\theta_1) + (\theta_2 - \theta_1)^\top \nabla F(\theta_1)$. Hence, C_f measures the maximum deviation of F at θ_2 from its linearization at θ_1, scaled with the inverse of the step-size α. For example, if F is a linear function, its curvature C_F is zero.

The curvature of F is related to the Lipschitz continuity constant of the gradient of F: if ∇F is L-Lipschitz continuous on $\mathcal{D}$ then $C_F \leq (\text{diam}(\mathcal{D}))^2 L$, where $\text{diam}(\mathcal{D}) \equiv \sup_{\theta_1,\theta_2 \in \mathcal{D}} \|\theta_1 - \theta_2\|$ denotes the diameter of the set $\mathcal{D}$ [30].

Lower Semi-continuous functions. The function $F : \mathbb{R}^N \to \mathbb{R}$ is lower semi-continuous at a point θ_0 if, for every $\epsilon > 0$, there exists a neighborhood $\mathcal{U}_{\theta_0}$ around θ_0, s.t., for all $\theta \in \mathcal{U}_{\theta_0}$ the following holds: $F(\theta) \geq F(\theta_0) + \epsilon$. For more details on lower semi-continuity, refer to Chapter 7 in the book by Kurdila and Zabarankin [73].

Quasi-convex functions. A function $F : \mathbb{R}^N \to \mathbb{R}$ is *quasi-convex* if (a) its domain $\textbf{dom}F$ and (b) all its t-level subsets, i.e., $S_t = \{\theta \in \textbf{dom}F | F(\theta) \leq t\}$ are convex sets. There is a counterpart of Jensen's Inequality, i.e., (2.1), for quasi-convex functions; the following holds for all quasi-convex function and also implies quasi-convexity for all functions with convex domains:

$$F((1-\alpha)\theta_1 + \alpha\theta_2) \leq \max(F(\theta_1), F(\theta_2)) \;\; \forall \theta_1, \theta_2 \in \textbf{dom}F, \alpha \in [0,1]. \tag{2.6}$$

It is easy to see that all functions that satisfy (2.6) also meet (2.1), i.e., all convex functions are also quasi-convex. Moreover, differentiable quasi-convex functions also satisfy a property similar to (2.2). In particular, for a differentiable function $F : \mathbb{R}^N \to \mathbb{R}$ with a convex domain, the quasi-convexity is equivalent to satisfying the following

$$\nabla F(\theta_1)^\top (\theta_2 - \theta_1) \leq 0 \;\; \forall \theta_1, \theta_2 \in \textbf{dom}F, \text{ s.t., } F(\theta_2) < F(\theta_1). \tag{2.7}$$

Convex optimization. A general *convex optimization* problem has the form:

$$\min_{\theta \in \mathcal{D}} F(\theta), \tag{2.8}$$

where $F : \mathbb{R}^N \to \mathbb{R}$ is a convex function and $\mathcal{D}$ is a convex set. A property that makes convex optimization appealing is that every local minimum is a global minimum. Furthermore, for convex optimization the necessary and sufficient optimality condition is given by the following theorem.

Theorem 2.1.1 (Bertsekas, [70]). *For a convex and differentiable function F, the point θ^* is a minimum of F over a convex set $\mathcal{D}$ if and only if*

$$\nabla F(\theta^*)^\top (\theta - \theta^*) \geq 0, \;\; \forall \theta \in \mathcal{D}.$$

2.1.1 Constrained Optimization and Optimality Conditions

Consider a general optimization problem of the following form:

$$\text{Maximize} \quad F(\theta) \tag{2.9a}$$

$$\text{Subj. to} \quad g_j(\theta) \geq 0 \quad j = 1, \ldots, r \tag{2.9b}$$

$$h_i(\theta) = 0 \quad i = 1, \ldots, m, \tag{2.9c}$$

where $F : \mathbb{R}^n \to \mathbb{R}$, $g_j : \mathbb{R}^n \to \mathbb{R},\ j = 1, \ldots, r$, and $h_i : \mathbb{R}^n \to \mathbb{R},\ i = 1, \ldots, m$ are continuously differentiable functions. Here we provide a statement of the most common first-order necessary optimality condition known as *KKT* condition. First let us define a regular point:

Definition 2.1.1. Regular point*: If the gradient of equality constraints and active inequality constraints are linearly independent at θ^*, then θ^* is called a regular point.*

Now, let us formally define *Karush-Kuhn-Tucker* (KKT) points which we use extensively throughout this paper and in stating optimality conditions:

Definition 2.1.2. *A point $\theta^* \in \mathbb{R}^n$ is called a KKT point for Problem* (2.9) *if there exist* Lagrangian variables $\boldsymbol{\nu}^* \in \mathbb{R}^m$ *and* $\boldsymbol{\mu}^* \in \mathbb{R}^r$*, such that:*

$$\begin{aligned} &\nabla_x L(\theta^*, \boldsymbol{\mu}^*, \boldsymbol{\nu}^*) = 0 \\ &h_i(\theta^*) = 0 \quad \forall i \in \{1, \ldots, m\} \\ &g_j(\theta^*) \geq 0 \quad \forall j \in \{1, \ldots, r\} \\ &\mu_j^* \geq 0 \quad \forall j \in \{1, \ldots, r\} \\ &\mu_j^* g_j(\theta^*) = 0 \quad \forall j \in \{1, \ldots, r\}. \end{aligned}$$

where $L(\theta, \boldsymbol{\mu}, \boldsymbol{\nu}) \triangleq F(\theta) + \sum_i \nu_i h_i(\theta) + \sum_j \mu_j g_j(\theta)$ is called the Lagrangian function.

Proposition 2.1.1.1. *(First-order KKT Necessary Conditions) Let θ^* be a local minimum of Problem* (2.9)*, and assume θ^* is regular. Then θ^* is a KKT point.*

Using the second derivatives of the Lagrangian function, we can state the sufficient condition for optimality.

Proposition 2.1.1.2. *(Second-order Sufficiency Conditions) Assume F, g_j, $\forall j = 1, \ldots, r$, and h_i, $\forall i = 1, \ldots, m$ are twice continuously differentiable, and θ^* is a KKT point with corresponding Lagrange variables $\boldsymbol{\mu}^*$ and $\boldsymbol{\nu}^*$. In addition let*

$$\boldsymbol{V}^T \nabla_{xx}^2 L(\theta^*, \boldsymbol{\mu}^*, \boldsymbol{\nu}^*) \boldsymbol{V} < 0,$$

for all $\boldsymbol{V} \neq 0$ *such that*

$$\nabla h_i(\theta^*)^T \boldsymbol{V} = 0, \ \forall i = 1, \ldots, m,$$
$$\nabla g_j(\theta^*)^T \boldsymbol{V} = 0, \ \forall j \in A(\theta^*).$$

In addition assume we have strict complementary slackness, i.e.,

$$\mu_j > 0 \quad \forall j \in A(\theta^*)$$

where $A(\theta^*)$ *is set of active inequality constraints in* θ^**, i.e.,*

$$A(\theta^*) \triangleq \{j \mid g_j(\theta^*) = 0\}$$

Then θ^* *is a strict local maximum of Problem* (2.9).

For further information on other forms of necessary and sufficient conditions for optimality, refer to the book by Bertsekas [70].

2.2 Methods for Convex Optimization

In the previous section, we reviewed some basic definitions of convexity and introduced the class of convex optimization problems (2.8). The study of convex optimization algorithms for solving these problems has been an active area of research. Here we review some algorithms that we use in later chapters of this thesis. For a more detailed discussion and other convex optimization algorithms, such as the barrier method, the interior-point methods, or Newton's method we refer the readers to Boyd and Vandenberg [31] and Bertsekas [70].

2.2.1 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) [74] is a popular algorithm suited for unconstrained optimization problems of the form

$$\text{Minimize} \quad F(\theta) = \sum_{i=1}^{N} \frac{1}{N} F_i(\theta), \tag{2.11}$$

where $F_i(\theta) = L(x_i, \theta) + r(\theta)$. The functions $L : \mathbb{R}^m \times \mathbb{R}^d \to \mathbb{R}, r : \mathbb{R}^d \to \mathbb{R}$ are convex, differentiable and Lipschitz continuous, and each x_i is, e.g., a feature vector of the i-th datapoint in some dataset. Usually, L is a loss function (e.g., for logistic regression it would be logistic loss) and r is a regularization term, e.g., $r(\theta) = \lambda \|\theta\|_2^2$.

SGD solves (2.11) by iteratively performing a sampled form of gradient descent. In particular, the gradient is approximated via the contribution of only one randomly chosen term F_j to ∇F: formally, each iteration $k+1$ of the algorithm is:

$$\theta^{k+1} = \theta^k - \alpha \nabla F_j(\theta^k),$$

where $j \in \{1, \ldots, N\}$ is drawn uniformly at random (u.a.r.) and independently of previous iterations.

ParallelSGD, proposed by [75], parallelizes SGD in the following way: assume that there are P workers, e.g., processors or threads, available. Each worker p independently runs SGD for T iterations and returns a solution θ_p. The final solution returned by the algorithm is the average of the solutions found by the workers:

$$\hat{\theta} = 1/P \sum_{p=1}^{P} \theta_p. \tag{2.12}$$

The most direct implementation requires the storage of the whole dataset on each worker; however, as it is discussed in [75] this can be avoided in practice, as each worker only touches T datapoints, selected u.a.r.; these can be pre-sampled to avoid transferring all N datapoints to a worker.

Despite the simplicity of ParallelSGD, the following strong convergence bound holds (here $g(\theta) = \lambda \|\theta\|_2^2$):

$$\mathop{\mathbb{E}}_{\hat{\theta} \in \mathcal{Q}} [F(\hat{\theta})] - \min_{\theta} F(\theta) \leq \frac{8\alpha G^2}{\sqrt{P\lambda}} \sqrt{L_{\nabla F}} + \frac{8\alpha G^2 L_{\nabla F}}{P\lambda} + 2\alpha G^2,$$

where $\mathcal{Q}$ is the distribution of $\hat{\theta}$, under P workers and $T = \frac{\log P - (\log \alpha - \log \lambda)}{2\alpha\lambda}$ iterations, and $G, L_{\nabla F}$ are upper-bounds for the Lipschitz continuity constants of $L, \nabla F$, respectively.

This and different implementations of distributed SGD [11, 76, 77, 78] suffer from several drawbacks: first of all, the algorithm heavily depends on a separable objective function, while many problems of interest do not have this form (see, e.g., D-OPTIMALDESIGN and A-OPTIMALDESIGN in Section 3.3). Second, it requires the communication and per-worker storage of the whole vector θ, which can be large; this can be avoided when each function F_i depends on a few coordinates of θ [76, 11, 78], but not when the functions $F_1, \ldots, F_N$ have dense support. Moreover, SGD requires a differentiable objective, which further limits the problems that it can solve. For example, it cannot be applied to problems with the widely-used ℓ_1 regularization term.

2.2.2 Frank-Wolfe

Frank-Wolfe (FW) [29] has attracted interest recently due to its numerous computational advantages [79, 80, 81, 82, 18, 30]. It maintains feasibility throughout execution while being

projection-free, and minimizes a linear objective in each step; the latter yields sparse solutions for several interesting constraint sets, which often accelerates computation [18, 16, 30].

In particular, the Frank-Wolfe algorithm (FW) [29] solves the convex optimization problems of the form:

$$\begin{aligned} \text{Minimize} \quad & F(\theta) \\ \text{subj. to:} \quad & \theta \in \mathcal{D}, \end{aligned}$$

where $F : \mathbb{R}^N \to \mathbb{R}$ is a convex function and $\mathcal{D}$ is a convex compact subset of $\mathbb{R}^N$. The algorithm selects an initial feasible point $\theta^0 \in \mathcal{D}$ and proceeds as follows:

$$s^k = \arg\min_{s \in \mathcal{D}} \, s^\top \cdot \nabla F(\theta^k), \tag{2.13}$$

$$\theta^{k+1} = (1 - \gamma^k)\theta^k + \gamma^k s^k, \tag{2.14}$$

for $k \in \mathbb{N}$, where $\gamma^k \in [0, 1]$ is the step size. Basically, at each iteration it finds a feasible point $s^k \in \mathcal{D}$ that minimizes the first-order Taylor approximation of the function F around the current solution θ^k. This is an advantage of FW as it reduces the optimization of a general form convex function F to the optimization of a linear function, subject to the same constraint set. Then, it adapts the solution by finding a convex combination of the points $\theta^k, s^k \in \mathcal{D}$. As a result, it maintains the feasibility of the solution θ^k at all iterations $k \in \mathbb{N}$.

2.2.3 Alternating Directions Method of Multipliers

The Alternating Directions Method of Multipliers (ADMM) was proposed by Gabay and Mercier [83, 84] Glowinski and Marroco [85]. Interest in ADMM surged after the seminal work by Boyd et al. [13], which showed that it can be applied to a variety of problems and proposed distributed implementations.

ADMM solves problems of the form

$$\text{Minimize} \quad F(\theta_1) + g(\theta_2) \tag{2.15a}$$

$$\text{subj. to} \quad A\theta_1 + B\theta_2 = C, \tag{2.15b}$$

where the functions $F : \mathbb{R}^N \to \mathbb{R}, g : \mathbb{R}^m \to \mathbb{R}$ are convex, $A \in \mathbb{R}^{p \times N}, B \in \mathbb{R}^{p \times m}$, and $C \in \mathbb{R}^p$. ADMM forms the augmented Lagrangian:

$$L(\theta_1, \theta_2, y) = F(\theta_1) + g(\theta_2) + y^\top (A\theta_1 + B\theta_2 - C) + \rho/2\|A\theta_1 + B\theta_2 - C\|_2^2,$$

where $y \in \mathbb{R}^p$ is the dual variable associated with (2.15b). ADMM minimizes the augmented Lagrangian w.r.t. the primal variables θ_1, θ_2 alternatively, i.e., keeping one fixed and optimizing w.r.t. other one. Then, it updates the dual variable via gradient ascent. Formally, the algorithm proceeds as follows:

$$\begin{aligned}
\theta_1^{k+1} &:= \arg\min_{\theta_1} L(\theta_1, \theta_2^k, y^k) \\
\theta_2^{k+1} &:= \arg\min_{\theta_2} L(\theta_1^{k+1}, \theta_2, y^k) \qquad\qquad \text{for } k \in \mathbb{N} \\
y^{k+1} &:= y^k + \alpha(A\theta_1^{k+1} + B\theta_2^{k+1} - C).
\end{aligned}$$

Note that this is similar to Dual Ascent, but allows us to divide the primal variables to two sets and optimize them alternatively.

Stopping criteria. Primal and dual residual are typically used as a certificate of convergence in ADMM [13]; the former measures the feasibility of the current solution, while the latter is an indicator of the current optimality. Formally, at the k-th iteration of ADMM, the primal residual is the vector $\mathbf{r}^k \in \mathbb{R}^p$, defined as

$$\mathbf{r}^k = A\theta_1 + B\theta_2 - C. \tag{2.16}$$

The dual residual is the vector $\mathbf{s}^k \in \mathbb{R}^{N,}$ is defined as

$$\mathbf{s}^k = \rho A^\top B\left(\theta_2^k - \theta_2^{k-1}\right). \tag{2.17}$$

The iterations of ADMM terminate, once the primal and dual residuals are smaller than a certain ϵ.

Consensus ADMM can be used to parallelize optimization problems with a separable objective function, i.e., problems of the form:

$$\text{Minimize} \quad \sum_{i=1}^{N} F_i(\theta) + g(\theta).$$

Usually, each term $F_i : \mathbb{R}^d \to \mathbb{R}$ represents a loss function associated with the i-th datapoint, $g : \mathbb{R}^d \to \mathbb{R}$ is a regularization term, e.g., ℓ_1 penalty term, and $\theta \in \mathbb{R}^d$. This problem can be reformulated as:

$$\text{Minimize} \quad \sum_{i=1}^{N} F_i(\theta_i) + g(z) \tag{2.18a}$$

$$\text{subj. to} \quad \theta_i = z \;\; i = 1, \ldots, N, \tag{2.18b}$$

which is known as the *consensus* problem. The ADMM steps for (2.18) take the form:

$$\theta_i^{k+1} := \arg\min_{\theta_i} F_i(\theta_i) + (y_i^k)^\top(\theta_i - z^k) + \rho/2\|\theta_i - z^k\|_2^2 \tag{2.19}$$

$$z^{k+1} := \arg\min_{z} g(z) + \sum_{i=1}^{N}\left(-(y_i^k)^\top z + \rho/2\|\theta_i^{k+1} - z\|_2^2\right) \tag{2.20}$$

$$y_i^{k+1} := y_i^k + \rho(\theta_i^{k+1} - z^{k+1}). \tag{2.21}$$

Note that here the optimization w.r.t. each θ_i and adaptation of y_i can be done independently by a separate worker, i.e., a core or a thread. Also, the second step (2.20) can be done by a central unit in case d is small. This step can be further parallelized again when $F_1, \ldots, F_N$ depend on few variables [13].

ADMM offers several advantages. First, it can solve convex optimization problems with non-differentiable terms in the objective. In particular, it reduces the optimization of any convex function plus a ℓ_1 regularization term to the optimization of a quadratic term plus the ℓ_1 regularization term, which has a closed-form solution. Moreover, it can parallelize other generic optimization problems (see Sections 7 and 8 of [13]). However, it again assumes a separable objective function, which restricts the class of problems that it can solve. Finally, Consensus ADMM may not scale where the variable θ is high-dimensional and the functions $F_1, \ldots, F_N$ have dense support.

2.2.4 Distributed Stochastic Dual Coordinate Ascent

Distributed Stochastic Dual Coordinate Ascent (DSDCA) is another parallelizable optimization algorithm, which solves problems in their dual domain [86]. DSDCA solves problems of the form:

$$\text{Minimize} \quad \sum_{i=1}^{N} F_i(\theta^\top x_i) + g(\theta), \tag{2.22}$$

where functions $F_i : \mathbb{R} \to \mathbb{R}, i = 1, \ldots, N$, are convex and Lipschitz continuous, $g : \mathbb{R}^d \to \mathbb{R}$ is a β-strongly convex function, and $x_1, \ldots, x_N \in \mathbb{R}^d$ are feature vectors. In (2.22), similar to (2.11), each $F_i, i = 1, \ldots, N$, denotes a loss function and g represents a regularization term; here, each loss term F_i is explicitly a function of the linear term $\theta^\top x_i$.

Let us denote the convex conjugate of F_i and g by F_i^* and g^*, respectively. The dual problem of (2.22) is given by

$$\max_{\alpha} \sum_{i=1}^{N} -F_i^*(-\alpha_i) - g^*\left(\sum_{i=1}^{N} \alpha_i x_i\right).$$

In order to solve this problem, DSDCA at each iteration updates a randomly selected subset of the dual variables $\alpha_1, \ldots, \alpha_N$ via gradient ascent.

The separable form of the dual problem allows its solution to be parallelized: assume that the dataset $x_i, i = 1, \ldots, N$, is distributed between P workers. At iteration k of DSDCA each worker p iteratively updates m randomly chosen dual variables corresponding to the datapoints that it holds:

$$\alpha_i^{k+1} = \alpha_i^k + \Delta_i.$$

The step-size Δ_i is given by maximizing the following lower-bound of the dual function, which is a result of strong convexity of g (see (2.3)):

$$\Delta_i = \max_{\Delta\alpha} -F_i^*(-(\alpha_i^k + \Delta\alpha)) - \Delta\alpha x_i^\top \theta^k - \beta(\Delta\alpha)^2 \|x_i\|_2^2,$$

where $\theta^k = \nabla g^*(\sum_{i=1}^N \alpha_i^k x_i)$. A central unit computes θ^k by evaluating the gradient function $\nabla F^* : \mathbb{R}^d \to \mathbb{R}^d$.

Interestingly, it can be verified that for the optimal primal and dual solutions, i.e., θ^* and α^*, the following holds [86]:

$$\theta^* = \nabla g^*(\sum_{i=1}^N \alpha_i^* x_i).$$

Therefore, as α^k converges to α^*, the primal solution θ^k also converges to its optimal value θ^*.

The main advantage of DSDCA over SGD and ADMM is its convergence rate: studies have shown that coordinate-ascent in the dual domain outperforms SGD [87, 88, 89, 86]. Therefore, when the objective has the particular form (2.22), DSDCA is a more efficient alternative. However, similar to SGD and ADMM, its applicability is curbed because of the separable objective assumption. Furthermore, just like SGD and ADMM, if the vector θ is high-dimensional, DSDCA is both computation and communication intensive.

2.2.5 Bi-section Method

Bi-section method is a classic root-finding method. It is different from the optimization methods we have covered so far in this section, this is in the sense that it is not an optimization algorithm on its own; however, it is useful for solving some optimization problems. Some examples are finding the proximal operators (see Sec. 5.3.2 and [68]) or solving quasi-convex problems (see Sec. 4.2.5 in [31]), we review the latter below.

Algorithm 1 Bi-section method

1: Input: Points $x_1, x_2 \in \mathbb{R}$, s.t., $f(x_1)f(x_2) < 0$, accuracy $\epsilon < 1$
2: $x_l := x_1$, $x_u := x_2$
3: **for** $i = 1, \ldots, \lceil -\log \epsilon \rceil$ **do**
4: $\quad x_m := \frac{x_l + x_u}{2}$
5: $\quad$ **if** $f(x_l)f(x_m) < 0$ **then**
6: $\quad\quad x_u := x_m$
7: $\quad$ **else if** $f(x_m)f(x_u) < 0$ **then**
8: $\quad\quad x_l := x_m$
9: $\quad$ **else**
10: $\quad\quad$ **return** x_m
11: $\quad$ **end if**
12: **end for**
13: **return** x_m

More formally, the bi-section method finds the solution to equations of the form

$$f(x) = 0,$$

where $f : \mathbb{R} \to \mathbb{R}$ is a continuous function defined on an interval $[x_1, x_2]$, s.t., $f(x_1)f(x_2) < 0$. Based on the intermediate mean value theorem, there exists a root $x^* \in [x_1, x_2]$. The bi-section method proceeds in iterations, where at each iteration a search interval $[x_l, x_u]$ containing x^* is given; initially this interval is set as $x_l := x_1$ and $x_u := x_2$. Then at each iteration, it checks the function value for the point in the middle of the interval, i.e., $x_m = \frac{x_l + x_u}{2}$, and updates the search interval. In particular, if $f(x_1)f(x_m) < 0$, it sets $x_u := x_m$, and $x_l = x_m$, otherwise. Also, in the unlikely case that $f(x_m) = 0$, the bi-section method returns $x^* = x_m$. We summarize the steps of the bi-section method in Alg. 1. Note that in each iteration the interval is halved. Therefore, the bisection method finds a solution within ϵ from the root in $\lceil \log_2(\frac{x_u - x_l}{\epsilon}) \rceil$ rounds.

Bi-section Method for Quasi-convex Optimization. Here we explain an application of the bi-section method for quasi-convex optimization, i.e., the following class of problems:

$$\min_{\theta \in \mathcal{D}} F(\theta), \tag{2.23}$$

where F is a quasi-convex function (see the definition in Sec. 2.1) and $\mathcal{D}$ is a convex set. Quasi-convex optimization problems are non-convex, in general. However, a variation of the bi-section method can be used to find a local minimum.

The key in using the bi-section method is the following property of quasi-convex functions; there exists a class of convex functions $\phi_t : \mathbb{R}^N \to \mathbb{R}$, indexed by $t \in \mathbb{R}$, s.t., $F(\theta) \leq t$ implies that $\phi_t(\theta) \leq 0$, and vice-versa. In other words, the t-level subsets of a quasi-convex function F can be represented by the 0-level subsets of convex functions ϕ_t. More formally, for all $\theta \in \mathbb{R}^N, t \in \mathbb{R}$, $F(\theta) \leq t$, we have that $\phi_t(\theta) \leq 0$. Also, for all $\theta \in \mathbb{R}^N, t \in \mathbb{R}$ and $\phi_t(\theta) \leq 0$, it holds that $F(\theta) \leq t$. Note that one choice for ϕ_t is the characteristic function of the t-level subset of F, i.e., $\phi_t(\theta) = \chi_{S_t}(\theta)$, where $\chi_{S_t} : \mathbb{R}^N \to \{0, \infty\}$. However, the choice for the family of functions ϕ_t is not unique.

Let us denote the optimal solution for (2.23) by $F^* \triangleq \min_{\theta \in \mathcal{D}} F(\theta)$. Also, assume that we are given an interval $[F_l, F_u]$,, s.t., $F_l \leq F^* \leq F_u$. Then consider the following feasibility problem

$$\text{Minimize} \quad \text{Const.} \tag{2.24a}$$

$$\text{Subj. to:} \quad \phi_{F_m}(\theta) \leq 0 \tag{2.24b}$$

$$\theta \in \mathcal{D}, \tag{2.24c}$$

where $F_m = \frac{F_l + F_u}{2}$ and the objective is a constant. Note that (2.24) is a convex optimization problem and can be solved via convex optimization methods. If (2.24) has an optimal solution $\bar{\theta}$, it holds that $\phi_{F_m}(\bar{\theta}) \leq 0$; as a result, it also holds that $F^* \leq F(\bar{\theta}) \leq F_m$. Otherwise, if (2.24) does not have an optimal solution, it holds that $\phi_{F_m}(\theta) > 0, \ \forall \theta \in \mathcal{D}$; similar to the previous case, this implies that $F(\theta) > F_m, \forall \theta \in \mathcal{D}$ and subsequently $F_m < F^*$.

The bi-section provides a way to find an approximate solution, similar to Alg. 1; in particular, at each iteration, given a search interval $[F_l, F_u]$, it solves the feasibility problem (2.24) for the middle point F_m. If the feasibility problem has an optimal solution (Line 5), i.e., $F^* \leq F(\bar{\theta})$, the bi-section updates the search interval by setting the upper-bound (Line 6) $F_u = F(\bar{\theta})$. Otherwise, it updates the lower bound as (Line 8) $F_l = F(\bar{\theta})$. Note that, again, the bi-section method obtains a solution F_m that is within ϵ neighborhood of F^* in logarithmic number of rounds $\lceil \log_2(\frac{F_u - F_l}{\epsilon}) \rceil$.

2.3 Submodular Maximization

Here we review a class of combinatorial optimization problems, known as the submodular maximization. These problems are known to be NP-hard [90]; however, a variant of the Frank-Wolfe algorithm, i.e., the continuous greedy algorithm [40] provides a $1 - 1/e$ approximation guarantee. In this thesis, we use the continuous greedy algorithm in Chapter 4 for solving cache networks problems. We next define submodularity and related concepts, then we present the continuous greedy algorithm,

and show its connection to FW. We first review some basic definitions, see the paper by Krause and Golovin [91] for more details.

2.3.1 Set Functions and Submodularity.

Given a finite set $\mathcal{X}$, a set function $f : 2^{\mathcal{X}} \to \mathbb{R}$ is called *non-decreasing* if $f(S) \leq f(S')$ for all $S \subseteq S' \subseteq \mathcal{X}$, and *non-increasing* if $-f$ is non-decreasing. Function f is called *submodular* if it satisfies the following *diminishing returns* property: for all $S \subseteq S' \subseteq \mathcal{X}$, and all $x \in \mathcal{X}$,

$$f(S' \cup \{x\}) - f(S') \leq f(S \cup \{x\}) - f(S), \tag{2.25}$$

A function is called *supermodular* if $-f$ is submodular (or, equivalently, (2.25) holds with the inequality reversed).

Matroids. A matroid is a pair $\mathcal{M} = (V, \mathcal{I})$, where $V = \{1, \dots, N\} \triangleq [N]$ is the *ground set* of N elements and $\mathcal{I} \subseteq 2^V$ is the collection of *independent sets*, for which the followings hold.

1. For all $B \in \mathcal{I}$ and $A \subset B$ it holds that $A \in \mathcal{I}$.

2. For all $A, B \in \mathcal{I}$ and $|A| < |B|$, there exists $x \in B - A$, s.t., $A + \{x\} \in \mathcal{I}$.

Matroid Polytope. A polytope associated with the matroid $\mathcal{M}$ is the matroid polytope $P(\mathcal{M}) \subset \mathbb{R}_+^{|V|}$, defined as follows

$$P(\mathcal{M}) = \mathbf{convex}(\{\mathbf{1}_S : S \in \mathcal{I}\}), \tag{2.26}$$

where $\mathbf{1}_S \in \mathbb{R}^{|V|}$ is a vector with its i-th element is 1, if $i \in S$ and 0, otherwise and **convex** denotes the convex hull. The convex hull of a set is the smallest convex set that contains the set.

We introduce two examples of matroids

1. **Uniform Matroids.** A uniform matroid with k cardinality is defined as $\mathcal{I} = \{S \in 2^V ||S| \leq k\}$.

2. **Partition Matroids.** Let $\mathcal{B}_1, \dots, \mathcal{B}_m \subset V$ be disjoint sets, i.e., $\cap_{i=1,\dots,m} \mathcal{B}_j = \emptyset$ and $\cup_{1,\dots,m} \mathcal{B}_j = V$, then a partition matroid is the following $\mathcal{I} = \{S \subset 2^V | \forall j = 1, \dots, m, S \cap \mathcal{B}_j \leq k_j\}$, where $k_j \in \mathbb{N}, j \in [d] \triangleq \{1, \dots, d\}$ are given budgets.

2.3.2 Problem Statement

Consider the problem of maximizing a submodular function f subject to matroid constraints $\mathcal{M}$:

$$\max_{S\in\mathcal{I}} f(S). \tag{2.27}$$

The *greedy* [39] algorithm produces a solution within $\frac{1}{2}$-approximation from the optimal; however for cases where $\mathcal{I}$ is a uniform matroid, the approximation ratio is $1-1/e$. The *continuous greedy* algorithm [39] achieves a $1-1/e$ approximation ratio for (2.27) and for a general matroid $\mathcal{M}$. Note that this is a superior result in comparison with the greedy algorithm. In the following we review the continuous greedy algorithm, which maximizes the *multilinear relaxation* of a submodular function in the continuous domain.

Change of Variables. Note that every set $S \in 2^V$ can be represented by a binary N-dimensional vector $\mathbf{x}_S \in \{0,1\}^N$, where its i-th element is 1 if $i \in S$ and 0 otherwise.

2.3.3 Continuous Greedy Algorithm

The continuous greedy algorithm maximizes a relaxation of the set function f in the continuous domain; this relaxation is known as the *multilinear relaxation* [39], which we define here.

Multilinear Relaxation. The multilinear relaxation of a submodular function F is the function $G : [0,1]^N \to \mathbb{R}_+$, s.t.,

$$G(\mathbf{y}) = \mathbb{E}_{\mathbf{x}\sim\mathbf{y}}[f(S)] = \sum_{\mathbf{x}_S\in\{0,1\}^N} f(S) \prod_{i:x_i=1} y_i \prod_{i:x_i=0} (1-y_i). \tag{2.28}$$

Here, the expectation is taken over a distribution parametrized by the vector $\mathbf{y} = [y_i]_{i=1}^N$. More precisely, each element $i \in [N]$ is one with probability y_i, independently from other elements.

The continuous-greedy algorithm provides approximation guarantees for maximizing the multilinear relaxation of a submodular function subject to a *down-closed convex set* $\mathcal{D}$. A down-closed convex set is a convex set that (1) $\forall \mathbf{y} \in \mathcal{D}, \mathbf{y} \geq 0$ and (2) $\forall \mathbf{y}_2 \in \mathcal{D}$, $\mathbf{y}_1 \leq \mathbf{y}_2$ implies that $\mathbf{y}_1 \in \mathcal{D}$. To be more precise we focus on the following problem

$$\max_{\mathbf{y}\in\mathcal{D}} G(\mathbf{y}), \tag{2.29}$$

where $\mathcal{D} \triangleq P(\mathcal{M})$ is the matroid polytope, which is a down-closed convex set [39].

Algorithm 2 the Continuous Greedy algorithm

1: Input: $G : \mathcal{D} \rightarrow \mathbb{R}_+$, $0 < \gamma \leq 1$
2: $\mathbf{y}_0 \leftarrow 0, t \leftarrow 0, k \leftarrow 0$
3: **while** $t < 1$ **do**
4: $\quad \mathbf{m}_k \leftarrow \arg\max_{\mathbf{m}\in\mathcal{D}} \langle \mathbf{v}, \nabla G(\mathbf{y}_k) \rangle$
5: $\quad \gamma_k \leftarrow \min(\gamma, 1 - t)$
6: $\quad \mathbf{y}_{k+1} \leftarrow \mathbf{y}_k + \gamma_k \mathbf{m}_k$, $t \leftarrow t + \gamma_k$, $k \leftarrow k + 1$
7: **end while**
8: **return** $\mathbf{y}_k$

The continuous-greedy algorithm initially starts with a zero solution $y_0 = [0]$. Then it proceeds in iterations, where in the k-th iteration it finds a feasible point $\mathbf{m}_k \in P(\mathcal{M})$ which is a solution for the following problem

$$\max_{\mathbf{m}\in\mathcal{D}} \big\langle \mathbf{m}, \nabla G(\mathbf{y}_k) \big\rangle, \tag{2.30}$$

where the objective in (2.29) is replaced with a linear function. As we discuss later (2.30) does not need to be solved exactly [92]. After finding $\mathbf{m}_k$ the continuous greedy updates the current solution $\mathbf{y}$ as follows

$$\mathbf{y}_{k+1} = \mathbf{y}_k + \gamma_k \mathbf{m}_k, \tag{2.31}$$

where $\gamma_k \in [0, 1]$ is a step size. We summarize the continuous greedy algorithm in Alg. 2.

2.3.4 Continuous Greedy Algorithm and Frank-Wolfe

The continuous greedy algorithm is similar to the Frank-Wolfe algorithm, introduced in Sec. 2.2.2. In particular, by comparing (2.13) and (2.30) we see that they both optimize a linear objective, i.e., the first order Taylor expansion of the objective centered at the current solution subject to a convex set. The second step, which updates the current solution are slightly different between the two algorithms. In particular, compare (2.14) and (2.31); we see that the Frwnk-Wolfe algorithm updates the solutions as a convex combination of the current solution and the solution of the problem with linear objective, while the continuous greedy algorithm increments the current solution with a coefficient of the solution of the sub-problem (2.30). The reason for this is to leverage properties of the multilinear relaxation and provide a $1 - 1/e$ guarantee, for details on the proof refer to the seminal work by Calinescu et al. [39].

Table 2.1: Properties of DR-submodular functions

Properties	DR-submodular $f(.)$, $\forall \boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}$
0'th order	$f(\boldsymbol{x}) + f(\boldsymbol{y}) \geq f(\boldsymbol{x} \vee \boldsymbol{y}) + f(\boldsymbol{x} \wedge \boldsymbol{y})$, and $f(.)$ is coordinate-wise concave.
1'st order	Definition 2.3.1
2'nd order	$\frac{\partial^2 f(\boldsymbol{x})}{\partial x_i \partial x_j} \leq 0,\ \forall i, j \in [n]$

2.3.5 DR-submodular Functions

The notion of submodularity has been extended to the continuous domain [92], as *DR-submodular* functions. Bian et al. [92] define a DR-submodular function as follows:

Definition 2.3.1. *Suppose $\mathcal{X}$ is a subset of $\mathbb{R}^n$. A function $f : \mathcal{X} \to \mathbb{R}$ is DR-submodular if for all $\boldsymbol{a} \leq \boldsymbol{b} \in \mathcal{X}$, $i \in [n]$, and $k \in \mathbb{R}_+$, such that $(ke_i + \boldsymbol{a})$ and $(ke_i + \boldsymbol{b})$ are still in $\mathcal{X}$, the following inequality holds:*

$$f(ke_i + \boldsymbol{a}) - f(\boldsymbol{a}) \geq f(ke_i + \boldsymbol{b}) - f(\boldsymbol{b})$$

Intuitively, a DR-submodular function f is concave coordinate-wise along any non-negative or non-positive direction. DR-submodular functions arise in a variety of different settings (see Bian et al. [92]), and in some sense satisfy a weakened notion of concavity. They can also be defined in alternative ways that parallel the zero-th, first, and second order conditions for concavity (see [31]). For example, for $\mathcal{X} \subseteq \mathbb{R}^n$, a function $f : \mathcal{X} \to \mathbb{R}$ is DR-submodular *iff* for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}$,

$$f(\boldsymbol{x}) + f(\boldsymbol{y}) \geq f(\boldsymbol{x} \vee \boldsymbol{y}) + f(\boldsymbol{x} \wedge \boldsymbol{y}),$$

where $\vee$ and $\wedge$ are coordinate-wise maximum and minimum operations, respectively. A list of such conditions of DR-submodular functions is summarized in Table 2.1. Each one of these properties serves as a necessary and sufficient condition for a function to be DR-submodular [92]. The continuous greedy algorithm provides approximation guarantees for maximizing DR-submodular functions subject to down-closed convex sets, which are non-convex problems. This has been a recent topic of research [93, 94].

2.4 Parallel and Distributed Computing Frameworks

2.4.1 Map-Reduce Framework

Map-reduce [95, 96] is a distributed framework used to massively parallelize computationally intensive tasks. It enjoys wide deployment in commercial cloud services such as Amazon Web Services, Microsoft Azure, and Google Cloud, and is extensively used to parallelize a broad array of data-intensive algorithms [97, 98, 99, 100, 101]. Expressing algorithms in map-reduce also allows fast deployment at a massive scale: any algorithm expressed in map-reduce operations can be quickly implemented and distributed on a commercial cluster via existing programming frameworks [95, 96, 33].

Consider a data structure $D \in \mathcal{X}^N$ comprising N elements $d_i \in \mathcal{X}$, $i \in [N]$, for some domain $\mathcal{X}$. A `map` operation over D applies a function to every element of the data structure. That is, given $f : \mathcal{X} \to \mathcal{X}'$, the operation $D' = D.\texttt{map}(f)$ creates a data structure D' in which every element d_i, $i \in [N]$, is replaced with $f(d_i)$. A `reduce` operation performs an aggregation over the data structure, e.g., computing the sum of the data structure's elements. Formally, let $\oplus$ be a binary operator $\oplus : \mathcal{X} \times \mathcal{X} \to \mathcal{X}$ that is *commutative* and *associative*, i.e.,

$$x \oplus y = y \oplus x, \quad \text{and} \quad ((x \oplus y) \oplus z) = (x \oplus (y \oplus z)).$$

Then, $D.\texttt{reduce}(\oplus)$ iteratively applies the binary operator $\oplus$ on D, returning

$$\bigoplus_{i \in [N]} d_i = d_1 \oplus \ldots \oplus d_N.$$

Examples of commutative, associative operators $\oplus$ include addition (+), the $\min$ and $\max$ operators, binary AND, OR, and XOR, etc.

Both `map` and `reduce` operations are "embarrassingly parallel". Presuming that the data structure D is distributed over P processors, a `map` can be executed without any communication among processors, other than the one required to broadcast the code that executes f. Such broadcasts require only $\log P$ rounds and the transmission of $P - 1$ messages, when the P processors are connected in a hypercube network; the same is true for `reduce` operations [102]. There exist several computational frameworks, including Hadoop [96] and Spark [33], that readily implement and parallelize map-reduce operations. Hence, expressing an algorithm like FW in terms of `map` and `reduce` operations allows us to (a) parallelize the algorithm in a straightforward manner, and (b) leverage these existing frameworks to quickly implement and deploy FW at scale.

We opted to implement our distributed algorithm in the map-reduce framework. The first reason is that its implementations such as Hadoop [96] or Spark [33] are readily available on commercially used clusters. The second reason is that map-reduce implementations allow programming in high-level languages, e.g., Python, which are easy to program with. Code in these high-level languages can be written in a compact and concise fashion. In particular, we implement our distributed FW algorithm over Spark [33], which is an open-source implementation of map-reduce well-suited for parallelizing iterative machine learning algorithms: this is because Spark caches the results on RAM, so that they can be used in the next iteration.

2.4.2 Message Passing Interface (MPI)

Message Passing Interface (MPI) is a standard for message passing libraries aimed at running programs on HPC platforms, e.g., clusters of computers. The standard defines syntax and semantics of a set of library routines for different programming languages such as C, C++, or python. MPI is the dominant framework used in HPC applications [103]. MPI provides topology, communication, and synchronization between a set of processes: each process is usually mapped to a processor. MPI uses objects called *communicators* to determine which subset of processes may communicate with each other. MPI provides both *point-to-point* and *collective* communication routines.

Point-to-point routines are used for communication between two specific processes. For example, `MPI_Send` allows a specified process to send a message to another specified process. Correspondingly, `MPI_Recieve` lets a specified process to receive a message from another specified process. Collective routines involve communication between all processes in a communicator. Examples of collective routines are `MPI_Bcast`, `MPI_Scatter`, `MPI_Reduce`, or `MPI_Allreduce`. `MPI_Bcast` sends data from a specified process to all other processes. `MPI_Scatter` distributes data from a stipulated process between all of the processes. `MPI_Reduce`, similar to `reduce` in the map-reduce framework, aggregates over data held by different processes. `MPI_Allreduce` performs a reduction over the data and then broadcasts the result to every processor; it is a `MPI_Reduce` followed by `MPI_Bcast`.

Map-reduce has several advantages over MPI. First, map-reduce is readily available on commercial clusters, e.g., Amazon Web Services, Microsoft Azure, and Google Cloud. Second, programming in the map-reduce framework is much easier: implementation in MPI is done on lower-level languages, moreover, it requires explicit specifications of communication types as well as of the

processors that need to communicate. On the other hand, MPI allows for both point-to-point routines for communication between two specific processors and more sophisticated collective operations such as `MPI_Allreduce`.

PyTorch Distributed Backends. Recently there has been a prevalent surge of interest in PyTorch[1], which provides libraries for implementing, training, and evaluating deep learning models. Moreover, PyTorch, provides distributed computation via 3 backends, i.e., MPI, Gloo[2], and Nccl[3]. Gloo and NCCL provide communications similar to MPI, but they are both made compatible with PyTorch objects, NCCL works on GPUs.

2.4.3 Open Multi-Processing (OpenMP)

OpenMP is an API for shared-memory multiprocessing programming in C, C++, and Fortran. It offers several pros. It is efficient and provides high scalability comparable to MPI, when running on shared-memory platforms. Moreover, it is easier to work with in comparison to MPI, as the user does not need to deal with message passing. Moreover, original serial code generally do not need to be modified to run in parallel with OpenMP. However, it suffers from some drawbacks. First, the scalability is limited by the memory of the platform, while distributed frameworks, such as, MPI and Map-Reduce are more scalable. Also, it is hard to debug with regards to synchronization issues.

[1] https://pytorch.org
[2] https://github.com/facebookincubator/gloo
[3] https://github.com/nvidia/nccl

Chapter 3

Frank-Wolfe via Map-Reduce

Map-reduce [95, 96] is a distributed framework used to massively parallelize computationally intensive tasks. It enjoys wide deployment in commercial cloud services such as Amazon Web Services, Microsoft Azure, and Google Cloud, and is extensively used to parallelize a broad array of data-intensive algorithms [97, 98, 99, 100, 101]. Expressing algorithms in map-reduce also allows fast deployment at a massive scale: any algorithm expressed in map-reduce operations can be quickly implemented and distributed on a commercial cluster via existing programming frameworks [95, 96, 33].

In this chapter, we focus on solving, via map-reduce, optimization problems of the form:

$$\min_{\theta \in \mathcal{D}_0} F(\theta), \tag{3.1}$$

where $F : \mathbb{R}^N \to \mathbb{R}$ is a convex, differentiable function, and

$$\mathcal{D}_0 \equiv \left\{ \theta \in \mathbb{R}_+^N : \sum_{i=1}^N \theta_i = 1 \right\} \tag{3.2}$$

is the N-dimensional simplex. Several important problems, including experimental design, training SVMs, Adaboost, and projection to a convex hull indeed take this form [18, 16, 72]. We are particularly interested in cases where (a) $N \gg 1$, i.e., the problem is high-dimensional, and, (b) F *cannot* be written as the sum of differentiable convex functions. We note that this is precisely the regime in which (3.1) is hard to parallelize via, e.g., stochastic gradient descent.

It is well known that (3.1) admits an efficient implementation through the Frank-Wolfe (FW) algorithm, also known as the conditional gradient algorithm [29]. Frank-Wolfe (FW) [29] has attracted interest recently due to its numerous computational advantages [79, 80, 81, 82, 18, 30]. It maintains feasibility throughout execution while being projection-free, and minimizes a linear

objective in each step; the latter yields sparse solutions for several interesting constraint sets, which often accelerates computation [18, 16, 30]. Indeed, as we discuss in Sec. 3.1.4, FW assumes a very simple, elegant form under simplex constraints. Our main contribution is to identify and formalize a set of conditions under which solving Problem (3.1) through FW *admits a massively parallel implementation via map-reduce*.

Relevant to our work, Bellet et al. [16] propose a distributed version of FW for objectives of the form $F(\theta) = g(A\theta)$, for some $A \in \mathbb{R}^{d\times N}$, where $d \ll N$. Several examples fall in this class, including two we study here (convex approximation and Adaboost); intuitively, $A\theta$ serves as the common information in our framework (c.f. Sec. 3.2). The authors characterize the message and parallel complexity when A is partitioned across multiple processors under broadcast operations. Moreover, Tran et al in [104] elaborated on their algorithm, and proposed an asynchronous version of the distributed Frank-Wolfe algorithm in [16]. It is based on their *Stale Synchronous Parallel* (SSP) model [104]. They showed that the SSP based algorithm runs faster than the one based on a *Bulk Synchronous Parallel* (BSP) model, which is commonly used in distributed processing frameworks. We (a) consider a broader class of problems, that do not abide by the structure presumed by Bellet et al or Tran et al.. (e.g., the two experimental design problems presented in Sec. 3.3), and (b) establish properties under which FW can be explicitly parallelized through map-reduce rather than the message passing environment proposed by Bellet et al. This allows us to leverage commercial map-reduce frameworks to readily implement and deploy parallel FW on a cluster.

In particular, we make the following contributions:

- We identify two properties of the objective F under which FW can be parallelized through map-reduce operations.
- We show that several important optimization problems, including experimental design, Adaboost, and projection to a convex hull satisfy the aforementioned properties.
- We implement our distributed FW algorithm on Spark [33], an engine for large-scale distributed data processing. Our implementation is generic: a developer using our code needs to only implement a few problem-specific computational primitives; our code handles execution over a cluster.
- We extensively evaluate our Spark implementation over large synthetic and real-life datasets, illustrating the speedup and scalability properties of our algorithm. For example, using 350

compute cores, we can solve problems of 20 million variables in 79 minutes, an operation that would take 48 hours when executed serially.

- We introduce two stochastic variants of distributed FW, in which we only compute a subsample of the elements of the gradient. We implement these algorithms on Spark and compare their performance with distributed FW.

The remainder of this chapter is organized as follows. We introduce FW and its variants in Sec. 3.1. In Sec. 3.2, we state the properties under which FW admits a parallel implementation via map-reduce, and describe the resulting algorithm. Examples of problems that satisfy these properties are given in Sec. 3.3. We extend possible applications of our algorithm on constraint sets beyond the simplex in Sec. 3.4. Finally, in Sec. 3.5 and 3.6 we describe our implementation and the results of our experiments over a Spark cluster. Finally, we conclude in Sec. 3.7

3.1 Technical Background

3.1.1 Frank-Wolfe Algorithm

The FW algorithm [29], summarized in Alg. 3, solves problems of the form:

$$\text{Minimize} \quad F(\theta) \tag{3.3a}$$

$$\text{subj. to:} \quad \theta \in \mathcal{D}, \tag{3.3b}$$

where $F : \mathbb{R}^N \to \mathbb{R}$ is a convex function and $\mathcal{D}$ is a convex compact subset of $\mathbb{R}^N$. The algorithm selects an initial feasible point $\theta^0 \in \mathcal{D}$ and proceeds as follows:

$$s^k = \arg\min_{s \in \mathcal{D}} \; s^\top \cdot \nabla F(\theta^k), \tag{3.4a}$$

$$\theta^{k+1} = (1 - \gamma^k)\theta^k + \gamma^k s^k, \tag{3.4b}$$

for $k \in \mathbb{N}$, where $\gamma^k \in [0, 1]$ is the step size. At each iteration $k \in \mathbb{N}$, FW finds a feasible point s^k minimizing the inner product with the current gradient, and interpolates between this point and the present solution. Note that $\theta^{k+1} \in \mathcal{D}$, as a convex combination of $\theta^k, s^k \in \mathcal{D}$; therefore, the algorithm maintains feasibility throughout its execution. Steps (3.4a),(3.4b) are repeated until a convergence criterion is met; we describe how to set this criterion and the step size γ^k below.

Algorithm 3 FRANK-WOLFE

1: Pick $\theta^0 \in \mathcal{D}$
2: $k := 0$
3: **repeat**
4: $\quad s^k := \arg\min_{s\in\mathcal{D}} s^\top \cdot \nabla F(\theta^k)$
5: $\quad \text{gap} := (\theta^k - s^k)^\top \nabla F(\theta^k)$
6: $\quad \theta^{k+1} := (1-\gamma^k)\theta^k + \gamma^k s^k$
7: $\quad k := k+1$
8: **until** $\text{gap} < \epsilon$

Convergence criterion. Convergence is typically determined in terms of the *duality gap* [30]. The duality gap at feasible point $\theta^k \in \mathcal{D}$ in iteration $k \in \mathbb{N}$ is:

$$g(\theta^k) \equiv \max_{s\in\mathcal{D}} (\theta^k - s)^\top \nabla F(\theta^k) \overset{(3.4a)}{=} (\theta^k - s^k)^T \nabla F(\theta^k), \tag{3.5}$$

The convexity of F implies that $F(\theta^k) - F(\theta^*) \leq g(\theta^k)$ for any optimal solution $\theta^* \in \arg\min_{\theta\in\mathcal{D}} F(\theta)$ [30]. In other words, $g(\theta)$ is an upper bound on the objective value error at step k. The algorithm, therefore, terminates once the duality gap is smaller than some $\epsilon > 0$.

Step Size. The step size can be diminishing, e.g., $\gamma^k = \frac{2}{k+2}$, or set through *line minimization*, i.e.:

$$\gamma^k = \arg\min_{\gamma\in[0,1]} F\big((1-\gamma)\theta^k + \gamma s^k\big). \tag{3.6}$$

Convergence to an optimal solution is guaranteed in both cases for problems in which the objective has a bounded curvature [29, 30]. In this case, both of the above step sizes imply that the k-th iteration of the Frank-Wolfe algorithm satisfies $F(\theta^k) - F(\theta^*) \leq O(\frac{1}{k})$ [30]. For arbitrary convex objectives with unbounded curvature, FW still converges if the step size is set by the line minimization rule [70].

FW has several advantages. First, it reduces the optimization of a general convex function subject to a convex constraint set to the optimization of a linear objective function (see (3.4a)) subject to the convex constraint set. For several constraint sets of practical interest, e.g., linear constraint sets, these sub-problems are solved efficiently [30, 105]. In particular, when the constraint set is a linear constraint set, FW reduces the original problem to solving a sequences of linear programming problems; these problems can be solved by efficient linear programming techniques. Another example is the simplex constraint set; we show in Section 3.3 that many problems of interest

have this constraint set. As we discuss in Section 3.1.4, sub-problem (3.4a) admits a very simple solution under this constraint set.

Another advantage of FW over other methods such as ADMM or barrier methods is that it maintains feasibility by finding convex combinations of feasible points (see (3.4b)). It is important for applications that require feasibility of the solutions through the iterations. Moreover, in contrast to other optimization methods that generate feasible solutions, e.g., projected gradient descent, FW is projection-free. Projected gradient descent, projects the solution on the constraint set at each iteration to obtain a feasible solution. The projection on a constraint set requires minimizing a quadratic function, which measures the distance between the given point and a point on the set, subject to the original constraint set. As discussed in [80] for many constraint sets of interest solving (3.4a) with a linear objective is significantly cheaper than solving the projection problem with a quadratic function. In particular, projection of a matrix $X \in \mathbb{R}^{M \times N}$ on the set of bounded trace norm matrices requires finding the SVD decomposition of X, while solving (3.4a) is done by only finding the top-eigenvectors of X. The former has a time complexity of $O(NM^2)$, while for the latter fast algorithms, linear in the number of non-zero elements of X, exist [80].

Moreover, FW for some constraint sets, e.g., the simplex, generates sparse solutions with provable approximations [18]. Sparse solutions are often desired in practice; for example in [106] they are used to speed up the solutions of SVM problem. The solution to (3.4a) for the simplex is given by (3.9). This solution is extremely sparse, i.e., it has only one non-zero element. This along with the simple adaptation step (3.4b) ensures that the solution at k-iteration θ^k has at most k non-zero elements. Moreover, considering the convergence guarantees of FW, this sparse solution is within the $\frac{1}{k}$-neighborhood of the optimal solution.

3.1.2 FW Variants

There are several variants of FW in the literature. In general, these variants are divided to two groups; first group improves the convergence rate of FW [107, 108, 109, 110, 111, 112]. The other group reduces computation time of each iteration of FW by randomizing the algorithm, while obtaining convergence guarantees [80, 113, 114, 115]. As these algorithms are not readily parallelizable or applicable to the problems we consider here, we focus on parallelizing the classic FW in this thesis. We leave parallelization of these variants for future work. Nonetheless, for completeness, we explain some of these variants in this Section.

Fast-convergent variants. Frank and Wolfe [29] showed a convergence rate of $O(\frac{1}{\epsilon})$ for smooth

objectives. When the optimal solution lies at the boundary of the constraint set, FW converges slowly, i.e., the $O(\frac{1}{\epsilon})$ convergence rate is tight [116, 117, 29, 30]. This is because the iterations of the classic FW *zig-zag* between the vertices defining the face that contains the optimal solution. To avoid this zig-zagging phenomenon, Wolfe in [107] proposed a variant using 'away-points'; the basic idea is to move away from a 'bad' direction. Guélat and Marcotte [118] analyzed this further, and showed a linear convergence rate on polytope constraint sets. Several recently proposed FW variants improve the previous results for Away-steps Frank-Wolfe and attain linear convergence under weaker conditions [108, 109, 110, 111, 112].

Here, we introduce two of these variants, i.e., Away-step FW and Pairwise FW: Lacoste-Jullien and Jaggi [110] showed that for a strongly-convex objective and a polytope constraint set these variants enjoy a linear convergence rate. In particular, they solve (3.3) for a strongly-convex F and the constraint set

$$\mathcal{D} = \textbf{conv}(\mathcal{A}),$$

where $\mathcal{A}$ is a finite set of M points $a_1, \ldots, a_M \in \mathbb{R}^N$, called *atoms*. Note that (3.4a) for this constraint set takes the form:

$$a_{FW}^k = \arg\min_{a \in \mathcal{A}} \; a^\top \cdot \nabla F(\theta^k). \tag{3.7}$$

They refer to a_{FW}^k as *FW atom*. In both of the variants, the solution θ^k at each iteration k is a convex combination of the atoms:

$$\theta^k = \sum_{i=1}^{M} \gamma_i^k a_i.$$

At each iteration, they denote the *active* atoms by the set $\mathcal{A}^k = \{a_i \in \mathcal{A} : \gamma_i^k > 0\}$.

Away-step FW. The basic idea for Away-step FW is to mitigate the zig-zagging phenomenon by moving *away* from a *bad* direction, i.e., the direction d that maximizes the descent potential given by $P \equiv d^\top \cdot \nabla F(\theta)$. At each iteration k, Away-step FW, just like FW, finds the FW atom a_{FW}^k given by (3.7). Then, it finds an *away atom* given by:

$$a_A^k = \arg\max_{a \in \mathcal{A}^k} \; a^\top \cdot \nabla F(\theta^k). \tag{3.8}$$

Note that, here, the search region is over the selected atoms $\mathcal{A}^k \subseteq \mathcal{A}$, which is usually smaller than $\mathcal{A}$. Therefore, this optimization problem is easier than (3.7). Then, Away-step FW evaluates the descent potentials corresponding to the FW and away atoms, i.e., P_{FW}^k and P_{Away}^k, respectively

defined as:

$$P_{\text{FW}}^k = (a_{FW}^k - \theta^k)^\top \cdot \nabla F(\theta^k),$$
$$P_{\text{Away}}^k = (\theta^k - a_A^k)^\top \cdot \nabla F(\theta^k).$$

Then, depending on which potential is minimum, Away-step FW adapts the current solution: in case $P_{\text{FW}}^k < P_{\text{Away}}^k$, θ^{k+1} is given by:

$$\theta^{k+1} = \theta^k + \alpha(a_{FW}^k - \theta^k).$$

Note that this is exactly the adaptation step in the classic FW (3.4b). Otherwise, if $P_{\text{FW}}^k > P_{\text{Away}}^k$, the adaption has the form:

$$\theta^{k+1} = \theta^k + \alpha(\theta^k - a_A^k).$$

Note that here for $\alpha \geq 0$, θ^{k+1} is *not* a convex combination of the points θ^k, a_A^k. Hence, the step-size α is selected to ensure $\theta^{k+1} \in \mathcal{D}$.

Pairwise FW. The basic idea in Pairwise FW is to move weights only between two atoms. Formally, at each iteration k, Pairwise FW, similar to Away-step FW, finds the FW and away atoms, i.e., a_{FW}^k and a_A^k, respectively. Then, it adapts the solution θ^k by swapping weights between these two atoms, while keeping all other weights fixed:

$$\theta^{k+1} = \theta^k + \alpha(a_{FW}^k - a_A^k).$$

Note that this is different from classic FW, which shrinks all of the weights, except the FW atom weight, by a factor of $1 - \alpha$. Again, note that θ^{k+1} for $\alpha \geq 0$ is not a convex combination of the points θ^k, a_{FW}^k and a_A^k. Therefore, the step-size is set in a way that guarantees feasibility of θ^{k+1}. The convergence bound for Pairwise FW, proved in [110], is looser than Away-step FW. However, it works well in practice [110].

These two FW variants, as well as others in the literature [119, 120, 121, 110] converge faster than FW. However, as setting the step-size is more challenging and keeping track of the active atoms further complicates the algorithm, in this thesis we focus on parallelizing the classic FW.

Stochastic variants. Stochastic variants of FW have recently been proposed [80, 113, 114, 115], which stochastically approximate the gradient. In general, these variants solve convex optimization problems (3.3), where the objective F has a separable form:

$$F(\theta) = 1/M \sum_{i=1}^{M} F_i(\theta).$$

Moreover, they consider constraint sets $\mathcal{D}$, for which solving the linear sub-problem (3.4a) is easy, while projection over them is expensive. Hazan and Kale [80] list such constraint sets.

Hazan and Kale [80] focused on an online learning setting; however, their result is inferior to a trivial stochastic FW algorithm, called SFW in [114], which similar to SGD estimates the gradient ∇F by ∇F_i for some $i \in \{1, \ldots, M\}$, selected u.a.r. Lan and Zhou [113] introduced the Conditional Gradient Sliding method, which allows the algorithm to skip the computation of the gradient from time to time. They also proposed a stochastic version of this algorithm. Hazan and Lou [114] improved these results by using a *variance-reduced stochastic gradient* [122, 123] to estimate the gradient. Variance-reduced stochastic gradient provides an unbiased estimate of the gradient with a bounded variance, at the expense of computing the exact gradient for a point. Hazan and Lou [114] provide a thorough survey, and show that their algorithms outperform [80, 113] and SFW, in terms of the number of exact and stochastic gradient computations and the number of subproblems (3.4a) needed to obtain a solution within ϵ-neighborhood of the optimal. In particular, they reduce the number of stochastic gradient evaluations from $O(\frac{1}{\epsilon^2})$ to $O(\frac{1}{\epsilon^{1.5}})$ for smooth objective functions and from $O(\frac{1}{\epsilon})$ to $O(\ln \frac{1}{\epsilon})$ for smooth and strongly-convex objective functions. Reddi et al. [115] use the variance-reduced stochastic gradient idea to propose two stochastic variants of FW for non-convex objective functions. They show that both algorithms converge to a stationary point, with convergence rates faster than classic FW.

A different stochastic variants of FW solves problems with block-separable constraints [124]. Lacoste-Julien et al. in [124], proposed a random single-block FW algorithm, in which only a single block of variables, selected u.a.r., is updated. At the expense of computing the duality gap, the convergence result was improved in [125].

We implement two stochastic FW variants based on gradient subsampling: the basic idea is to compute the partial derivatives in only a random subset of the directions.We compare the relative performance of subsampling to increasing parallelism in Section 3.6.

3.1.3 Distributed implementations

More recently, and more relevant to our work, Bellet et al. [16] propose a distributed version of FW for objectives of the form $F(\theta) = g(A\theta)$, for some $A \in \mathbb{R}^{d \times N}$, where $d \ll N$. Several examples fall in this class, including two we study here (convex approximation and Adaboost); intuitively, $A\theta$ serves as the common information in our framework (c.f. Sec. 3.2). The authors characterize the message and parallel complexity when A is partitioned across multiple processors

under broadcast operations. Moreover, Tran et al in [104] elaborated on their algorithm, and proposed an asynchronous version of the distributed Frank-Wolfe algorithm in [16]. It is based on their *Stale Synchronous Parallel* (SSP) model [104]. They showed that the SSP based algorithm runs faster than the one based on a *Bulk Synchronous Parallel* (BSP) model, which is commonly used in distributed processing frameworks.

We differ from these implementations in the following two ways:

- We consider a broader class of problems, that do not abide by the structure presumed by Bellet et al. or Tran et al. (e.g., the two experimental design problems presented in Sec. 3.3). Our algorithm can be viewed as a generalization of their algorithm. In particular, our distributed algorithm can be applied to the problems they consider, i.e., $F(\theta) = g(A\theta)$, by defining the common information as $A\theta$ (see Prop. 1 and 2 in Section 3.2).
- We establish properties under which FW can be explicitly parallelized through map-reduce rather than the message passing environment proposed by Bellet et al. This allows us to leverage commercial map-reduce frameworks to readily implement and deploy parallel FW on a cluster.

3.1.4 Frank-Wolfe Over the Simplex

We focus on FW for the special case where the feasible set $\mathcal{D}$ is the simplex D_0, given by (3.2). As described in Section 3.3, this set of constraints arises in many problems, including training SVMs, convex approximation, Adaboost, and experimental design (see also [18]). Under this set of constraints, the linear optimization problem in (3.4a) has a simple solution: it reduces to finding the minimum element of the gradient $\nabla F(\theta^k)$. Formally, for $[N] \equiv \{1, 2, \dots, N\}$, and $\{e_i\}_{i\in[N]}$ the standard basis of $\mathbb{R}^N$, (3.4a) reduces to:

$$s^k = e_{i^*}, \text{ where } i^* \in \arg\min_{i\in[N]} \frac{\partial F(\theta^k)}{\partial \theta_i}. \tag{3.9}$$

Note that s^k is a vector in the standard basis of $\mathbb{R}^N$, for all $k \in \mathbb{N}$: as such, it is extremely sparse, having only one non-zero element. The sparsity of s^k plays a role in producing our efficient, distributed implementation, as discussed below.

3.2 Frank-Wolfe via Map-Reduce

3.2.1 Gradient Computation through Common Information

In this section, we identify two properties of function F under which FW over the simplex $\mathcal{D}_0$ admits a distributed implementation through map-reduce. Intuitively, our approach exploits an additional structure exhibited by several important practical problems: the objective function F often depends on the variables θ as well as a *dataset*, given as input to the problem. We represent this dataset through a matrix $X = [x_i]_{i\in[N]} \in \mathbb{R}^{N\times d}$ whose rows are vectors $x_i \in \mathbb{R}^d, i \in [N]$. The dataset can be large, as $N \gg 1$; as such, X may be horizontally (i.e., row-wise) partitioned over multiple processors. Note here that the dataset size (N) equals the number of variables in F.

We assume that the dependence of F to the dataset X is governed by two properties. The first property asserts that the partial derivative $\frac{\partial F}{\partial \theta_i}$ for any $i \in [N]$ depends on (a) the variable θ_i, (b) a datapoint x_i in the dataset, as well as (c) some *common information* h. This common information, not depending on i, fully abstracts any additional effect that θ and X may have on partial derivative $\frac{\partial F}{\partial \theta_i}$. Our second property asserts that this common information is *easy to update*: as variables θ^k are adapted according to the FW algorithm (3.4), the corresponding common information h can be re-computed efficiently, through a computation that does not depend on N. More formally, we assume that the following two properties hold:

Property 1. *There exists a matrix $X = [x_i]_{i\in[N]} \in \mathbb{R}^{N\times d}$, whose rows are vectors $x_i \in \mathbb{R}^d$, $i \in [N]$, such that for all $i \in [N]$:*

$$\frac{\partial F(\theta)}{\partial \theta_i} = G(h(X;\theta), x_i, \theta_i), \tag{3.10}$$

for some $h : \mathbb{R}^{N\times d} \times \mathbb{R}^N \to \mathbb{R}^m$, and $G : \mathbb{R}^m \times \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}$, where $m, d \ll N$.

We refer to h as the *common information* and to G as the *gradient function*. When $X \in \mathbb{R}^{d\times N}$ is partitioned over multiple processors, Prop. 1 implies that a processor having access to θ_i, x_i, and the common information $h(X;\theta)$ can compute the partial derivative $\frac{\partial F}{\partial \theta_i}$. No further information on other variables or datapoints is required other than h. Moreover, computing G is efficient, as its inputs are variables of size $m, d \ll N$.

Recall from (3.4) and (3.9) that, when the constraint set is the simplex, adaptations to θ^k take the form:

$$\theta^{k+1} = (1-\gamma^k)\theta^k + \gamma e_{i^*}, \quad \text{where } i^* \in \arg\min_{i\in[N]} \frac{\partial F(\theta^k)}{\partial \theta_i}.$$

Our second property asserts that when θ^k is adapted thusly, the common information h can be easily updated, rather than re-computed from scratch from X and θ^{k+1}:

Property 2. *Let $\mathcal{D} = \mathcal{D}_0$. Given $h(X;\theta^k)$, the common information at iteration k of the FW algorithm, the common information $h(X;\theta^{k+1})$ at iteration $k+1$ is:*

$$h(X;\theta^{k+1}) = H(h(X;\theta^k), x_{i^*}, \theta^k_{i^*}, \gamma^k), \tag{3.11}$$

for some $H : \mathbb{R}^m \times \mathbb{R}^d \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}^m$, where $i^* \in \arg\min_{i\in[N]} \frac{\partial F(\theta^k)}{\partial \theta_i}$.

Prop. 2, therefore, implies that a machine having access to x_{i^*}, $\theta^k_{i^*}$, γ^k, and the common information $h(X;\theta^k)$ in the last iteration can compute the new common information $h(X;\theta^{k+1})$. Again, no additional knowledge of X or θ^k is required. Moreover, similar to the computation of G in Prop. 1, this computation is efficient, as it again only depends on variables of size $m, d \ll N$. As we will see, in establishing that Prop. 2 holds for different problems, we leverage the sparsity of s^k at iteration $k \in \mathbb{N}$, as induced by (3.9): the fact that θ^k is interpolated with vector e_{i^*}, containing only a single non-zero coordinate, is precisely the reason why the common information can be updated efficiently.

Example: For the sake of concreteness, we give an example of an optimization problem over the simplex that satisfies Properties 1 and 2, namely, CONVEXAPPROXIMATION; additional examples are presented in Section 3.3. Given a point $p \in \mathbb{R}^d$ and N vectors $x_i \in \mathbb{R}^d, i \in [N]$, the goal of CONVEXAPPROXIMATION is to find the projection of p on the convex hull of set $\{x_i \mid i \in [N]\}$. This can be formulated as:

CONVEXAPPROXIMATION

$$\text{Minimize} \quad F(\theta) = \|X^T\theta - p\|_2^2 \tag{3.12a}$$

$$\text{subj. to:} \quad \theta \in \mathcal{D}_0, \tag{3.12b}$$

where $X = [x_i]_{i\in[N]} \in \mathbb{R}^{N\times d}$. CONVEXAPPROXIMATION satisfies Prop. 1 as

$$\frac{\partial F(\theta)}{\partial \theta_i} = 2x_i^T(X^T\theta - p) = G(h(X;\theta), x_i) \quad \text{for all } i \in [N],$$

where common information $h : \mathbb{R}^{N\times d} \times \mathbb{R}^N \to \mathbb{R}^d$ is

$$h(X;\theta) = X^T\theta - p, \tag{3.13}$$

Algorithm 4 SERIAL FW UNDER PROPERTIES 1 AND 2

1: Pick $\theta^0 \in \mathcal{D}$
2: $h := h(X; \theta^0)$
3: $k := 0$
4: **repeat**
5: **for** each $i \in [N]$ **do**
6: $z_i := G(h, x_i, \theta_i)$
7: **end for**
8: Find $i^* := \arg\min_{i \in [N]} z_i$
9: $\text{gap} := (\theta^k - e_{i^*})^\top z$
10: $\theta^{k+1} := (1 - \gamma^k)\theta^k + \gamma^k e_{i^*}$
11: $h := H(h, x_{i^*}, \theta^k_{i^*}, \gamma^k)$.
12: $k := k + 1$
13: **until** $\text{gap} < \epsilon$

and gradient function $G : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is

$$G(h, x) = 2x^T h.$$

Prop. 1 thus holds when $d \ll N$. Prop. 2 also holds because, under (3.4) and (3.9), the common information at step $k + 1$ is:

$$\begin{aligned} h(X; \theta^{k+1}) &= (1 - \gamma^k) h(X; \theta^k) + \gamma^k (x_{i^*} - p) \\ &= H(h(X; \theta^k), x_{i^*}, \gamma^k), \end{aligned}$$

where $H : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ is given by

$$H(h, x, \gamma) = (1 - \gamma)h + \gamma(x - p).$$

Note that, in this problem, $m = d \ll N$. Moreover, given their arguments, functions G and H can be computed in $O(d)$ time (i.e., their complexity does not depend on $N \gg 1$).

3.2.2 A Serial Algorithm

Before describing our parallel version of FW, we first discuss how it can be implemented serially when Properties 1 and 2 hold. The main steps are outlined in Alg. 4. Beyond picking an initial feasible point, the algorithm computes the initial value of the common information h. At each iteration of the for loop, the algorithm computes the gradient ∇F using the present common information, and updates both θ^k and the common information h to be used in the next step. It is easy to see that all steps in the main loop of Alg. 4 that involve computations depending on N (namely,

Lines 5–10) can be parallelized through map-reduce operations, when X and θ are distributed over multiple processors. We describe this in detail in the next section; crucially, the adaptation of the common information h (Line 11) does *not depend on* N, and can, therefore, be performed efficiently in one processor.

We note here that exploiting Properties 1 and 2 has efficiency advantages even in *serial execution*. In general, the complexity of computing the gradient ∇F as a function of $\theta \in \mathbb{R}^N$ may be quadratic in N, or higher, as each partial derivative $\frac{\partial F}{\partial \theta_i}$, $i \in [N]$, is a function of N variables. Instead, Properties 1 and 2 imply that the complexity of computing the gradient ∇F at each iteration of (3.4) is $O(N)$: this is the complexity when the common information is adapted through H and used to compute new partial derivatives through the gradient function G. For example, in the case of CONVEXAPPROXIMATION, the complexity is $O(Nd)$. As we show in Section 3.6, this leads to a significant speedup, allowing Alg. 4 to outperform interior-point methods even when executed serially.

3.2.3 Parallelization Through Map-Reduce

We now outline how to parallelize Alg. 4 through map-reduce operations. The algorithm is summarized in Alg. 5, where we use the notation $x \mapsto f(x)$ and $x, y \mapsto g(x, y)$, to indicate a unitary function f and a binary function g, respectively. The main data structure D contains tuples of the form (i, x_i, θ_i^k), for $i \in [N]$, partitioned and distributed over P processors. A master processor executes the map-reduce code in Alg. 5, keeping track of the common information h and the duality gap at each step. A `reduce` returns the computed value to the master, while a `map` constructs a new data structure distributed over the P processors.

Each step in the main loop of Alg. 4 has a corresponding map-reduce implementation in Alg. 5. In the main loop, a simple `map` using function G appends $z_i = \frac{\partial F(\ell^k)}{\partial \theta_i}$ to every tuple in D, yielding D' (Line 7 in Alg. 5). A `reduce` on D' (Line 8) computes a tuple $(i^*, x_{i^*}, \theta_{i^*}, z_{i^*})$, for $i^* \in \arg\min_{i \in [N]} z_i$. Similarly, a `map` and a `reduce` on D' (a summation) yields the duality gap (Line 9), while a `map` adapts the present solution θ in data structure D (Line 10). Finally, the common information h is adapted centrally at the master node (Line 11), as in Alg. 4.

Message and Parallel Complexity. The `reduce` in Line 8 requires $\log P$ parallel rounds, involving $P - 1$ messages of size $O(d)$ [102]. Computing the gradient in parallel through a `map` in Line 7 requires knowledge of the common information at each processor. Hence, in the beginning of each iteration, h is broadcast to the P processors over which D is distributed: this again requires in $\log P$

Algorithm 5 FW VIA MAP-REDUCE

1: Pick $\theta^0 \in \mathcal{D}$
2: Compute $h := h(X; \theta^0)$
3: Let $D := \{(i, x_i, \theta_i^0)\}_{i \in [N]}$
4: Distribute D over P processors
5: $k := 0$
6: **repeat**
7: $D' = D.\texttt{map}\big((i, x_i, \theta_i) \mapsto (i, x_i, \theta_i, G(h, x_i, \theta_i) \big)$
8: $(i^*, x_{i^*}, \theta_{i^*}, z_{i^*}) := D'.\texttt{reduce}\left((i, x_i, \theta_i, z_i), (i', x_{i'}, \theta_{i'}, z_{i'}) \mapsto \begin{cases} (i, x_i, \theta_i, z_i) & \text{if } z_i < z_{i'} \\ (i', x_{i'}, \theta_{i'}, z_{i'}) & \text{if } z_i \geq z_{i'} \end{cases} \right)$
9: $\text{gap} := D'.\texttt{map}\left((i, x_i, \theta_i, z_i) \mapsto \begin{cases} \theta_i \cdot z_i & \text{if } i \neq i^* \\ (\theta_i - 1) \cdot z_i & \text{if } i = i^* \end{cases} \right).\texttt{reduce}(+)$
10: $D := D.\texttt{map}\left((i, x_i, \theta_i) \mapsto \begin{cases} (i, x_i, (1-\gamma^k)\theta_i) & \text{if } i \neq i^* \\ (i, x_i, (1-\gamma^k)\theta_i + \gamma^k) & \text{if } i = i^* \end{cases} \right)$
11: $h := H(h, x_{i^*}, \theta_{i^*}, \gamma^k)$.
12: $k := k + 1$
13: **until** gap $< \epsilon$

rounds and $P - 1$ messages. Note that the corresponding message has size $O(m)$, that does not depend on N. Similarly, the reductions in Lines 9 and 10 require broadcasting i^*, which has size $O(1)$. In practice, such variables are typically shipped to the processors by the master along with the code of the function or operator to be executed by the corresponding `map` or `reduce`. The operations in Lines 7–10 thus require $\log P$ parallel rounds and the transmission of $O(P)$ messages of size $O(m + d)$.

3.2.4 Selecting the step size.

Our exposition so far assumes that the step size γ^k is computed at the master node before updating D and h. This is certainly the case if, e.g., $\gamma^k = \frac{2}{k+2}$, but it does not readily follow when the line minimization rule (3.6) is used. Nevertheless, all problems we consider here, including CONVEXAPPROXIMATION, satisfy an additional property that ensures that (3.6) can also be computed efficiently in a centralized fashion:

Property 3. *There exists an* $\hat{F} : \mathbb{R}^m \to \mathbb{R}$ *such that* $F(\theta) = \hat{F}\left(h(X; \theta)\right)$.

Prop. 3 implies that line minimization (3.6) at iteration k is:

$$\gamma^k = \operatorname*{arg\,min}_{\gamma \in [0,1]} \hat{F}\left(h(X; (1-\gamma)\theta^k + \gamma e_{i^*}) \right). \tag{3.14}$$

The argument of $\hat{F}$ is the updated common information h^{k+1} under step size γ. Hence, using Prop. 2, Eq. (3.14) becomes:

$$\gamma^k = \underset{\gamma \in [0,1]}{\arg\min}\ \hat{F}\left(H(h, x_{i^*}, \theta^k_{i^*}, \gamma)\right), \tag{3.15}$$

where h is the present common information. As F is convex in θ^k, it is also convex in γ, so (3.15) is also a convex optimization problem. Crucially, (3.15) depends on the full dataset X and the full variable θ only through h. Therefore, the master processor (having access to x_{i^*}, $\theta^k_{i^*}$, γ, and h) can find the step size via standard convex optimization techniques solving (3.15). In fact, for several of the problems we consider here, line minimization has a closed form solution; for example, for CONVEXAPPROXIMATION, the optimal step size is given by:

$$\gamma^k = \frac{h^\top h - (x_{i^*} - p)^\top h}{(x_{i^*} - p)^\top (x_{i^*} - p) + h^\top h - 2(x_{i^*} - p)^\top h}.$$

Though all problems we study, listed in Table 3.1, satisfy Prop. 1, 2, *as well as* 3, we stress again that Prop. 3 is not strictly necessary to parallelize FW, as a parallel implementation can always resort to a diminishing step size.

3.3 Examples

We provide several examples of problems that satisfy Prop. 1, 2, and 3; a summary is given in Table 3.1.

Problems	$F(\theta)$	m	G compl.	H compl.
Convex Approximation	$\|X\theta - p\|_2^2$	d	$O(d)$	$O(d)$
Adaboost	$\log\left(\sum_{j=1}^d \exp(Cc_j r_j)\right)$	d	$O(d)$	$O(d)$
D-optimal Design	$-\log\det A(\theta)$	d^2	$O(d^2)$	$O(d^2)$
A-optimal Design	$\text{trace}\left(A^{-1}(\theta)\right)$	$2d^2$	$O(d^2)$	$O(d^2)$

Table 3.1: Examples of problems satisfying Prop. 1–3.

Experimental Design: In experimental design, a learner wishes to regress a linear model $\beta \in \mathbb{R}^d$ from input data $(x_i, y_i) \in \mathbb{R}^d \times \mathbb{R}, i \in [N]$, where

$$y_i = \beta^\top x_i + \epsilon_i,$$

for ϵ_i, $i \in [N]$, i.i.d. noise variables. The learner has access to features $x_i, i \in [N]$, and wishes to determine which labels y_i to collect (i.e., which experiments to conduct) to accurately estimate β. This problem can be posed as [72]:

$$\min_{\theta \in \mathcal{D}_0} \mathtt{f}\left(\left(\sum_{i=1}^{N} \theta_i x_i x_i^\top\right)^{-1}\right), \tag{3.16}$$

where θ_i indicates the portion of experiments conducted by the learner with feature x_i. The quantity

$$A(X;\theta) = \sum_{i=1}^{N} \theta_i x_i x_i^\top$$

is the *design matrix* of the experiment. For brevity, we represent $A(X;\theta)$ as $A(\theta)$ below. Different choices of $\mathtt{f} : \mathbb{R}^{d\times d} \to \mathbb{R}$ lead to different optimality criteria; we review two below.

D-Optimal Design: In D-Optimal design $\mathtt{f}$ is the log-determinant, and (3.16) becomes:

$$\text{D-OPTIMALDESIGN}$$

$$\text{Minimize} \quad F(\theta) = \text{logdet}\left(\sum_{i=1}^{N} \theta_i x_i x_i^\top\right)^{-1} \tag{3.17a}$$

$$\text{subj. to:} \quad \theta \in \mathcal{D}_0, \tag{3.17b}$$

D-OPTIMALDESIGN satisfies Prop. 1 as:

$$\frac{\partial F}{\partial \theta_i} = -x_i^\top A^{-1}(\theta) x_i = G(h(X,\theta), x_i), \qquad \text{for all } i \in [N],$$

where the common information $h : \mathbb{R}^{N\times d} \times \mathbb{R}^N \to \mathbb{R}^{d\times d}$ is

$$h(X;\theta) = A^{-1}(\theta),$$

and the gradient function $G : \mathbb{R}^{d\times d} \times \mathbb{R}^d \to \mathbb{R}$, is given by

$$G(h,x) = -x^\top h x.$$

Hence, Prop. 1 holds when $d^2 \ll N$. Using the Sherman-Morrison formula [22] we can show that the common information at step $k+1$ is:

$$A^{-1}(\theta^{k+1}) = \frac{A^{-1}(\theta^k)}{1-\gamma} - \frac{\frac{\gamma}{(1-\gamma)^2} A^{-1}(\theta^k) x_{i^*} x_{i^*}^\top A^{-1}(\theta^k)}{1 + \frac{\gamma}{1-\gamma} x_{i^*}^\top A^{-1}(\theta^k) x_{i^*}}. \tag{3.18}$$

As a result,

$$h(X;\theta^{k+1}) = H(h(X,\theta^k), x_{i^*}, \gamma),$$

where $H : \mathbb{R}^{d\times d} \times \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^{d\times d}$ is:

$$H(h,x,\gamma) = \frac{h}{1-\gamma} - \frac{\frac{\gamma}{(1-\gamma)^2} hxx^\top h}{1+\frac{\gamma}{1-\gamma}x^\top hx}. \tag{3.19}$$

Therefore, Prop. 2 also holds. Note that, in this problem, $m = d^2 \ll N$. Functions G and H include only matrix-to-vector and vector-to-vector multiplications; hence, given their arguments, they can be computed in $O(d^2)$ time.

A-Optimal Design: In A-Optimal design `f` is the trace:

A-OPTIMALDESIGN

$$\text{Minimize} \quad F(\theta) = \text{Tr}\left(A^{-1}(\theta)\right) \tag{3.20a}$$

$$\text{subj. to:} \quad \theta \in \mathcal{D}_0. \tag{3.20b}$$

The partial derivative of the F can be written as:

$$\frac{\partial F}{\partial \theta_i} = -x_i^\top A^{-2}(\theta)x_i = G(h(X;\theta), x_i), \qquad \text{for all } i \in [N].$$

where the common information $h : \mathbb{R}^{N\times d} \times \mathbb{R}^N \to \mathbb{R}^{d\times d} \times \mathbb{R}^{d\times d}$ is

$$h(X;\theta) = (h_1, h_2),$$

where

$$h_1 = A^{-1}(\theta),$$
$$h_2 = A^{-2}(\theta).$$

The gradient function $G : \mathbb{R}^{d\times d} \times \mathbb{R}^d \to \mathbb{R}$ is

$$G((h_1,h_2),x) = -x^\top h_2 x.$$

Hence, Property 1 holds when $d^2 \ll N$. The common information at step $k+1$ is $\left(A^{-1}(\theta^{k+1}), A^{-2}(\theta^{k+1})\right)$. The first term can be computed as in (3.18). The second term is the square of the first term; expanding it gives a formula in terms of $A^{-1}(\theta^k)$ and $A^{-2}(\theta^k)$. More formally, the common information at iteration $k+1$ can be written as:

$$h(X;\theta^{k+1}) = (h_1^{k+1}, h_2^{k+1}) = H(h(X;\theta^k), x_{i^*}, \gamma),$$

where

$$H((h_1,h_2),x,\gamma) = (H_1(h_1,x,\gamma), H_2(h_1,h_2,x,\gamma)),$$

and function H_1 is given by (3.19), while $H_2 : \mathbb{R}^{d\times d} \times \mathbb{R}^{d\times d} \times \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^{d\times d}$ is:

$$H_2(h_1, h_2, x, \gamma) = \frac{h_2}{(1-\gamma)^2} - \frac{\frac{\gamma}{(1-\gamma)^3} h_2 x x^\top h_1}{1 + \frac{\gamma}{1-\gamma} x^\top h_1 x_i} - \frac{\frac{\gamma}{(1-\gamma)^3} h_1 x x^\top h_2}{1 + \frac{\gamma}{1-\gamma} x^\top h_1} + \frac{\frac{\gamma^2}{(1-\gamma)^4} x^\top h_2 x h_1 x x^\top h_2}{(1 + \frac{\gamma}{1-\gamma} x^\top h_1 x)^2}.$$

This illustrates why common information includes both $A^{-1}(\theta^k)$ and $A^{-2}(\theta^k)$: adapting the latter requires knowledge of both quantities. Note also that $m = 2d^2 \ll N$. Functions G and H again only require matrix-to-vector and vector-to-vector multiplications and, hence, can be computed in $O(d^2)$ time.

AdaBoost: Assume that N classifiers and ground-truth labels for d data points are given. The classification result is represented by a binary matrix $X \in \{-1, +1\}^{N\times d}$, where x_{ij} is the label generated by the i-th classifier for the j-th data point. The true classification labels are given by a binary vector $r \in \{-1, +1\}^d$. The goal of Adaboost is to find a linear combination of classifiers, defined as:

$$c(X, \theta) = X^\top \theta,$$

such that the mismatch between the new classifiers and ground-truth labels is minimized. The problem can be formulated as:

$$\textsc{AdaBoost}$$

$$\text{Minimize} \quad F(\theta) = \log\left(\sum_{j=1}^{d} \exp(-\alpha c_j(X,\theta) r_j)\right) \tag{3.21a}$$

$$\text{subj. to:} \quad \theta \in \mathcal{D}_0, \tag{3.21b}$$

where r_j and c_j are, respectively, the j th element of the r and c vectors, and $\alpha \in \mathbb{R}$ is a tunable parameter. Again, (3.21) satisfies Prop. 1 as:

$$\frac{\partial F(\theta)}{\partial \theta_i} = -x_i^\top b = G(h(X;\theta), x_i), \qquad \text{for all } i \in [N],$$

where $b \in \mathbb{R}^d$ is a vector, whose elements are

$$b_j = \frac{\alpha r_j \exp\left(-\alpha c_j r_j\right)}{\sum_{i=1}^d \exp(-\alpha c_j r_j)}, \qquad \text{for all } j \in [d].$$

The common information, $h : \mathbb{R}^{N\times d} \times \mathbb{R}^N \to \mathbb{R}^d$ is

$$h(X;\theta) = \left[\exp -\alpha c_j r_j\right]_{j\in[d]},$$

and the gradient function $G : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is

$$G(h, x) = x^\top \hat{h},$$

where

$$\hat{h} = \left[\frac{\alpha r_j h_j}{\sum_{i=1}^{d} h_i}\right]_{j\in[d]}.$$

Hence, Prop. 1 holds when $d \ll N$. Prop. 2 also holds because, under (3.4) and (3.9), the common information at step $k+1$ is

$$h(X;\theta^{k+1}) = H(h(X,\theta^k), x_{i^*}, \gamma),$$

where $H : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ is given by

$$H(h, x_i, \gamma) = \left[h_j^{(1-\gamma)} \exp(-\gamma\alpha x_{ji} r_j)\right]_{j\in[d]}.$$

In this problem, $m = d \ll N$ and functions G and H can be computed in $O(d)$ time.

Serial Solvers: All four problems in Table 3.1 are convex, and some admit specialized solvers. A-OPTIMALDESIGN can be reduced to a semidefinite program, (see Sec. 7.5 of [72]), and solved as an SDP. ADABOOST can be expressed as a geometric program (GP) [18], and CONVEXAPPROXIMATION is a quadratic program (QP). D-OPTIMALDESIGN is a general convex optimization problem, and can be solved by standard techniques such as, e.g., barrier methods. In Sec. 3.6 we compare FW to the above specialized solvers, and we see that it outperforms them in all cases.

3.4 Extensions

Our proposed distributed Frank-Wolfe algorithm can be extended to a more general class of problems, with constraints beyond the simplex.

ℓ_1-**constraint:** The ℓ_1 (or *lasso*) constraint $\|\theta\|_1 \le K$ appears in many optimization problems as means of enforcing sparsity [126, 127]. For this constraint, adaptation (3.4b) becomes:

$$s^k = \sigma_{i^*} e_{i^*}, \text{ where } i^* = \arg\max_{i\in[N]} \left|\frac{\partial f}{\partial \theta_i}\right|, \tag{3.22}$$

and $\sigma_{i^*} = -K \,\text{sign}(\frac{\partial f}{\partial \theta_{i^*}})$. Eq. (3.22) can be computed in parallel through a `reduce`. The adaptation step of γ^k is slightly different from the simplex case, as we interpolate between θ^k a scaled basis vector $\sigma_{i^*} e_{i^*}$.

As an example, consider the LASSO problem [127]:

$$\min_{\theta:\|\theta\|_1\le K} \|X^\top\theta - p\|_2^2. \tag{3.23}$$

Here, $\theta \in \mathbb{R}^N$ is the vector of weights, $X \in \mathbb{R}^{N\times d}$ is the matrix of $N-$dimensional features for d datapoints, and $p \in \mathbb{R}^d$ is the observed outputs. Note that LASSO has exactly the same objective as CONVEXAPPROXIMATION, so the common information from (3.13) is

$$h(X;\theta) = X^T\theta - p.$$

The common information can be updated as

$$h(X;\theta^{k+1}) = (1-\gamma^k)h(X;\theta^k) + \gamma^k(\sigma_{i^*}x_{i^*} - p),$$

i.e., it is a function of $h(X;\theta^k)$ and the usual "local" information at i^*, now including also σ_{i^*}.

Atomic Norms: More generally, consider the problem

$$\min_{\theta:\|\theta\|_{\mathcal{A}}\leq K} f(\theta),$$

where $\|x\|_{\mathcal{A}}$ denotes the *atomic norm*: given a set of atoms $\mathcal{A} = \{a_i \in \mathbb{R}^N\}$ the atomic norm is defined as

$$\|x\|_{\mathcal{A}} = \inf\{t \geq 0 : x \in t\mathbf{conv}(\mathcal{A})\},$$

where $\mathbf{conv}(\mathcal{A})$ is the convex hull of the atoms. Atomic norms are used to encourage solutions that have a low-dimensional structure, modelled as a linear combination of only few atoms [128, 129, 130, 105]. Tewari et al. [105] propose an FW-like algorithm for this class of problems. In this algorithm, the step 4 of Alg. 3 is replaced by

$$s^k = \arg\min_{a\in\mathcal{A}} a^\top \cdot \nabla F(\theta^k). \tag{3.24}$$

Then, the new solution is convex combination of the current solution and Ks^k, similar to FW Algorithm.

Our approach can be extended to problems of this form, where the set $\mathcal{A}$ comprises atoms $\{\pm\alpha_i e_i\}$, where $\alpha_i > 0$ s are arbitrary scalars. Eq. (3.24) becomes $s^k = -\alpha_{i^*}\,\text{sign}\,(\frac{\partial f}{\partial\theta_{i^*}})e_{i^*}$, where $i^* = \arg\max_{i\in[N]} |\alpha_i \frac{\partial f}{\partial\theta_i}|$. This can be implemented through a reduce, and adaptation is slightly different from the simplex case as again s^k is a scaled basis vector. An appropriate variant of Prop. 2, should hold w.r.t. this adaptation step.

3.5 Implementation

We implemented Alg. 5 over Spark, an open-source cluster-computing framework [33]. Spark inherently supports map-reduce operations, and is well-suited for parallelizing iterative

algorithms; this is because results of map-reduce operations can be cached in RAM, over multiple machines, and accessed in the next iteration of the algorithm [33].

Our FW implementation is generic, relying on an abstract class. A developer only needs to implement three methods in this class: (a) the gradient function G, (b) the common information function h, and (c) the common information adaptation function H. Once these functions are implemented, our code takes care of executing Alg. 5 in its entirety, and distributes its execution over a Spark cluster. Our implementation, which is publicly available,[1] can thus be used to solve arbitrary problems that satisfy Prop. 1 and 2, and quickly deploy and parallelize their execution over a Spark cluster. We have also instantiated this class for the problems summarized in Table 3.1 and used it in our experiments.

3.6 Experiments

3.6.1 Experiment Setup

Cluster. Our cluster comprises 8 worker machines. Each worker has 2 Intel(R) Xeon(R) CPUs (E5-2680 v4) with 2.4GHz clock speed and 14 cores, at 28 cores in total. Moreover, each core supports hyper-threading; as a result, each physical core appears as two logical cores to the operating system. Therefore, each worker has 2CPU $\times\, 14\frac{\text{cores}}{\text{CPU}} \times 2\frac{\text{threads}}{\text{core}} = 56$ threads (virtual cores), and the cluster has $8 \times 56 = 448$ (virtual) cores in total. Thus, the maximum level of parallelism for our cluster is 448. Also, each worker has 529 GB of RAM, 32KB L1 cache for instruction and data, 256KB for L2 cache, and 35.84MB for L3 cache. The cluster has 4TB of RAM in total. All code is implemented in Python (v2.7.5) and Spark (v1.4.1); we also use python's CVXOPT module (v1.1.8).
Algorithms. We solve Convex Approximation, Adaboost, D-Optimal Design, and A-Optimal Design summarized in Table 3.1, as well as LASSO (c.f. Sec. 3.4). We implement both serial and parallel solvers. First, we implement Serial FW (Alg. 4) in Python, setting γ using the line minimization rule (3.6). In addition, we solve Convex Approximation, D-Optimal Design, A-Optimal Design, and Adaboost using CVXOPT solvers, `qp`, `cp`, `sdp`, and `gp`, respectively. CVXOPT is a software package for convex optimization based on the Python programming language.[2] Beyond Serial FW and using CVXOPT solvers, we also implement a third, naïve serial algorithm in which the gradient is computed from scratch at each iteration, not exploiting the common information introduced in Prop. 1 and 2 (as Serial FW does). We call this implementation Oracle FW, as it computes the

[1] `https://github.com/neu-spiral/FrankWolfe`
[2] `cvxopt.org`

gradient via a "function oracle". We also implement our parallel algorithm (Alg. 5) using our Spark generic implementation. We again set the step size using the line minimization rule (3.6). We refer to this algorithm as Parallel FW. We control the level of parallelism, i.e., the number of cores P, by either setting the number of partitions of Spark resilient distributed datasets (RDDs) to P or by controlling the `total-executor-cores` in Spark's configuration and using a fixed high number of partitions, e.g., 600. We use the former approach when dealing with smaller datasets and the latter for larger datasets, as maintaining a large number of small partitions (executed serially) avoids memory crashes in Spark. We also introduce two stochastic parallel variants that subsample the gradient; we discuss these in Section 3.6.4. Finally, we also implement distributed ADMM for the LASSO problem, as described in Section 8.3 of [13].

Synthetic Data. For D-optimal Design, A-optimal Design, Convex Approximation, and LASSO, the synthetic data has the form of a matrix $X \in \mathbb{R}^{N\times d}$. The point p in Convex Approximation is a vector $p \in \mathbb{R}^d$. The elements of X and p are sampled independently from a uniform distribution in $[0, 1]$. For Adaboost, input data is given by a binary matrix $X \in \{-1, +1\}^{N\times d}$ and ground-truth labels are represented by a binary vector $r \in \{-1, +1\}^d$. The elements of r are sampled independently from a Bernoulli distribution with parameter 0.5. Then each row of X is generated from r as follows: each element x_{ij} is equal to r_j with probability 0.7, and it is equal to $-r_j$ with probability 0.3. For LASSO, the observed outputs are denoted by a vector $p \in \mathbb{R}^d$, which is generated as follows: a sparse vector $\theta^* \in \mathbb{R}^N$ is sampled from a uniform distribution in [0,1], s.t., only 1 percent of its elements are non-zero. Then the vector p is synthesized as $p = X^\top\theta^* + \epsilon$, where $\epsilon \in \mathbb{R}^d$ is the noise vector, and its elements are sampled from a uniform distribution in $[0, 0.01]$. We create three synthetic datasets with different values of N and d, summarized in Tables 3.2–3.4.

Real Data. We also experiment with 4 real datasets, summarized in Table 3.5. The first dataset is Movielens [131]. This includes 20,000,263 ratings for 27,278 movies generated by 138,493 users. We have kept the top 500 most-rated movies, resulting in 413,304 ratings, rated by 137,768 users. We have represented the data as a matrix $X \in \mathbb{R}^{N\times d}$ with $N = 137768$ and $d = 500$, so that x_{ij} indicates the rating of user i for movie j. Missing entries are set to zero. The second dataset is a high-energy physics dataset, HEPMASS [132]. The dataset has 10^6 data points and 28 features. We represent it as a matrix with $N = 10^6$ and $d = 28$. The third dataset is the MSD dataset [132], which comprises 515345 songs with 90 features. We represent it as a matrix with $N = 515345$ and $d = 90$. The fourth dataset is from Yahoo Webscope.[3] It represents a snapshot of the Yahoo!

[3] `https://webscope.sandbox.yahoo.com`

Music community's preferences for various songs. We used the test section of the dataset, which contains 18,231,790 ratings of 136,735 songs by over 1.8M users. We find the 100-dimensional latent vectors via matrix factorization technique [46], using the parameters $\mu = 0.001$ and $\lambda = 0.001$. We represent the latent vectors corresponding to users as a matrix $X \in \mathbb{R}^{N \times d}$ with $N = 1,823,179$ (number of users) and $d = 100$. We refer to this dataset as YAHOO dataset. When solving Convex Approximation problem for the YAHOO dataset, the vector $p \in \mathbb{R}^{100}$ is generated as follows. An arbitrary point from the dataset x_i is chosen, then it is corrupted by noise: $p = x_i + \epsilon$, where the elements of $\epsilon \in \mathbb{R}^{100}$ are sampled independently form a uniform distribution in $[0, 0.1]$. Finally, the point x_i is removed from the dataset.

Metrics. We use two metrics. The first is the objective F of each problem, whose evolution we track as different algorithms progress. Our second metric is t_ϵ, the minimum time for the algorithm to obtain a solution θ within an ϵ-neighborhood of the optimal solution $F(\theta^*)$. As we do not know $F(\theta^*)$, we use $F(\theta) - g(\theta) \leq F(\theta^*)$ instead. More formally:

$$t_\epsilon = \min \left\{ t : \tfrac{F(\theta(t))}{F(\theta(t)) - g(\theta(t))} \leq 1 + \epsilon \right\}, \tag{3.25}$$

where $\theta(t)$ denotes the obtained solution at time t. As $F(\theta) - g(\theta) \leq F(\theta^*)$, t_ϵ overestimates the time to convergence.

3.6.2 Serial Execution

Our first experiment compares the Serial FW algorithm with (a) the specialized interior point solvers mentioned in Section 3.3 (i.e., `cp`, `qp`, `sdp`, and `gp`) and (b) with Oracle FW, for each of the problems in Table 3.1. We use the small synthetic dataset (Dataset A) in Table 3.2.

In each execution, we keep track of the objective function F as a function of time elapsed. Unlike FW, the interior-point methods do not generate feasible solutions at each iteration. Therefore, we project the solutions at each iteration on the feasible set, and compute the objective F on the projected solution. The time taken for the projection is not considered in time measurements; as such, our plots underestimate the time taken by the interior-point algorithms.

Fig. 3.1 shows function values generated by the algorithms as a function of time. Serial FW outperforms the interior-point methods, even when not accounting for projections. The reason is that, in contrast to interior-point methods, the time complexity of computations at each iteration of Serial FW is linearly dependent on N. As a result, when $d \ll N$, Serial FW is considerably faster, even

Problem	N	d	algs
Conv. Approx.	5000	20	`qp`
Adaboost	5000	100	`gp`
D-opt. Design	5000	20	`cp`
A-opt. Design	5000	20	`sdp`

Table 3.2: Dataset A

Problem	N	d	ϵ
Conv. Approx.	100M	100	0.15
Adaboost	100M	100	0.004
D-opt. Design	20M	100	0.09
A-opt. Design	20M	100	0.19

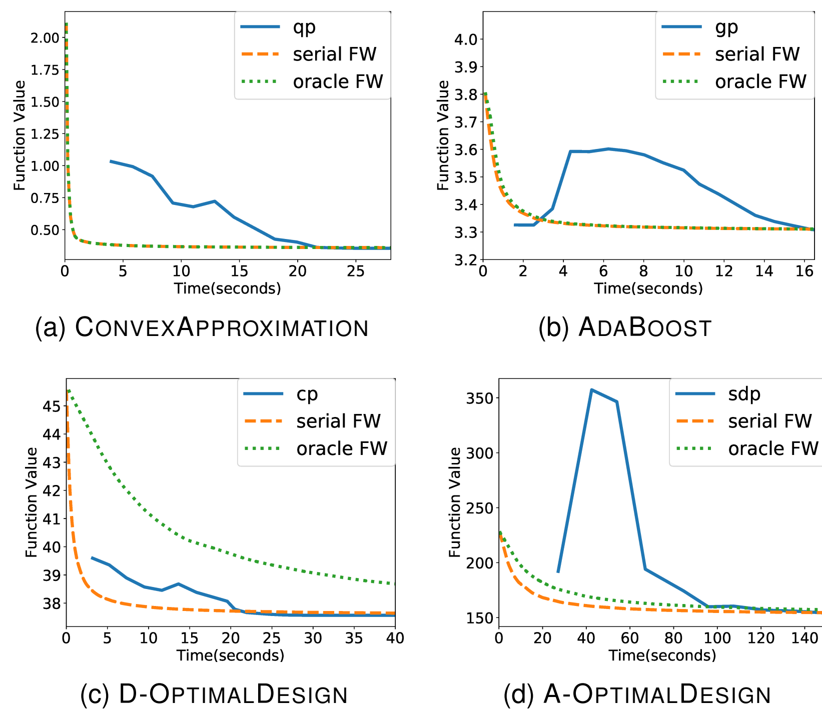
Table 3.3: Dataset B

Problem	N	d	ϵ
Conv. Approx.	500000	5000	0.13
Adaboost	500000	5000	0.003
D-opt. Design	100000	1000	0.03
A-opt. Design	100000	1000	0.09

Table 3.4: Dataset C

Problem	Dataset	N	d	ϵ
D-opt. Design	Movielens	137768	500	0.18
D-opt. Design	HEPMASS	1M	38	0.04
D-opt. Design	MSD	515345	90	0.01
D-opt. Design	YAHOO	1,823,179	100	0.09
A-opt. Design	YAHOO	1,823,179	100	0.17
Conv. Approx.	YAHOO	1,823,178	100	0.03

Table 3.5: Real Datasets



(a) CONVEXAPPROXIMATION (b) ADABOOST

(c) D-OPTIMALDESIGN (d) A-OPTIMALDESIGN

Figure 3.1: Values of the objective function generated by the algorithms as a function of time over Dataset A. We see that Serial FW converges faster than interior point methods. Comparing it to Oracle FW, the benefits of exploiting the common information in serial computation are more pronounced for experimental design objectives, where partial derivative computation is quadratic.

though it requires more iterations to converge. Note that the objective values generated by interior-point methods are non-monotone, as these methods alternate between improving feasibility and optimality. Comparing Serial FW to Oracle FW, the benefits of exploiting the common information in serial computation are more pronounced for experimental design objectives, where partial derivative computation is quadratic.

3.6.3 Effect of Parallelism

We compare the speedup of Parallel FW over three serial implementations. The first is Parallel FW with $P = 1$, i.e., our parallel Spark code using only one processor. The second is Serial FW, as described in Alg. 4; the third is Oracle FW, which computes the gradient naïvely from scratch, not exploiting the common information introduced in Prop. 1 and 2. We do not report results of serial execution via CVXOPT, as the latter crashes with out-of-memory errors on all these inputs. We execute 10 iterations of these serial implementations and then estimate the total running time based on the average per-iteration running time; all values reported correspond to the same number of iterations.

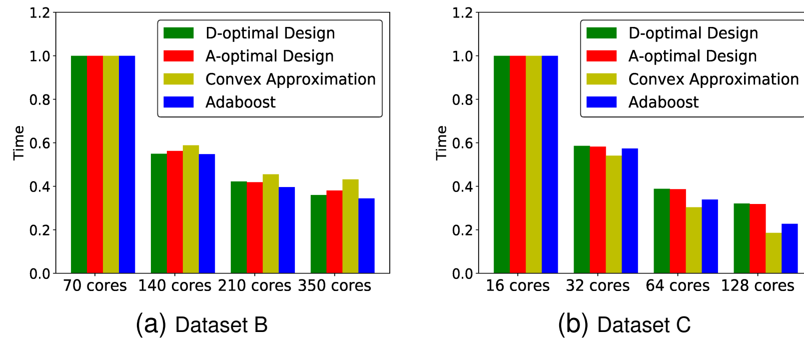
The measured speedups of Parallel FW over these serial implementations are shown in Table 3.6. Increasing parallelism leads to significant speedups. For example, using 350 compute cores, we can solve the 20M-variable instance of D-optimal Design in 79 minutes, when Serial FW would take and 48.3 hours for the same problem and input. Note that, even in serial implementation, exploiting Properties 1-3 leads to accelarated execution: this is evident from the fact that the speedup over Oracle FW is considerably higher than over Serial FW. This is more prominent in the two experimental design objectives: this is expected, as the complexity of computing the gradient is $O(Nd^3)$, while by using the common information we can compute the gradient in $O(Nd^2)$. For example, the same 20M-variable D-optimal instance would take more than 14 days for Oracle FW.

We note that, for the input sizes used in these experiments, the benefit of parallelism saturates beyond 350 cores and 128 cores, for Datasets B and C, respectively. The reason is that for this input size, after increasing the level of parallelism beyond these values, the cost of computing the gradient at each core becomes negligible.

We further illustrate the effect of increasing parallelism on two large-scale synthetic datasets: Dataset B, a dataset with $N = O(\text{20M})$ and $d = 100$ (Table 3.3), and Dataset C with $N = O(\text{100K})$ and $d = O(\text{1K})$ (Table 3.4). Fig. 3.2 shows t_ϵ as a function of the level of parallelism, measured in terms of the number of cores P, for each of the two datasets. We normalize t_ϵ by its

Problem	Dataset	Speedup vs. Parallel FW with P=1	Speedup vs. Serial FW	Speedup vs. Oracle FW	# of cores
Conv. Approx.	Dataset C	42	1.12	1.45	128
Conv. Approx.	Dataset B	98	28.57	29.82	350
Conv. Approx.	YAHOO	78	18.6	21.83	210
Adaboost	Dataset C	45	1.24	2.07	128
Adaboost	Dataset B	133	35.4	35.47	350
D-opt. Design	Dataset C	48	12.7	112.3	128
D-opt. Design	Dataset B	126	36.7	271.6	350
D-opt. Design	HEPMASS	35	7.5	23.28	64
D-opt. Design	Movielens	33	3.77	115.5	64
D-opt. Design	MSD	35	6.52	37.24	64
D-opt. Design	YAHOO	93	19	159.1	210
A-opt. Design	Dataset C	49	10.5	125.07	128
A-opt. Design	Dataset B	102	32.5	273.12	350
A-opt. Design	YAHOO	90	20.3	164.95	210

Table 3.6: A summary of speedups (over three serial implementations) obtained by parallel FW for each problem and dataset, along with the level of parallelism. Beyond this number of cores, no significant speedup improvement is observed.



(a) Dataset B (b) Dataset C

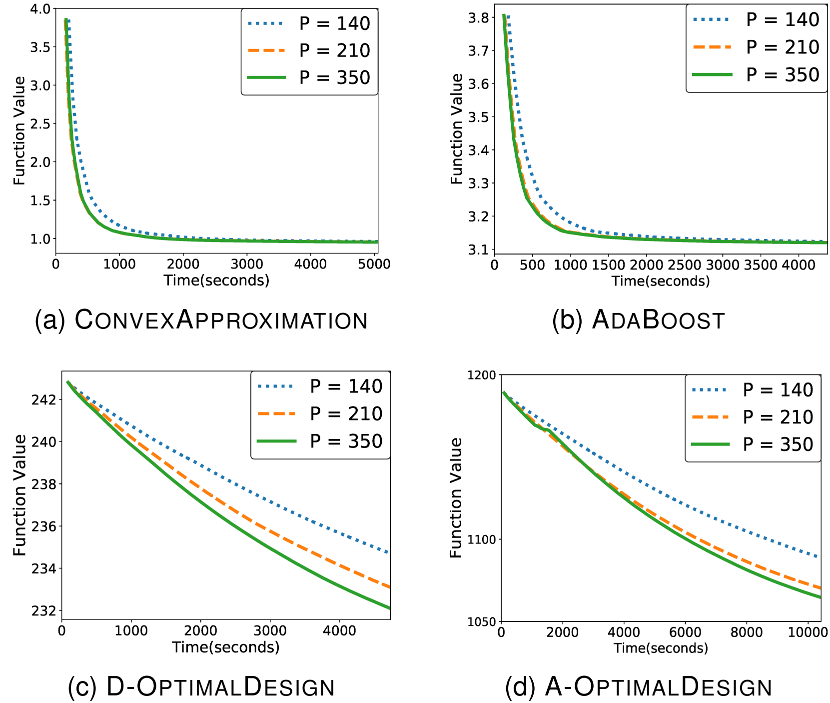
Figure 3.2: The t_ϵ as a function of the level of parallelism, measured in terms of cores P. Fig. 3.2a shows results on the $O(20\text{M})$ variable dataset (Table 3.3) while Fig. 3.2b shows results on the dataset with $d = O(1000)$ (Table 3.4). We normalize t_ϵ by its value at the lowest level of parallelism (13134s, 27420s, 11727s, and 11375s, respectively, for each of the four problems in Fig. 3.2a and 487s, 486s, 2096s, and 4783s, respectively, in Fig. 3.2b). We see that increasing the level of parallelism speeds up convergence.

value at $P = 70$ and $P = 16$, respectively. This lowest level of parallelism ($P = 70$ and $P = 16$) is chosen so that the slowest execution time is moderate, i.e., approximately 10 hours. Figure 3.3 shows objective F, as a function of time for different levels of parallelism. The highest level of parallelism (e.g., 350 for dataset B) is the saturation point, beyond which no significant speedup is observed. By comparing Figures 3.3a and 3.3b with Figures 3.3c and Figure 3.3d, we see that Parallel FW converges much faster for Convex Approximation and Adaboost. The reason is that the objective function in D-Optimal Design and A-optimal Design does not have a bounded curvature; therefore, as mentioned in Section 3.2, FW for these problems does not have a $O(\frac{1}{k})$ convergence rate.

We also illustrate how parallelism affects performance on real datasets, summarized in Table 3.5. For brevity, we only report D-Optimal Design for Movielens, HEPMASS, and MSD datasets, and D-optimal design, A-optimal Design, and Convex Approximation for the YAHOO dataset. Fig. 3.4 shows the measured t_ϵ for different levels of parallelism. For each dataset, t_ϵ is normalized by the value of t_ϵ for the lowest level of parallelism. Again, we see that we gain a significant speedup by parallelism.

3.6.4 Subsampling the Gradient

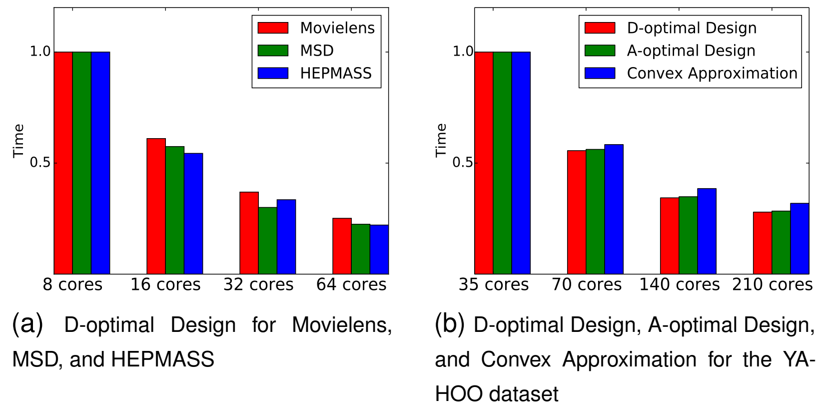
In this section, we study the effect of subsampling the gradient on the performance of FW. We have seen that parallelism reduces the cost of computation of the gradients. An alternative is to compute the gradient stochastically by subsampling only a few partial derivatives and using the



(a) CONVEXAPPROXIMATION (b) ADABOOST

(c) D-OPTIMALDESIGN (d) A-OPTIMALDESIGN

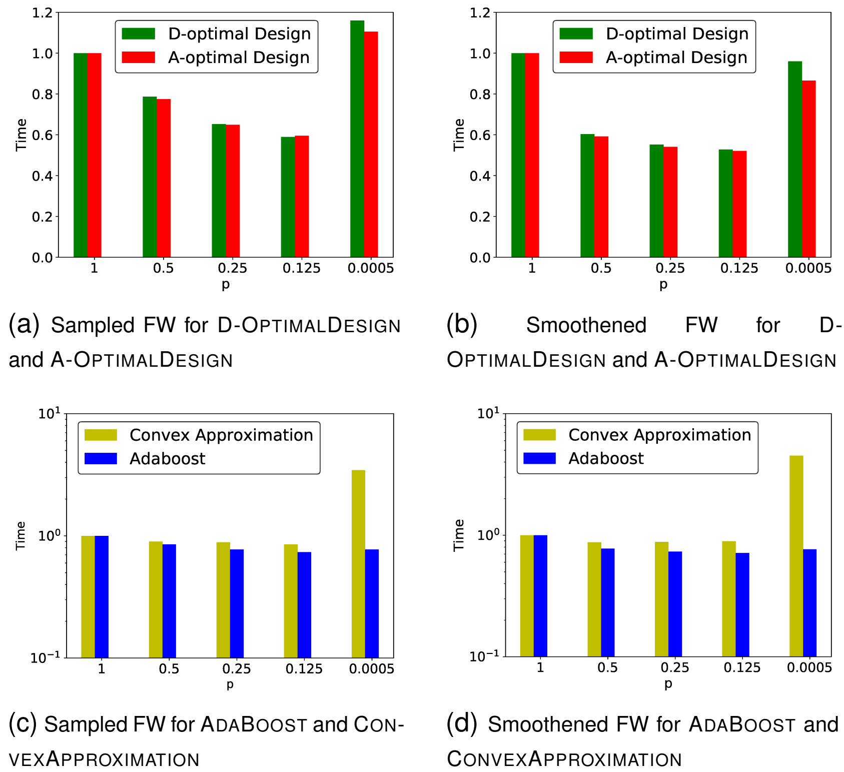
Figure 3.3: The objective F as a function of time over Dataset B. We see that increasing the level of parallelism makes convergence faster. By comparing Figures 3.3a and 3.3b with Figures 3.3c and 3.3d, we see that FW for D-Optimal Design and A-Optimal Design converges slower.



(a) D-optimal Design for Movielens, MSD, and HEPMASS

(b) D-optimal Design, A-optimal Design, and Convex Approximation for the YAHOO dataset

Figure 3.4: The summary of parallelism experiments on the real datasets. We normalize t_ϵ by its value at the lowest level of parallelism (15247s, 3899s, and 4766s for Movielens, MSD, and HEPMASS, respectively, in Fig. 3.4a, and 9888s, 7060s, and 1302s for D-optimal Design, A-optimal Design, and Convex Approximation, respectively, in Fig 3.4b.



(a) Sampled FW for D-OPTIMALDESIGN and A-OPTIMALDESIGN

(b) Smoothened FW for D-OPTIMALDESIGN and A-OPTIMALDESIGN

(c) Sampled FW for ADABOOST and CONVEXAPPROXIMATION

(d) Smoothened FW for ADABOOST and CONVEXAPPROXIMATION

Figure 3.5: The measured t_ϵ under Sampled and Smoothened FW, over Dataset C. We normalize t_ϵ by the measured t_ϵ for 16 cores, which is reported in Fig. 3.2. By comparing Figures 3.5a and 3.5c with Fig. 3.2b, subsampling does not match the benefits of parallelism. In an ultra-low regime, e.g., $p = 0.0005$ convergence is very slow. Smoothened FW can enhance the performance in this case.

minimal in this sub-sampled set. This reduces the amount of computation occurring in each iteration. Moreover, such a stochastic estimation of the gradient still guarantees convergence [115], albeit at a slower rate. Therefore, subsampling decreases the computation time for each iteration; this has a similar effect to increasing parallelism, without incurring additional communication overhead. In contrast to increasing parallelism, however, subsampling may also increase the number of iterations till convergence.

We consider two variants of subsampling. In Sampled FW, we compute each partial derivative $\frac{\partial F}{\partial \theta_i}$ with probability p. Then, we find the minimum among the computed partial derivatives. Note that this speeds derivative computations: at most $p \cdot N$ partial derivatives are computed, in expectation. In Smoothened FW, we compute each partial derivative with probability p, but maintain an exponentially-weighted moving average (EWMA) between the computed value and past values: this estimate is used instead to compute the current minimum partial derivative.

We use Dataset C (Table 3.4) in this experiment: we solve the corresponding problems using Sampled FW and Smoothened FW on 16 cores. The results are shown in Fig. 3.5. Values t_ϵ are normalized by t_ϵ for $p = 1$. This makes experiments in Figures 3.5 and 3.2b comparable: each core computes the same number of partial derivatives in expectation.

By comparing Figures 3.5 and 3.2b, we see that subsampling matches the benefits of parallelism, at least for large p, for D-optimal and A-optimal design. In contrast, the benefits of subsampling for Convex Approximation and AdaBoost are almost negligible. This is because Parallel FW guarantees a $O(\frac{1}{k})$ convergence rate for these problems. As a result, though subsampling reduces the cost of computation per iteration, the increase in number of iterations negates this advantage. In fact, when p is in an ultra-low regime, e.g., $p = 0.0005$, Sampled FW converges extremely slowly for *all problems*. Interestingly, Smoothened FW performs better in this case, ameliorating the performance deterioration. This is most evident in Figures 3.5d and 3.5c, where t_ϵ for Convex Approximation and AdaBoost is considerably smaller under Smoothened FW.

3.6.5 LASSO Experiment

To show the performance of our algorithm on the cases beyond simplex constrained problems, we solve the LASSO problem (3.23). We compare our distributed FW with distributed ADMM.

The input data is synthetic and with $N = 100,000$ and $d = 1000$. First, we solve the following problem:

$$\min_\theta \frac{1}{2}\|X^\top\theta - p\|_2^2 + \|\theta\|_1,$$

with distributed ADMM using 400 cores and for different values of ρ, which is a parameter controlling convergence (see Section 8.3 of [13]). We then solve the LASSO with our Distributed FW algorithm, setting K equal to the ℓ_1 norm of the solution obtained by ADMM. For a fair comparison, we use 400 cores. Fig. 3.6 shows the value of the squared loss $\frac{1}{2}\|X\theta - p\|_2^2$ as a function of time for FW and ADMM. As we see, FW outperforms ADMM.

3.7 Conclusion

We establish structural properties under which FW admits a highly scalable parallel implementation via map-reduce. We show that problems distributed by our algorithm achieve significant speedups. In particular, by using 350 cores we are able to solve a problem with 20 million

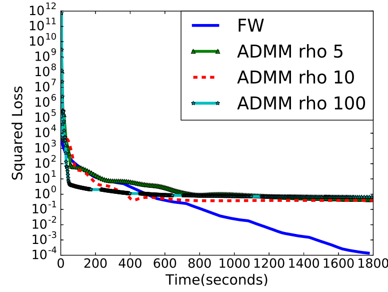


Figure 3.6: The comparison between ADMM and our distributed Frank-Wolfe algorithm. Each algorithm uses 400 cores.

variables in 79 minutes, while the serial implementation takes 165 hours. Moreover, we show that our results extend beyond the simplex constraint. For instance, our distributed FW algorithm can be applied to problems with the popular and widely-used ℓ_1-norm constraint.

The Frank-Wolfe algorithm is related to approximate algorithms for so-called *submodular maximization problems*. *Submodularity* is a structural property associated with *set functions*. Submodularity captures the notion of *diminishing returns* or *decreasing marginal utilities* [133]; this makes it a suitable objective in computer science or economics to represent subset evaluations of, e.g, a set of utilities [133]. In particular, *maximizing submodular functions subject to matroid constraints* has numerous applications in the combinatorial optimization domain, such as variable selection [134], dictionary learning [135, 136], document summarization [137, 138], etc. Maximizing submodular functions subject to matroid constraints are known as the convex optimization counterpart in the combinatorial optimization domain [133]: though NP-hard, these problems can be solved with approximate guarantees [39, 139, 140]. A *greedy* algorithm [139] produces a solution that is guaranteed to be within $1/2$ ratio of the optimal solution. The so-called *continuous greedy algorithm* improved this ratio to $(1 - 1/e)$ [141, 40]; moreover, this ratio cannot be improved further for polynomial-time algorithms [142]. The continuous greedy algorithm is in fact a variant of FW. It maximizes a *multi-linear relaxation* [39] of the original submodular maximization problem. Through this connection, FW has important applications in solving these combinatorial optimization problems. Parallelizing this variant of FW, which solves generic submodular optimization problems with guarantees is an important direction as a future work.

Though submodularity is conventionally defined for set functions, its definition has also been extended for continuous functions [143, 144]. More recently, Bian et al. [140] defined *DR-submodular* functions, which are a subset of the continuous submodular functions. The scope of the DR-submodular functions comprises a subset of convex functions, a subset of concave functions, and

a subset of functions that are neither convex nor concave. Moreover, they show that many interesting computer science problems such as, maximzing linear extensions, e.g., the Lovasz extension [145], of submodular set functions, non-convex/non-concave quadratic programming, optimal budget allocation, etc, have DR-submodular objectives. They also prove that a FW variant, which is similar to the continuous greedy algorithm, maximizes the monotone DR-submodular functions again with the $(1 - 1/e)$ ratio within the optimal solution. In other words, this FW variant, with guarantees, solves a class of generic problems with diverse applications, which interestingly includes non-convex optimization problems. In particular, we apply the continuous greedy algorithm in the next chapter for designing cache networks. Moreover, another useful area as future work is parallelizing this FW variant: this allows to solve large-scale monotone DR-submodular maximization problems in a moderate time.

Chapter 4

Design of Kelly Cache Networks via Submodular Maximization

Kelly networks [146] are multi-class networks of queues capturing a broad array of queue service disciplines, including FIFO, LIFO, and processor sharing. Both Kelly networks and their generalizations (including networks of quasi-reversible and symmetric queues) are well-studied, classic topics [146, 147, 148, 149]. One of their most appealing properties is that their steady-state distributions have a product-form: as a result, steady state properties such as expected queue sizes, packet delays, and server occupancy rates have closed-form formulas as functions of, e.g., routing and scheduling policies.

Here, we consider Kelly networks in which nodes are equipped with caches, i.e., storage devices of finite capacity, which can be used to store objects. Exogenous requests for objects are routed towards nodes that store them; upon reaching a node that stores the requested object, a response packet containing the object is routed towards the request source. As a result, object traffic in the network is determined not only by the demand but, crucially, by where objects are cached. This abstract setting is motivated by–and can be used to model–various networking applications involving the placement and transmission of content. This includes information centric networks (ICNs) [150, 151, 7], content delivery networks (CDNs) [152, 153], web-caches [154, 155, 156], wireless/femtocell networks [157, 158, 159], and peer-to-peer networks [160, 161], to name a few.

In many of these applications, determining the *object placement*, i.e., how to place objects in network caches, is a decision that can be made by the network designer in response to object popularity and demand. To that end, we are interested in determining how to place objects in caches

so that traffic attains a design objective such as minimizing delay. Several papers have studied the cache optimization problems under restricted topologies [162, 163, 164, 165, 152]. These works model the network as a bipartite graph: nodes generating requests connect directly to caches in a single hop. The resulting algorithms do not readily generalize to arbitrary topologies. In general, the approximation technique of Ageev and Sviridenko [166] applies to this bipartite setting, and additional approximation algorithms have been devised for several variants [162, 163, 164, 152]. We differ by (a) considering a multi-hop setting, and (b) introducing queuing, which none of the above works considers.

Submodular function maximization subject to matroid constraints appears in many important problems in combinatorial optimization; for a brief review of the topic and applications, see [91] and [133], respectively. Nemhauser et al. [38] show that the greedy algorithm produces a solution within 1/2 of the optimal. Vondrák [141] and Calinescu et al. [40] show that the continuous-greedy algorithm produces a solution within $(1 - 1/e)$ of the optimal in polynomial time, which cannot be further improved [142]. In the general case, the continuous-greedy algorithm requires sampling to estimate the gradient of the so-called multilinear relaxation of the objective (see Sec. 4.2.1). One of our main contributions is to show that MAXCG, the optimization problem we study here, exhibits additional structure: we use this to construct a sampling-free estimator of the gradient via a power-series or Taylor expansion. To the best of our knowledge, we are the first to use such an expansion to eschew sampling; this technique may apply to submodular maximization problems beyond MAXCG.

In particular, we make the following contributions.

- We study the problem of optimizing the placement of objects in caches in Kelly cache networks of M/M/1 queues, with the objective of minimizing a cost function of the system state. We show that, for a broad class of cost functions, including packet delay, system size, and server occupancy rate, *this optimization amounts to a submodular maximization problem with matroid constraints*. This result applies to general Kelly networks with fixed service rates; in particular, it holds for FIFO, LIFO, and processor sharing disciplines at each queue.

- The so-called continuous greedy algorithm [40] attains a $1 - 1/e$ approximation for this NP-hard problem. However, it does so by computing an expectation over a random variable with exponential support via randomized sampling. The number of samples required to attain the $1 - 1/e$ approximation guarantee can be prohibitively large in realistic settings. Our second contribution is to show that, for Kelly networks of M/M/1 queues, *this randomization can be*

entirely avoided: a closed-form solution can be computed using the Taylor expansion of our problem's objective. To the best of our knowledge, we are the first to identify a submodular maximization problem that exhibits this structure, and to exploit it to eschew sampling.

- Finally, we extend our results to *networks of M/M/k and symmetric M/D/1 queues*, and prove a negative result: submodularity does *not* arise in networks of M/M/1/k queues. We extensively evaluate our proposed algorithms over several synthetic and real-life topologies.

The remainder of this chapter is organized as follows. We present our mathematical model of a Kelly cache network along with our problem formulation in Sec. 4.1. We show our results on submodularity, the continuous-greedy algorithm in networks of M/M/1 queues, and our novel algorithm in Section 4.2. Our extensions are described in Sec. 4.3; our numerical evaluation is in Sec. 4.4. Finally, we conclude and explain several recent developments related to our study in Sec. 4.5.

4.1 Model

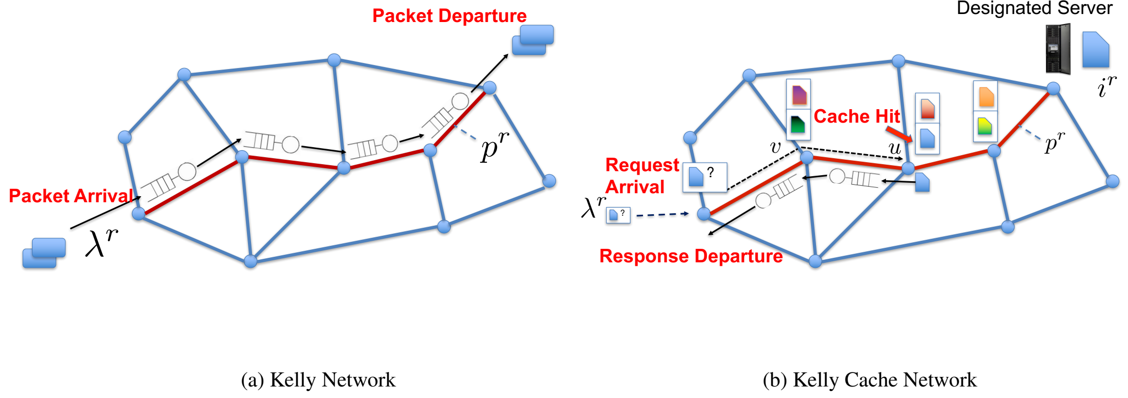
4.1.1 Kelly Cache Networks

Motivated by applications such as ICNs [150], CDNs [152, 153], and peer-to-peer networks [160], we introduce Kelly cache networks. In contrast to classic Kelly networks, each node is associated with a cache of finite storage capacity. Exogenous traffic consisting of *requests* is routed towards nodes that store objects; upon reaching a node that stores the requested object, a *response* packet containing the object is routed towards the node that generated the request. As a result, content traffic in the network is determined not only by demand but, crucially, by how contents are cached. An illustration highlighting the differences between Kelly cache networks, introduced below, and classic Kelly networks, can be found in Fig. 4.1.

Although we describe Kelly cache networks in terms of FIFO M/M/1 queues, the product form distribution (c.f. (4.7)) arises for many different service principles beyond FIFO (c.f. Section 3.1 of [146]) including Last-In First-Out (LIFO) and processor sharing. All results we present extend to these service disciplines; we discuss more extensions in Sec. 4.3.

Kelly Cache Networks. We now formally describe Kelly cache networks.

Graphs and Paths. We use the notation $G(V, E)$ for a directed graph G with nodes V and edges $E \subseteq V \times V$. A directed graph is called *symmetric* or *bidirectional* if $(u, v) \in E$ if and only if $(v, u) \in E$. A *path* p is a sequence of adjacent nodes, i.e., $p = p_1, p_2, \ldots, p_K$ where $(p_k, p_{k+1}) \in E$,



(a) Kelly Network

(b) Kelly Cache Network

Figure 4.1: (a) Example of a Kelly network. Packets of class r enter the network with rate λ^r, are routed through consecutive queues over path p^r, and subsequently exit the network. (b) Example of a Kelly cache network. Each node $v \in V$ is equipped with a cache of capacity c_v. Exogenous requests of type r for object i^r enter the network and are routed over a predetermined path p^r towards the designated server storing i^r. Upon reaching an intermediate node u storing the requested object i^r, a response packet containing the object is generated. The response is then forwarded towards the request's source in the reverse direction on path p^r. Request packets are of negligible size compared to response messages; as a result, we ignore request traffic and focus on queuing due to response traffic alone.

for all $1 \leq k < K \equiv |p|$. A path is *simple* if it contains no loops (i.e., each node appears once). We use the notation $v \in p$, where $v \in V$, to indicate that node v appears in the path, and $e \in p$, where $e = (u, v) \in E$, to indicate that nodes u,v are two consecutive (and, therefore, adjacent) nodes in p. For $v \in p$, where p is simple, we denote by $k_p(v) \in \{1, \ldots, |p|\}$ the position of node $v \in V$ in p, i.e., $k_p(v) = k$ if $p_k = v$.

Network Definition. Formally, we consider a Kelly network of M/M/1 FIFO queues, represented by a symmetric directed graph $G(V, E)$. As in classic Kelly networks, each edge $e \in E$ is associated with an M/M/1 queue with service rate μ_e[1]. In addition, each node has a cache that stores objects of equal size from a set $\mathcal{C}$, the *object catalog*. Each node $v \in V$ may store at most $c_v \in \mathbb{N}$ objects from $\mathcal{C}$ in its cache. Hence, if $x_{vi} \in \{0, 1\}$ is a binary variable indicating whether node $v \in V$ is storing object $i \in \mathcal{C}$, then

$$\sum_{i \in \mathcal{C}} x_{vi} \leq c_v, \quad \text{for all } v \in V. \tag{4.1}$$

We refer to $\mathbf{x} = [x_{vi}]_{v \in V, i \in \mathcal{C}} \in \{0, 1\}^{|V||\mathcal{C}|}$ as the *global placement* or, simply, *placement* vector. We

[1] We associate queues with edges for concreteness. Alternatively, queues can be associated with nodes, or both nodes and edges; all such representations lead to product form distributions (4.7), and all our results extend to these cases.

denote by

$$\mathcal{D} = \left\{ \mathbf{x} \in \{0,1\}^{|V||\mathcal{C}|} : \textstyle\sum_{i \in \mathcal{C}} x_{vi} \leq c_v, \forall v \in V \right\}, \tag{4.2}$$

the set of *feasible* placements that satisfy the storage capacity constraints. We assume that for every object $i \in \mathcal{C}$, there exists a set of nodes $\mathcal{S}_i \subseteq V$ that *permanently store* i. We refer to nodes in $\mathcal{S}_i$ as *designated servers* for $i \in \mathcal{C}$. We assume that designated servers store i in permanent storage *outside* their cache. Put differently, the aggregate storage capacity of a node is $c'_v = c_v + |\{i : v \in \mathcal{S}_i\}|$, but only the non-designated slots c_v are part of the system's design.

Object Requests and Responses. Traffic in the cache network consists of two types of packets: *requests* and *responses*, as shown in Fig. 4.1(b). Requests for an object are always routed towards one of its designated servers, ensuring that every request is satisfied. However, requests may terminate early: upon reaching any node that caches the requested object, the latter generates a response carrying the object. This is forwarded towards the request's source, following the same path as the request, in reverse. Consistent with prior literature [7, 167], we treat request traffic as negligible when compared to response traffic, which carries objects, and henceforth focus only on queues bearing response traffic.

Formally, a request and its corresponding response are fully characterized by (a) the object being requested, and (b) the path that the request follows. That is, for the set of requests $\mathcal{R}$, a request $r \in \mathcal{R}$ is determined by a pair (i^r, p^r), where $i^r \in \mathcal{C}$ is the object being requested and p^r is the path the request follows. Each request r is associated with a corresponding Poisson arrival process with rate $\lambda^r \geq 0$, independent of other arrivals and service times. We denote the vector of arrival rates by $\boldsymbol{\lambda} = [\lambda^r]_{r \in \mathcal{R}} \in \mathbb{R}_+^{|\mathcal{R}|}$. For all $r \in \mathcal{R}$, we assume that the path p^r is well-routed [7], that is: (a) path p^r is simple, (b) the terminal node of the path is a designated server, i.e., a node in $\mathcal{S}_{i^r}$, and (c) no other intermediate node in p^r is a designated server. As a result, requests are always served, and response packets (carrying objects) always follow a sub-path of p^r in reverse towards the request source (namely, p_1^r).

Steady State Distribution. Given an object placement $\mathbf{x} \in \mathcal{D}$, the resulting system is a multi-class Kelly network, with packet classes determined by the request set $\mathcal{R}$. This is a Markov process over the state space determined by queue contents. In particular, let n_e^r be the number of packets of class $r \in \mathcal{R}$ in queue $e \in E$, and

$$n_e = \sum_{r \in \mathcal{R}} n_e^r \tag{4.3}$$

be the total queue size. The state of a queue $\mathbf{n}_e \in \mathcal{R}^{n_e}$, $e \in E$, is the vector of length n_e representing the class of each packet in each position of the queue. The *system state* is then given by $\mathbf{n} = [\mathbf{n}_e]_{e \in E}$; we denote by Ω the state space of this Markov process.

In contrast to classic Kelly networks, network traffic and, in particular, the load on each queue, depend on placement $\mathbf{x}$. Indeed, if $(v, u) \in p^r$ for $r \in \mathcal{R}$, the arrival rate of responses of class $r \in \mathcal{R}$ in queue $(u, v) \in E$ is:

$$\lambda^r_{(u,v)}(\mathbf{x}, \boldsymbol{\lambda}) = \lambda^r \prod_{k'=1}^{k_{p^r}(v)} (1 - x_{p^r_{k'} i^r}), \quad \text{for } (v, u) \in p^r, \tag{4.4}$$

i.e., responses to requests of class r pass through edge $(u, v) \in E$ if and only if no node preceding u in the path p^r stores object i^r–see also Fig. 4.1(b). Note that (4.4) presumes queues on p^r are stable. As $\mu_{(u,v)}$ is the service rate of the queue in $(u, v) \in E$, the load on edge $(u, v) \in E$ is:

$$\rho_{(u,v)}(\mathbf{x}, \boldsymbol{\lambda}) = \tfrac{\lambda_{(u,v)}(\mathbf{x}, \boldsymbol{\lambda})}{\mu_{(u,v)}}, \tag{4.5}$$

where

$$\lambda_{(u,v)}(\mathbf{x}, \boldsymbol{\lambda}) = \textstyle\sum_{r \in \mathcal{R} : (v,u) \in p^r} \lambda^r_{(u,v)}(\mathbf{x}, \boldsymbol{\lambda}) \tag{4.6}$$

is the total arrival rate of responses in queue $(u, v) \in E$. The Markov process $\{\mathbf{n}(t); t \geq 0\}$ is positive recurrent when $\rho_{(u,v)}(\mathbf{x}, \boldsymbol{\lambda}) < 1$, for all $(u, v) \in E$ [146, 168]. Then, the steady-state distribution has a *product form*, i.e.:

$$\pi(\mathbf{n}) = \textstyle\prod_{e \in E} \pi_e(\mathbf{n}_e), \quad \mathbf{n} \in \Omega, \tag{4.7}$$

where

$$\pi_e(\mathbf{n}_e) = (1 - \rho_e(\mathbf{x}, \boldsymbol{\lambda})) \textstyle\prod_{r \in \mathcal{R} : e \in p^r} \left(\frac{\lambda^r_e(\mathbf{x}, \boldsymbol{\lambda})}{\mu_e} \right)^{n^r_e}, \tag{4.8}$$

and $\lambda^r_e(\mathbf{x}, \boldsymbol{\lambda})$, $\rho_e(\mathbf{x}, \boldsymbol{\lambda})$ are given by (4.4), (4.5), respectively. As a consequence, the queue sizes n_e, $e \in E$, also have a product form distribution in steady state, and their marginals are given by:

$$\mathbf{P}(n_e = k) = (1 - \rho_e(\mathbf{x}, \boldsymbol{\lambda})) \rho^k_e(\mathbf{x}, \boldsymbol{\lambda}), \quad k \in \mathbb{N}. \tag{4.9}$$

Stability Region. Given a placement $\mathbf{x} \in \mathcal{D}$, a vector of arrival rates $\boldsymbol{\lambda} = [\lambda^r]_{r \in \mathcal{R}}$ yields a stable (i.e., positive recurrent) system if and only if $\boldsymbol{\lambda} \in \Lambda_{\mathbf{x}}$, for

$$\Lambda_{\mathbf{x}} := \{\boldsymbol{\lambda} : \boldsymbol{\lambda} \geq 0, \rho_e(\mathbf{x}, \boldsymbol{\lambda}) < 1, \forall e \in E\} \subset \mathbb{R}^{|\mathcal{R}|}_+, \tag{4.10}$$

where loads ρ_e, $e \in E$, are given by (4.5). Conversely, given a vector $\boldsymbol{\lambda} \in \mathbb{R}_+^{|\mathcal{R}|}$,

$$\mathcal{D}_{\boldsymbol{\lambda}} = \{\mathbf{x} \in \mathcal{D} : \rho_e(\mathbf{x}, \boldsymbol{\lambda}) < 1, \forall e \in E\} \subseteq \mathcal{D} \tag{4.11}$$

is the set of feasible placements under which the system is stable. It is easy to confirm that, by the monotonicity of ρ_e w.r.t. $\mathbf{x}$, if $\mathbf{x} \in \mathcal{D}_{\boldsymbol{\lambda}}$ and $\mathbf{x}' \geq \mathbf{x}$, then $\mathbf{x}' \in \mathcal{D}_{\boldsymbol{\lambda}}$, where the vector inequality $\mathbf{x}' \geq \mathbf{x}$ is component-wise. In particular, if $\mathbf{0} \in D_{\boldsymbol{\lambda}}$ (i.e., the system is stable without caching), then $\mathcal{D}_{\boldsymbol{\lambda}} = \mathcal{D}$.

4.1.2 Cache Optimization

Our approach is closest to, and inspired by, recent work by Shanmugam et al. [157] and Ioannidis and Yeh [7]. Ioannidis and Yeh consider a setting very similar to ours but without queuing: edges are assigned a fixed weight, and the objective is a linear function of incoming traffic scaled by these weights. This can be seen as a special case of our model, namely, one where edge costs are linear (see also Eq. (4.16) in Sec. 4.1.2). Shanmugam et al. [157] study a similar optimization problem, restricted to the context of femtocaching. The authors show that this is an NP-hard, submodular maximization problem with matroid constraints. They provide a $1 - 1/e$ approximation algorithm based on a technique by Ageev and Sviridenko [166]: this involves maximizing a concave relaxation of the original objective, and rounding via pipage-rounding[166]. Ioannidis and Yeh show that the same approximation technique applies to more general cache networks with linear edge costs. They also provide a distributed, adaptive algorithm that attains an $1 - 1/e$ approximation. The same authors extend this framework to jointly optimize both caching and routing decisions [167].

Given a Kelly cache network represented by graph $G(V, E)$, service rates μ_e, $e \in E$, storage capacities c_v, $v \in V$, a set of requests $\mathcal{R}$, and arrival rates λ_r, for $r \in \mathcal{R}$, we wish to determine placements $\mathbf{x} \in \mathcal{D}$ that optimize a certain design objective. In particular, we seek placements that are solutions to optimization problems of the following form:

MINCOST

$$\text{Minimize:} \quad C(\mathbf{x}) = \textstyle\sum_{e \in E} C_e(\rho_e(\mathbf{x}, \boldsymbol{\lambda})), \tag{4.12a}$$

$$\text{subj. to:} \quad \mathbf{x} \in \mathcal{D}_{\boldsymbol{\lambda}}, \tag{4.12b}$$

where $C_e : [0, 1) \to \mathbb{R}_+$, $e \in E$, are positive *cost* functions, $\rho_e : \mathcal{D} \times \mathbb{R}_+^{|\mathcal{R}|} \to \mathbb{R}_+$ is the load on edge e, given by (4.5), and $\mathcal{D}_{\boldsymbol{\lambda}}$ is the set of feasible placements that ensure stability, given by (4.11).

We make the following standing assumption on the cost functions appearing in MINCOST:

Assumption 1. *For all $e \in E$, functions $C_e : [0,1) \to \mathbb{R}_+$ are convex and non-decreasing on $[0,1)$.*

Assumption 1 is natural; indeed it holds for many cost functions that often arise in practice. We list several examples:

Example 1. Queue Size: Under steady-state distribution (4.7), the expected number of packets in queue $e \in E$ is given by

$$\mathbb{E}[n_e] = C_e(\rho_e) = \frac{\rho_e}{1-\rho_e}, \tag{4.13}$$

which is indeed convex and non-decreasing for $\rho_e \in [0,1)$. Note that, for C_e given by (4.13), objective (4.12a) captures the expected total number of packets in the system in steady state.

Example 2. Delay: From Little's Theorem [168], the expected delay experienced by a packet in the system is

$$\mathbb{E}[T] = \frac{1}{\|\boldsymbol{\lambda}\|_1} \sum_{e \in E} \mathbb{E}[n_e], \tag{4.14}$$

where $\|\boldsymbol{\lambda}\|_1 = \sum_{r \in \mathcal{R}} \lambda^r$ is the total arrival rate, and $\mathbb{E}[n_e]$ is the expected size of each queue. Thus, the expected delay can also be written as the sum of functions that satisfy Assumption 1. We note that the same is true for the sum of the expected delays per queue $e \in E$, as the latter are given by

$$\mathbb{E}[T_e] = \frac{1}{\lambda_e}\mathbb{E}[n_e] = \frac{1}{\mu_e(1-\rho_e)}, \tag{4.15}$$

which are also convex and non-decreasing in ρ_e.

Example 3. Queuing Probability/Load per Edge: In a FIFO queue, the *queuing probability* is the probability of arriving in a system where the server is busy; by (4.9), this is:

$$C_e(\rho_e) = \rho_e = \lambda_e/\mu_e, \tag{4.16}$$

which is again non-decreasing and convex. This is also, of course, the load per edge. By treating $1/\mu_e$ as the weight of edge $e \in E$, this setting recovers the objectives of Shanmugam et al. [157] and Ioannidis and Yeh [7] as a special case of our model.

Example 4. Monotone Separable Costs: More generally, Assumption 1 holds for *arbitrary monotone separable costs*, i.e., costs that (1) are summed across queues, (2) depend only on queue sizes n_e, and (3) are non-decreasing. Formally:

Lemma 4.1.1. *Consider a state-dependent cost function $c : \Omega \to \mathbb{R}_+$ such that:*

$$c(\mathbf{n}) = \sum_{e \in E} c_e(n_e),$$

where $c_e : \mathbb{N} \to \mathbb{R}_+$, $e \in E$, *are non-decreasing functions of the queue sizes* n_e, $e \in E$. *Then, the steady state cost under distribution* (4.7) *takes the form* (4.12a)*, i.e.,*

$$\mathbb{E}[c(\mathbf{n})] = \sum_{e \in E} C_e(\rho_e)$$

where $C_e : [0, 1) \to \mathbb{R}_+$ *satisfy Assumption 1.*

Proof. As the cost at state $\mathbf{n} \in \Omega$ can be written as $c(\mathbf{n}) = \sum_{e \in E} c_e(n_e)$, we have that $\mathbb{E}[c(\mathbf{n})] = \sum_{e \in E} \mathbb{E}[c_e(n_e)]$. On the other hand, as $c_e(n_e) \geq 0$,

$$\begin{aligned} \mathbb{E}[c_e(n_e)] &= \sum_{n=0}^{\infty} c_e(n)\mathbf{P}(n_e = n) \\ &= c_e(0) + \sum_{n=0}^{\infty}(c_e(n+1) - c_e(n))\mathbf{P}(n_e > n) \\ &\overset{(4.9)}{=} c_e(0) + \sum_{n=0}^{\infty}(c_e(n+1) - c_e(n))\rho_e^n \end{aligned} \tag{4.17}$$

As c_e is non-decreasing, $c_e(n+1) - c_e(n) \geq 0$ for all $n \in \mathbb{N}$. On the other hand, for all $n \in \mathbb{N}$, ρ^n is a convex non-decreasing function of ρ in $[0, 1)$, so $\mathbb{E}[c_e(n_e)]$ is a convex function of ρ_e as a positively weighted sum of convex non-decreasing functions. □

In summary, MINCOST captures many natural cost objectives, while Assumption 1 holds for *any* non-decreasing cost function that depends only on queue sizes.

4.2 Submodularity of Cache Optimization and Algorithms

Our work can be seen as an extension of [7, 157], in that it incorporates queuing in the cache network. In contrast to both [7] and [157] however, costs like delay or queue sizes are highly non-linear in the presence of queuing. From a technical standpoint, this departure from linearity requires us to employ significantly different optimization methods than the ones in [7, 157]. In particular, our objective *does not admit a concave relaxation* and, consequently, the technique by Ageev and Sviridenko [166] used in [7, 157] *does not apply*. In fact, Problem MINCOST is NP-hard; this is true even when cost functions c_e are linear, and the objective is to minimize the sum of the loads per edge [7, 157]. In what follows, we outline our methodology for solving this problem; it relies on the fact that the objective of MINCOST is a *supermodular set function*; our first main contribution is to show that this property is a direct consequence of Assumption 1.

Cost Supermodularity and Caching Gain. First, observe that the cost function C in MINCOST can be naturally expressed as a set function. Indeed, for $S \subset V \times \mathcal{C}$, let $\mathbf{x}_S \in \{0,1\}^{|V||\mathcal{C}|}$ be the binary vector whose support is S (i.e., its non-zero elements are indexed by S). As there is a 1-1 correspondence between a binary vector $\mathbf{x}$ and its support $\text{supp}(\mathbf{x})$, we can interpret $C : \{0,1\}^{|V||\mathcal{C}|} \to \mathbb{R}_+$ as a set function $C : V \times \mathcal{C} \to \mathbb{R}_+$ via $C(S) \triangleq C(\mathbf{x}_S)$. Then, the following theorem holds:

Theorem 4.2.1. *Under Assumption 1, $C(S) \triangleq C(\mathbf{x}_S)$ is non-increasing and supermodular over* $\{\text{supp}(\mathbf{x}) : \mathbf{x} \in \mathcal{D}_{\boldsymbol{\lambda}}\}$.

Proof. We use the following auxiliary lemma (see, e.g., [91]); we prove this here for completeness.

Lemma 4.2.2. *Let $f : \mathbb{R} \to \mathbb{R}$ be a convex and non-decreasing function. Also, let $g : \mathcal{X} \to \mathbb{R}$ be a non-increasing supermodular set function. Then $h \triangleq f \circ g$ is also supermodular.*

Proof. Since g is non-increasing, for any $\mathbf{x}, \mathbf{x}' \subseteq \mathcal{X}$ we have $g(\mathbf{x} \cap \mathbf{x}') \geq g(\mathbf{x}) \geq g(\mathbf{x} \cup \mathbf{x}')$, and $g(\mathbf{x} \cap \mathbf{x}') \geq g(\mathbf{x}') \geq g(\mathbf{x} \cup \mathbf{x}')$. Due to supermodularity of g, we can find $\alpha, \alpha' \in [0,1]$, $\alpha + \alpha' \geq 1$ such that $g(\mathbf{x}) = (1-\alpha) g(\mathbf{x} \cap \mathbf{x}') + \alpha g(\mathbf{x} \cup \mathbf{x}')$, and $g(\mathbf{x}') = (1-\alpha') g(\mathbf{x} \cap \mathbf{x}') + \alpha' g(\mathbf{x} \cup \mathbf{x}')$. Then, we have

$$\begin{aligned} f(g(\mathbf{x})) + f(g(\mathbf{x}')) &\leq (1-\alpha) f(g(\mathbf{x} \cap \mathbf{x}')) + \alpha f(g(\mathbf{x} \cup \mathbf{x}')) \\ &\quad + (1-\alpha') f(g(\mathbf{x} \cap \mathbf{x}')) + \alpha' f(g(\mathbf{x} \cup \mathbf{x}')) \\ &= f(g(\mathbf{x} \cap \mathbf{x}')) + f(g(\mathbf{x} \cup \mathbf{x}')) \\ &\quad + (1-\alpha-\alpha')(f(g(\mathbf{x} \cap \mathbf{x}')) - f(g(\mathbf{x} \cup \mathbf{x}'))) \\ &\leq f(g(\mathbf{x} \cap \mathbf{x}')) + f(g(\mathbf{x} \cup \mathbf{x}')), \end{aligned}$$

where the first inequality is due to convexity of f, and the second one is because $\alpha + \alpha' \geq 1$ and $f(g(.))$ is non-increasing. This proves $h(\mathbf{x}) \triangleq f(g(\mathbf{x}))$ is supermodular. □

To conclude the proof of Thm. 4.2.1, observe that it is easy to verify that $\rho_e, \forall e \in E$, is supermodular and non-increasing in S (see also [7]). Since, by Assumption 1, C_e is a non-decreasing function, then, $C_e(S) \triangleq C_e(\rho_{u,v}(S))$ is non-increasing. By Lemma 4.2.2, $C_e(S)$ is also supermodular. Hence, the cost function is non-increasing and supermodular as the sum of non-increasing and supermodular functions. □

In light of Lemma 4.1.1, Thm. 4.2.1 implies that supermodularity arises *for a broad array of natural cost objectives*, including expected delay and system size; it also applies under

the *full generality of Kelly networks* under which a product form arises, including FIFO, LIFO, and round robin service disciplines. Armed with this theorem, we turn our attention to converting MINCOST to a *submodular maximization* problem. In doing so, we face the problem that the domain $\mathcal{D}_{\boldsymbol{\lambda}}$, determined not only by storage capacity constraints, but also by stability, may be difficult to characterize. Nevertheless, we show that a problem that is amenable to approximation can be constructed, provided that a placement $\mathbf{x}_0 \in \mathcal{D}_{\boldsymbol{\lambda}}$ is known.

In particular, suppose that we have access to a single $\mathbf{x}_0 \in \mathcal{D}_{\boldsymbol{\lambda}}$. We define the *caching gain* $F : \mathcal{D}_{\boldsymbol{\lambda}} \to \mathbb{R}_+$ as $F(\mathbf{x}) = C(\mathbf{x}_0) - C(\mathbf{x})$. Note that, for $\mathbf{x} \geq \mathbf{x}_0$, $F(\mathbf{x})$ is the relative decrease in the cost compared to the cost under $\mathbf{x}_0$. We consider the following optimization problem:

MAXCG

$$\text{Maximize:} \quad F(\mathbf{x}) = C(\mathbf{x}_0) - C(\mathbf{x}) \tag{4.18a}$$

$$\text{subj. to:} \quad \mathbf{x} \in \mathcal{D}, \mathbf{x} \geq \mathbf{x}_0 \tag{4.18b}$$

Recall that, if $\mathbf{0} \in \mathcal{D}_{\boldsymbol{\lambda}}$, then $\mathcal{D}_{\boldsymbol{\lambda}} = \mathcal{D}$; in this case, taking $\mathbf{x}_0 = \mathbf{0}$ ensures that problems MINCOST and MaxCG are equivalent. If $\mathbf{x}_0 \neq \mathbf{0}$, the above formulation attempts to maximize the gain restricted to placements $\mathbf{x} \in \mathcal{D}$ that dominate $\mathbf{x}_0$: such placements necessarily satisfy $\mathbf{x} \in \mathcal{D}_{\boldsymbol{\lambda}}$, as $\mathbf{x}_0 \in \mathcal{D}_{\boldsymbol{\lambda}}$. Thm. 4.2.1 has the following immediate implication:

Corollary 4.2.2.1. *The caching gain $F(S) \triangleq F(\mathbf{x}_S)$ is non-decreasing and submodular over $\{\text{supp}(\mathbf{x}) : \mathbf{x} \in \mathcal{D}_{\boldsymbol{\lambda}}\}$.*

Greedy Algorithm. Constraints (4.18b) define a (partition) matroid [40, 157]. This, along with the submodularity and monotonicity of F imply that we can produce a solution within $\frac{1}{2}$-approximation from the optimal via the *greedy* algorithm [39]. The algorithm, summarized in Alg. 6, iteratively allocates items to caches that yield the largest marginal gain. The solution produced by Algorithm 6 is guaranteed to be within a $\frac{1}{2}$-approximation ratio of the optimal solution of MAXCG [38]. The approximation guarantee of $\frac{1}{2}$ is tight:

Lemma 4.2.3. *For any $\varepsilon > 0$, there exists a cache network such that the greedy algorithm solution is within $\frac{1}{2} + \varepsilon$ from the optimal, when the objective is the sum of expected delays per edge.*

Proof. Consider the path topology illustrated in Fig. 4.2. Assume that requests for files 1 and 2 are generated at node u with rates $\lambda_1 = \lambda_2 = \delta$, for some $\delta \in (0, 1)$. Files 1 and 2 are stored permanently at v and z, respectively. Caches exist only on u and w, and have capacity $c_u = c_w = 1$. Edges (u, v), (w, z) have bandwidth $\mu_{(u,v)} = \mu_{(w,z)} = 1$, while edge (u, w) is a high bandwidth link,

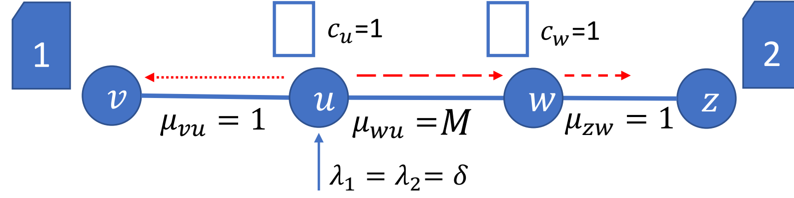


Figure 4.2: A path graph, illustrating that the $1/2$-approximation ratio of greedy is tight. Greedy caches item 2 in node u, while the optimal decision is to cache item 1 in u and item 2 in node w. For M large enough, the approximation ratio can be made arbitrarily close to 1/2. In our experiments in Sec. 4.4, we set $\delta = 0.5$ and $M = 200$.

Algorithm 6 Greedy

Require: $F : \mathcal{D} \to \mathbb{R}_+, \mathbf{x}_0$

1: $\mathbf{x} \leftarrow \mathbf{x}_0$
2: **while** $A(\mathbf{x}) := \{(v,i) \in V \times \mathcal{C} : \mathbf{x} + \mathbf{e}_{vi} \in \mathcal{D}\}$ is not empty **do**
3: $\quad (v^*, i^*) \leftarrow \arg\max_{(v,i)\in A(\mathbf{x})} (F(\mathbf{x} + \mathbf{e}_{vi}) - F(\mathbf{x}))$
4: $\quad \mathbf{x} \leftarrow \mathbf{x} + \mathbf{e}_{v^*i^*}$
5: **end while**
6: **return** $\mathbf{x}$

having capacity $M \gg 1$. Let $\mathbf{x}_0 = \mathbf{0}$. The greedy algorithm starts from empty caches and adds item 2 at cache u. This is because the caching gain from this placement is $c_{(u,w)} + c_{(w,z)} = \frac{1}{M-\delta} + \frac{1}{1-\delta}$, while the caching gain of all other decisions is at most $\frac{1}{1-\delta}$. Any subsequent caching decisions do not change the caching gain. The optimal solution is to cache item 1 at u and item 2 at w, yielding a caching gain of $2/(1-\delta)$. Hence, the greedy solution attains an approximation ratio $0.5 \cdot (1 + \frac{1-\delta}{M-\delta})$. By appropriately choosing M and δ, this can be made arbitrarily close to 0.5. □

As we discuss in Sec. 4.4, the greedy algorithm performs well in practice for some topologies; however, Lemma 4.2.3 motivates us to seek alternative algorithms, that attain improved approximation guarantees. We note that it is easy to extend Lemma 4.2.3 to other objectives, including, e.g., expected delay, queue size, etc. We note also that tight instances can be constructed using caches with capacities larger than 1 (see, e.g., Fig. 4.4).

4.2.1 Continuous-Greedy Algorithm

The *continuous-greedy* algorithm by Calinescu et al. [40] attains a tighter guarantee than the greedy algorithm, raising the approximation ratio from 0.5 to $1 - 1/e \approx 0.63$. The algorithm maximizes the so-called *multilinear extension* of objective F, thereby obtaining a fractional solution

Algorithm 7 Continuous-Greedy

Require: $G : \tilde{\mathcal{D}} \to \mathbb{R}_+$, $\mathbf{x}_0$, stepsize $0 < \gamma \leq 1$
1: $t \leftarrow 0, k \leftarrow 0$ $\mathbf{y}_0 \leftarrow \mathbf{x}_0$
2: **while** $t < 1$ **do**
3: $\quad\mathbf{m}_k \leftarrow \arg\max_{\mathbf{m} \in \tilde{\mathcal{D}}} \langle \mathbf{m}, \nabla G(\mathbf{y}_k) \rangle$
4: $\quad\gamma_k \leftarrow \min\{\gamma, 1 - t\}$
5: $\quad\mathbf{y}_{k+1} \leftarrow \mathbf{y}_k + \gamma_k \mathbf{m}_k, t \leftarrow t + \gamma_k, k \leftarrow k + 1$
6: **end while**
7: **return** $\mathbf{y}_k$

Y in the convex hull of the constraint space. The resulting solution is then rounded to produce an integral solution. The algorithm requires estimating the gradient of the multilinear extension via sampling; interestingly, we prove that MAXCG exhibits additional structure, which can be used to construct a polynomial-time estimator of this gradient that *eschews sampling altogether*, by using a Taylor expansion.

Algorithm Overview. Formally, the multilinear extension of the caching gain F is defined as follows. Define the convex hull of the set defined by the constraints (4.18b) in MAXCG as:

$$\tilde{\mathcal{D}} = \texttt{conv}(\{\mathbf{x} : \mathbf{x} \in \mathcal{D}, \mathbf{x} \geq \mathbf{x}_0\}) \subseteq [0, 1]^{|V||\mathcal{C}|} \tag{4.19}$$

Intuitively, $\mathbf{y} \in \tilde{\mathcal{D}}$ is a *fractional* vector in $\mathbb{R}^{|V||\mathcal{C}|}$ satisfying the capacity constraints, and the bound $\mathbf{y} \geq \mathbf{x}_0$.

Given a $\mathbf{y} \in \tilde{\mathcal{D}}$, consider a random vector $\mathbf{x}$ in $\{0, 1\}^{|V||\mathcal{C}|}$ generated as follows: for all $v \in V$ and $i \in \mathcal{C}$, the coordinates $x_{vi} \in \{0, 1\}$ are independent Bernoulli variables such that $\mathbf{P}(x_{vi} = 1) = y_{vi}$. The multilinear extension $G : \tilde{\mathcal{D}} \to \mathbb{R}_+$ of $F : \mathcal{D}_{\boldsymbol{\lambda}} \to \mathbb{R}_+$ is defined via expectation

$$G(\mathbf{y}) = \mathbb{E}_{\mathbf{y}}[F(\mathbf{x})], \tag{4.20}$$

parameterized by $\mathbf{y} \in \tilde{\mathcal{D}}$. That is:

$$G(\mathbf{y}) = \sum_{\mathbf{x} \in \{0,1\}^{|V||\mathcal{C}|}} F(\mathbf{x}) \times \prod_{(v,i) \in V \times \mathcal{C}} y_{vi}^{x_{vi}} (1 - y_{vi})^{1 - x_{vi}}, \tag{4.21}$$

The continuous-greedy algorithm, summarized in Alg. 7, proceeds by first producing a fractional vector $\mathbf{y} \in \tilde{\mathcal{D}}$. Starting from $\mathbf{y}_0 = \mathbf{x}_0$, the algorithm iterates over:

$$\mathbf{m}_k \in \arg\max_{\mathbf{m} \in \tilde{\mathcal{D}}} \langle \mathbf{m}, \nabla G(\mathbf{y}_k) \rangle, \tag{4.22a}$$

$$\mathbf{y}_{k+1} = \mathbf{y}_k + \gamma_k \mathbf{m}_k, \tag{4.22b}$$

for an appropriately selected step size $\gamma_k \in [0,1]$. Intuitively, this yields an approximate solution to the non-convex problem:

$$\text{Maximize:} \quad G(\mathbf{y}) \tag{4.23a}$$

$$\text{subj. to:} \quad \mathbf{y} \in \tilde{\mathcal{D}}. \tag{4.23b}$$

Even though (4.23) is not convex, the output of Alg. 7 is within a $1-1/e$ factor from the optimal solution $\mathbf{y}^* \in \tilde{\mathcal{D}}$ of (4.23). This fractional solution can be rounded to produce a solution to MAXCG with the same approximation guarantee using either the pipage rounding [166] or the swap rounding [40, 169] schemes. Note that the maximization in (4.22a) is a Linear Program (LP): it involves maximizing a linear objective subject to a set of linear constraints, and can thus be computed in polynomial time. However, this presumes access to the gradient ∇G. On the other hand, the expectation $G(\mathbf{y}) = \mathbb{E}_{\mathbf{y}}[F(\mathbf{x})]$ alone, given by (4.21), involves a summation over $2^{|V||\mathcal{C}|}$ terms, and it may not be easily computed in polynomial time. To address this, the customary approach is to first generate random samples $\mathbf{x}$ and then use these to produce an unbiased estimate of the gradient (see, e.g., [40]); this estimate can be used in Alg. 7 instead of the gradient. Before presenting our estimator tailored to MAXCG, we first describe this sampling-based estimator.

A Sampling-Based Estimator. Function G is linear when restricted to each coordinate y_{vi}, for some $v \in V$, $i \in \mathcal{C}$ (i.e., when all inputs except y_{vi} are fixed). As a result, the partial derivative of G w.r.t. y_{vi} can be written as:

$$\frac{\partial G(\mathbf{y})}{\partial y_{vi}} = \mathbb{E}_{\mathbf{y}}[F(\mathbf{x})|x_{vi}=1] - \mathbb{E}_{\mathbf{y}}[F(\mathbf{x})|x_{vi}=0] \geq 0, \tag{4.24}$$

where the last inequality is due to monotonicity of F. One can thus estimate the gradient by (a) producing T random samples $\mathbf{x}^{(\ell)}$, $\ell = 1, \ldots, T$ of the random vector $\mathbf{x}$, consisting of independent Bernoulli coordinates, and (b) computing, for each pair $(v,i) \in V \times \mathcal{C}$, the average

$$\widehat{\frac{\partial G(\mathbf{y})}{\partial y_{vi}}} = \frac{1}{T}\sum_{\ell=1}^{T}\left(F([\mathbf{x}^\ell]_{+(v,i)}) - F([\mathbf{x}^\ell]_{-(v,i)})\right), \tag{4.25}$$

where $[\mathbf{x}]_{+(v,i)}$,$[\mathbf{x}]_{-(v,i)}$ are equal to vector $\mathbf{x}$ with the (v,i)-th coordinate set to 1 and 0, respectively. Using this estimate, Alg. 7 is poly-time and attains an approximation ratio arbitrarily close to $1-1/e$ for appropriately chosen T. In particular, the following theorem holds:

Theorem 4.2.4. *[Calinescu et al. [40]] Consider Alg. 7, with $\nabla G(\mathbf{y}_k)$ replaced by the sampling-based estimate $\widehat{\nabla G(\mathbf{y}^k)}$, given by* (4.25)*. Set $T = \frac{10}{\delta^2}(1+\ln(|\mathcal{C}||V|))$, and $\gamma = \delta$, where $\delta = \frac{1}{40|\mathcal{C}||V|\cdot(\sum_{v\in V} c_v)^2}$. Then, the algorithm terminates after $K = 1/\gamma = 1/\delta$ steps and, with high probability,*

$$G(\mathbf{y}^K) \geq (1-(1-\delta)^{1/\delta})G(\mathbf{y}^*) \geq (1-1/e)G(\mathbf{y}^*),$$

where $\mathbf{y}^*$ *is an optimal solution to* (4.23).

The proof of the theorem can be found in Appendix A of Calinescu et al. [40] for general submodular functions over arbitrary matroid constraints; we state Thm. 4.2.4 here with constants T and γ set specifically for our objective G and our set of constraints $\tilde{D}$.

Complexity. Under this parameterization of T and γ, Alg. 7 runs in polynomial time. More specifically, note that $1/\delta = O(|\mathcal{C}||V| \cdot (\sum_{v\in V} c_v)^2)$ is polynomial in the input size. Moreover, the algorithm runs for $K = 1/\delta$ iterations in total. Each iteration requires $T = O(\frac{1}{\delta^2}(1 + \ln(|\mathcal{C}||V|)$ samples, each involving a polynomial computation (as F can be evaluated in polynomial time). LP (4.22a) can be solved in polynomial time in the number of constraints and variables, which are $O(|V||\mathcal{C}|)$. Finally, the rounding schemes require at most $O(|V||\mathcal{C}|)$ steps.

4.2.2 A Novel Estimator via Taylor Expansion

The classic approach to estimate the gradient via sampling has certain drawbacks. The number of samples T required to attain the $1 - 1/e$ ratio is quadratic in $|V||\mathcal{C}|$. In practice, even for networks and catalogs of moderate size (say, $|V| = |\mathcal{C}| = 100$), the number of samples becomes prohibitive (of the order of 10^8). Producing an estimate for ∇G via a closed form computation that eschews sampling thus has significant computational advantages. In this section, we show that the multilinear relaxation of the caching gain F admits such a closed-form characterization.

We say that a polynomial $f : \mathbb{R}^d \to \mathbb{R}$ is in *Weighted Disjunctive Normal Form* (*W-DNF*) if it can be written as

$$f(\mathbf{x}) = \textstyle\sum_{s\in\mathcal{S}} \beta_s \cdot \prod_{j\in\mathcal{I}(s)}(1 - x_j), \tag{4.26}$$

for some index set $\mathcal{S}$, positive coefficients $\beta_s > 0$, and index sets $I(s) \subseteq \{1, \ldots, d\}$. Intuitively, treating binary variables $x_j \in \{0, 1\}$ as Boolean values, each W-DNF polynomial can be seen as a weighted sum (disjunction) among products (conjunctions) of negative literals. These polynomials arise naturally in the context of our problem; in particular:

Lemma 4.2.5. *For all* $k \geq 1$, $\mathbf{x} \in \mathcal{D}$, *and* $e \in E$, $\rho_e^k(\mathbf{x}, \boldsymbol{\lambda})$ *is a W-DNF polynomial whose coefficients depend on* $\boldsymbol{\lambda}$.

Proof. We prove this by induction on $k \geq 1$. Observe first that, by (4.5), the load on each edge $e = (u, v) \in E$ can be written as a polynomial of the following form:

$$\rho_e(\mathbf{x}, \boldsymbol{\lambda}) = \sum_{r\in\mathcal{R}_e} \beta_r(\boldsymbol{\lambda}) \cdot \prod_{j\in\mathcal{I}_e(r)} (1 - x_j), \tag{4.27}$$

for appropriately defined

$$\mathcal{R}_e = \mathcal{R}_{(u,v)} = \{r \in \mathcal{R} : (v,u) \in p^r\}, \tag{4.28a}$$

$$\mathcal{I}_e(r) = \{(w, i^r) \in V \times \mathcal{C} : w \in p^r, k_{p^r}(w) \leq k_{p^r}(v)\}, \tag{4.28b}$$

$$\beta_r(\boldsymbol{\lambda}) = \lambda^r/\mu_e. \tag{4.28c}$$

In other words, $\rho_e : \mathcal{D}_{\boldsymbol{\lambda}} \to \mathbb{R}$ is indeed a *W-DNF* polynomial. For the induction step, observe that W-DNF polynomials, seen as functions over the integral domain $\mathcal{D}_{\boldsymbol{\lambda}}$, are *closed under multiplication*. In particular, the following lemma holds:

Lemma 4.2.6. *Given two W-DNF polynomials $f_1 : \mathcal{D}_{\boldsymbol{\lambda}} \to \mathbb{R}$ and $f_2 : \mathcal{D}_{\boldsymbol{\lambda}} \to \mathbb{R}$, given by*

$$f_1(\mathbf{x}) = \sum_{r \in \mathcal{R}_1} \beta_r \prod_{t \in \mathcal{I}_1(r)} (1 - x_t), \quad \textit{and}$$

$$f_2(\mathbf{x}) = \sum_{r \in \mathcal{R}_2} \beta_r \prod_{t \in \mathcal{I}_2(r)} (1 - x_t),$$

their product $f_1 \cdot f_2$ is also a W-DNF polynomial over $\mathcal{D}_{\boldsymbol{\lambda}}$, given by:

$$(f_1 \cdot f_2)(\mathbf{x}) = \sum_{(r,r') \in \mathcal{R}_1 \times \mathcal{R}_2} \beta_r \beta'_r \prod_{t \in \mathcal{I}_1(r) \cup \mathcal{I}_2(r')} (1 - x_t)$$

Proof. To see this, observe that

$$f_1(\mathbf{x}) f_1(\mathbf{x}) = \sum_{(r,r') \in \mathcal{R}_1 \times \mathcal{R}_2} \beta_r \beta'_r \prod_{t \in \mathcal{I}_1(r) \cap \mathcal{I}_2(r')} (1 - x_t)^2 \prod_{t \in \mathcal{I}_1(r) \triangle \mathcal{I}_2(r')} (1 - x_t)$$

where $\triangle$ is the symmetric set difference. On the other hand, as $(1 - x_t) \in \{0, 1\}$, we have that $(1 - x_t)^2 = (1 - x_t)$, and the lemma follows. □

Hence, if $\rho_e^k(\mathbf{x}, \boldsymbol{\lambda})$ is a W-DNF polynomial, by (4.27) and Lemma 4.2.6, so is $\rho_e^{k+1}(\mathbf{x}, \boldsymbol{\lambda})$. □

Hence, *all load powers are W-DNF polynomials.* Expectations of W-DNF polynomials have a remarkable property:

Lemma 4.2.7. *Let $f : \mathcal{D}_{\boldsymbol{\lambda}} \to \mathbb{R}$ be a W-DNF polynomial, and let $\mathbf{x} \in \mathcal{D}$ be a random vector of independent Bernoulli coordinates parameterized by $\mathbf{y} \in \tilde{\mathcal{D}}$. Then $\mathbb{E}_{\mathbf{y}}[f(\mathbf{x})] = f(\mathbf{y})$, where $f(\mathbf{y})$ is the evaluation of the W-DNF polynomial representing f over the real vector $\mathbf{y}$.*

Proof. As f is W-DNF, it can be written as

$$f(\mathbf{x}) = \sum_{s\in\mathcal{S}} \beta_s \prod_{t\in\mathcal{I}(s)} (1 - x_t)$$

for appropriate $\mathcal{S}$, and appropriate $\beta_s, \mathcal{I}(s)$, where $s \in \mathcal{S}$. Hence,

$$\begin{aligned}\mathbb{E}_{\mathbf{y}}[f(\mathbf{x})] &= \sum_{s\in\mathcal{S}} \beta_s \mathbb{E}_{\mathbf{y}}\left[\prod_{t\in\mathcal{I}(s)} (1 - x_t)\right] \\ &= \sum_{s\in\mathcal{S}} \beta_s \prod_{t\in\mathcal{I}(s)} (1 - \mathbb{E}_{\mathbf{y}}[x_t]), \text{ by independence} \\ &= \sum_{s\in\mathcal{S}} \beta_s \prod_{t\in\mathcal{I}(s)} (1 - y_t).\end{aligned}$$

□

Lemma 4.2.7 states that, to compute the expectation of a W-DNF polynomial f over i.i.d. Bernoulli variables with expectations $\mathbf{y}$, it suffices to *evaluate f over input* $\mathbf{y}$. Expectations computed this way therefore do not require sampling.

We leverage this property to approximate $\nabla G(\boldsymbol{y})$ by taking the Taylor expansion of the cost functions C_e at each edge $e \in E$. This allows us to write C_e as a power series w.r.t. ρ_e^k, $k \geq 1$; from Lemmas 4.2.5 and 4.2.7, we can compute the expectation of this series in a closed form. In particular, by expanding the series and rearranging terms it is easy to show the following lemma:

Lemma 4.2.8. *Consider a cost function $C_e : [0, 1) \to \mathbb{R}_+$ which satisfies Assumption 1 and for which the Taylor expansion exists at some $\rho^* \in [0, 1)$. Then, for $\mathbf{x} \in \mathcal{D}$ a random Bernoulli vector parameterized by $\mathbf{y} \in \tilde{\mathcal{D}}$,*

$$\frac{\partial G(\mathbf{y})}{\partial y_{vi}} \approx \sum_{e\in E}\sum_{k=1}^{L} \alpha_e^{(k)} \Big[\rho_e^k\left([\mathbf{y}]_{-(v,i)}, \boldsymbol{\lambda}\right) - \rho_e^k\left([\mathbf{y}]_{+(v,i)}, \boldsymbol{\lambda}\right)\Big] \tag{4.29}$$

where, $\alpha_e^{(k)} = \sum_{j=k}^{L} \frac{(-1)^{j-k}\binom{j}{k}}{j!} C_e^{(j)}(\rho^*)(\rho^*)^{j-k}$, *for* $k = 0, 1, \cdots, L$, *and the error of the approximation is:* $\frac{1}{(L+1)!}\sum_{e\in E} C_e^{(L+1)}(\rho')\Big[\mathbb{E}_{[\mathbf{y}]_{-(v,i)}}[(\rho_e(\mathbf{x}, \boldsymbol{\lambda}) - \rho^*)^{L+1}] - \mathbb{E}_{[\mathbf{y}]_{+(v,i)}}[(\rho_e(\mathbf{x}, \boldsymbol{\lambda}) - \rho^*)^{L+1}]\Big]$*,where* $\rho' \in [\rho^*, \rho]$.

Proof. The Taylor expansion of C_e at ρ^* is given by:

$$C_e(\rho) = C_e(\rho^*) + \sum_{k=1}^{L} \frac{1}{k!} C_e^{(k)}(\rho^*)(\rho - \rho^*)^k + \\ + \frac{1}{(L+1)!} C_e^{(L+1)}(\rho')(\rho - \rho^*)^{L+1},$$

where $\rho' \in [\rho^*, \rho]$ and $C_e^{(k)}$ is the k-th order derivative of C_e. By expanding this polynomial and reorganizing the terms, we get

$$C_e(\rho) = \sum_{k=0}^{L} \alpha_e^{(k)} \rho^k + \frac{1}{(L+1)!} C_e^{(L+1)}(\rho')(\rho - \rho^*)^{L+1},$$

where

$$\alpha_e^{(k)} = \textstyle\sum_{j=k}^{L} \frac{(-1)^{j-k}\binom{j}{k}}{j!} C_e^{(j)}(\rho^*)(\rho^*)^{j-k},$$

for $k = 0, 1, \cdots, L$. Consider now the L-th order Taylor approximation of C_e, given by

$$\hat{C}_e(\rho) = \sum_{k=0}^{L} \alpha_e^{(k)} \rho^k.$$

Clearly, this is an estimator of C_e, with an error of the order $|C_e(\rho) - \hat{C}_e(\rho)| = o\left((\rho - \rho_*)^L\right)$. Thus, for $\mathbf{x} \in \mathcal{D}$ a random Bernoulli vector parameterized by $\mathbf{y} \in \tilde{\mathcal{D}}$,

$$\mathbb{E}_{\mathbf{y}}[C_e(\rho_e(\mathbf{x}, \boldsymbol{\lambda}))] \approx \mathbb{E}_{\mathbf{y}}[\hat{C}_e(\rho_e(\mathbf{x}, \boldsymbol{\lambda}))] = \sum_{k=0}^{L} \alpha_e^{(k)} \mathbb{E}_{\mathbf{y}}[\rho_e^k(\mathbf{x}, \boldsymbol{\lambda})] \tag{4.30}$$

On the other hand, for all $v \in V$ and $i \in \mathcal{C}$:

$$\begin{aligned}
\frac{\partial G(\mathbf{y})}{\partial y_{vi}} &\overset{(4.24)}{=} \mathbb{E}_{\mathbf{y}}[F(\mathbf{x})|x_{vi} = 1] - \mathbb{E}_{\mathbf{y}}[F(\mathbf{x})|x_{vi} = 0] \\
&\overset{(4.18a)}{=} \mathbb{E}_{\mathbf{y}}[C(\mathbf{x})|x_{vi} = 0] - \mathbb{E}_{\mathbf{y}}[C(\mathbf{x})|x_{vi} = 1] \\
&\overset{(4.12a),(4.30)}{\approx} \sum_{e \in E} \sum_{k=1}^{L} \alpha_e^{(k)} \Big(\mathbb{E}_{\mathbf{y}}[\rho_e^k(\mathbf{x}, \boldsymbol{\lambda})|x_{vi} = 0] \\
&\qquad - \mathbb{E}_{\mathbf{y}}[\rho_e^k(\mathbf{x}, \boldsymbol{\lambda})|x_{vi} = 1]\Big),
\end{aligned} \tag{4.31}$$

where the error of the approximation is given by

$$\begin{aligned}
&\frac{1}{(L+1)!} \sum_{e \in E} C_e^{(L+1)}(\rho') \Big[\mathbb{E}_{\mathbf{y}}[(\rho_e(\mathbf{x}, \boldsymbol{\lambda}) - \rho^*)^{L+1}|x_{vi} = 0] \\
&\quad - \mathbb{E}_{\mathbf{y}}[(\rho_e(\mathbf{x}, \boldsymbol{\lambda}) - \rho^*)^{L+1}|x_{vi} = 1]\Big]
\end{aligned}$$

The lemma thus follows from Lemmas 4.2.5 and 4.2.7. □

Estimator (4.29) is *deterministic*: no random sampling is required. Moreover, Taylor's theorem allows us to characterize the error (i.e., the *bias*) of this estimate. We use this to characterize the final fractional solution $\mathbf{y}$ produced by Alg. 7:

Theorem 4.2.9. *Assume that all C_e, $e \in E$, satisfy Assumption 1, are $L+1$-differentiable, and that all their $L+1$ derivatives are bounded by $W \geq 0$. Then, consider Alg. 7, in which $\nabla G(\mathbf{y}_k)$ is estimated via the Taylor estimator* (4.29)*, where each edge cost function is approximated at $\rho_e^* = \mathbb{E}_{\mathbf{y}_k}[\rho_e(\mathbf{x}, \boldsymbol{\lambda})] = \rho_e(\mathbf{y}_k, \boldsymbol{\lambda})$. Then,*

$$G(\mathbf{y}_K) \geq (1 - \tfrac{1}{e})G(\mathbf{y}^*) - 2DB - \tfrac{P}{2K}, \tag{4.32}$$

where $K = \frac{1}{\gamma}$ is the number of iterations, $\mathbf{y}^*$ is an optimal solution to (4.23)*, $D = \max_{\mathbf{y} \in \tilde{\mathcal{D}}} \|\mathbf{y}\|_2 \leq |V| \cdot \max_{v \in V} c_v$, is the diameter of $\tilde{\mathcal{D}}$, $B \leq \frac{W|E|}{(L+1)!}$ is the bias of the estimator* (4.29)*, and $P = 2C(\mathbf{x}_0)$, is a Lipschitz constant of ∇G.*

Proof. We begin by bounding the bias of estimator (4.31). Indeed, given a set of continuous functions $\{C_{(u,v)}\}_{(u,v) \in E}$ where their first $L+1$ derivatives within their operating regime, $[0, 1)$, are upperbounded by a finite constant, W, the bias of estimator $\mathbf{z} \equiv [z_{vi}]_{v \in V, i \in \mathcal{C}}$, where z_{vi} is defined by (4.29), is given by

$$\begin{aligned} B &\equiv ||\mathbf{z} \ - \nabla G(\mathbf{y})||_2 \\ &= || \sum_{e \in E} \frac{1}{(L+1)!} C_e^{(L+1)}(\rho_e')(\rho_e - \rho_e^*)^{L+1} ||_2, \end{aligned} \tag{4.33}$$

where $\rho_e' \in [\rho_e^*, \rho_e]$. To compute the bias, we note that $\rho_e, \rho_e^* \in [0, 1]$. Specifically, we assume $\rho_e, \rho_e^* \in [0, 1)$. Hence, $|\rho_e - \rho_e^*| \leq 1$, and $C_e^{(L+1)}(\rho_e') \leq \max\{C_e^{(L+1)}(\rho_e), C_e^{(L+1)}(\rho_e^*)\} < \infty$. In particular, let $W = \max_{e \in E} C_e^{(L+1)}(\rho_e')$. Then, it is easy to compute the following upper bound on the bias of $\mathbf{z}$:

$$B \leq \frac{W|E|}{(L+1)!}. \tag{4.34}$$

In addition, note that G is linear in y_{vi}, and hence [40]:

$$\begin{aligned} \frac{\partial G}{\partial y_{vi}} &= \mathbb{E}[F(\mathbf{x})|x_{vi} = 1] - \mathbb{E}[F(\mathbf{x})|x_{vi} = 0] \\ &= \mathbb{E}[C(\mathbf{x})|x_{vi} = 0] - \mathbb{E}[C(\mathbf{x})|x_{vi} = 1] \geq 0, \end{aligned} \tag{4.35}$$

which is ≥ 0 due to monotonicity of $F(\mathbf{x})$. It is easy to verify that $\frac{\partial^2 G}{\partial y_{vi}^2} = 0$. For $(v_1, i_1) \neq (v_2, i_2)$, we can compute the second derivative of G [40] as given by

$$\begin{aligned} \frac{\partial^2 G}{\partial y_{v_1 i_1} \partial y_{v_2 i_2}} &= \mathbb{E}[C(\mathbf{x})|x_{v_1 i_1} = 1, x_{v_2 i_2} = 0] \\ &+ \mathbb{E}[C(\mathbf{x})|x_{v_1 i_1} = 0, x_{v_2 i_2} = 1] \\ &- \mathbb{E}[C(\mathbf{x})|x_{v_1 i_1} = 1, x_{v_2 i_2} = 1] \\ &- \mathbb{E}[C(\mathbf{x})|x_{v_1 i_1} = 0, x_{v_2 i_2} = 0] \leq 0, \end{aligned}$$

which is ≤ 0 due to the supermodularity of $C(\mathbf{x})$. Hence, $G(\mathbf{y})$ is component-wise concave [40] .

In addition, it is easy to see that for $\mathbf{y} \in \tilde{\mathcal{D}}$, $|G(\mathbf{y})|$, $||\nabla G(\mathbf{y})||$, and $||\nabla^2 G(\mathbf{y})||$ are bounded by $C(\mathbf{x}_0)$, $C(\mathbf{x}_0)|\mathcal{C}||V|$, and $2C(\mathbf{x}_0)(|\mathcal{C}||V|)^2$, respectively. Consequently, ∇G is P-Lipschitz continuous, with $P = 2C(\mathbf{x}_0)(|\mathcal{C}||V|)^2$.

In the kth iteration of the Continuous Greedy algorithm, let $\mathbf{m}^* = \mathbf{m}^*(\mathbf{y}_k) := (\mathbf{y}^* \vee (\mathbf{y}_k + \mathbf{y}_0)) - \mathbf{y}_k = (\mathbf{y}^* - \mathbf{y}_k) \vee \mathbf{y}_0 \geq \mathbf{y}_0$, where $x \vee y := (\max\{x_i, y_i\})_i$. Since $\mathbf{m}^* \leq \mathbf{y}^*$ and $\mathcal{D}$ is closed-down, $\mathbf{m}^* \in \mathcal{D}$. Due to monotonicity of G, it follows

$$G(\mathbf{y}_k + \mathbf{m}^*) \geq G(\mathbf{y}^*). \tag{4.36}$$

We introduce univariate auxiliary function $g_{\mathbf{y},\mathbf{m}}(\xi) := G(\mathbf{y} + \xi\mathbf{m}), \xi \in [0,1], \mathbf{m} \in \tilde{\mathcal{D}}$. Since $G(\mathbf{y})$ is component-wise concave, then, $g_{\mathbf{y},\mathbf{m}}(\xi)$ is concave in $[0,1]$. In addition, since $g_{\mathbf{y}_k,\mathbf{m}^*}(\xi) = G(\mathbf{y}_k + \xi\mathbf{m}^*)$ is concave for $\xi \in [0,1]$, it follows

$$\begin{aligned} g_{\mathbf{y}_k,\mathbf{m}^*}(1) - g_{\mathbf{y}_k,\mathbf{m}^*}(0) &= G(\mathbf{y}_k + \mathbf{m}^*) - G(\mathbf{y}_k) \\ &\leq \frac{dg_{\mathbf{y}_k,\mathbf{m}}(0)}{d\xi} \times 1 = \langle \mathbf{m}^*, \nabla G(\mathbf{y}_k) \rangle. \end{aligned} \tag{4.37}$$

Now let $\mathbf{m}_k$ be the vector chosen by Algorithm 7 in the kth iteration. We have

$$\langle \mathbf{m}_k, \mathbf{z}(\mathbf{y}_k) \rangle \geq \langle \mathbf{m}^*, \mathbf{z}(\mathbf{y}_k) \rangle. \tag{4.38}$$

For the LHS, we have

$$\begin{aligned} \langle \mathbf{m}_k, \mathbf{z} \rangle &= \langle \mathbf{m}_k, \nabla G(\mathbf{y}_k) \rangle + \langle \mathbf{m}_k, \mathbf{z} - \nabla G(\mathbf{y}_k) \rangle \\ &\overset{(i)}{\leq} \langle \mathbf{m}_k, \nabla G(\mathbf{y}_k) \rangle + ||m_k||_2 \cdot |\mathbf{z} - \nabla G(\mathbf{y}_k)||2 \\ &\leq \langle \mathbf{m}_k, \nabla G(\mathbf{y}_k) \rangle + DB. \end{aligned} \tag{4.39}$$

where $D = \max_{\mathbf{m} \in \tilde{\mathcal{D}}} \|\mathbf{m}\|_2 \leq |V| \cdot \max_{v \in V} c_v$, is the upperbound on the diameter of $\tilde{\mathcal{D}}$, B is as defined in (4.34), and (i) follows from Cauchy-Schwarz inequality. Similarly, we have for the RHS of that (4.38)

$$\langle \mathbf{m}^*, \mathbf{z}(\mathbf{y}_k) \rangle \geq \langle \mathbf{m}^*, \nabla G(\mathbf{y}_k) \rangle - DB. \tag{4.40}$$

It follows

$$\begin{aligned} \langle \mathbf{m}_k, \nabla G(\mathbf{y}_k) \rangle + 2DB &\geq \langle \mathbf{m}^*, \nabla G(\mathbf{y}_k) \rangle \\ &\overset{(a)}{\geq} G(\mathbf{y}_k + \mathbf{m}^*) - G(\mathbf{y}_k) \overset{(b)}{\geq} G(\mathbf{y}^*) - G(\mathbf{y}_k), \end{aligned} \tag{4.41}$$

where (a) follows from (4.37), and (b) follows from (4.36).

Using the P-Lipschitz continuity property of $\frac{dg_{\mathbf{y}_k,\mathbf{m}_k}(\xi)}{d\xi}$ (due to P-Lipschitz continuity of $\triangledown G$), it is straightforward to see that

$$\begin{aligned}-\frac{P\gamma_k^2}{2} &\leq g_{\mathbf{y}_k,\mathbf{m}_k}(\gamma_k) - g_{\mathbf{y}_k,\mathbf{m}_k}(0) - \gamma_k \cdot \frac{dg_{\mathbf{y}_k,\mathbf{m}_k}(0)}{d\xi} \\ &= G(\mathbf{y}_k + \gamma_k \mathbf{m}_k) - G(\mathbf{y}_k) - \gamma_k < \mathbf{m}_k, \triangledown G(\mathbf{y}_k) >, \end{aligned} \tag{4.42}$$

hence,

$$G(\mathbf{y}_{k+1}) - G(\mathbf{y}_k) \geq \gamma_k \langle \mathbf{m}_k, \triangledown G(\mathbf{y}_k)\rangle - \frac{P\gamma_k^2}{2} \tag{4.43}$$

$$\begin{aligned}&\geq \gamma_k \langle \mathbf{m}_k, \triangledown G(\mathbf{y}_k)\rangle - \frac{P\gamma_k^2}{2} \\ &\overset{(c)}{\geq} \gamma_k (G(\mathbf{y}^*) - G(\mathbf{y}_k)) - 2\gamma_k DB - \frac{P\gamma_k^2}{2},\end{aligned} \tag{4.44}$$

where (c) follows from (4.41), respectively. By rearranging the terms and letting $k = K-1$, we have

$$\begin{aligned}G(\mathbf{y}_K) - G(\mathbf{y}^*) &\geq \prod_{j=0}^{K-1}(1-\gamma_j)(G(\mathbf{y}_0) - G(\mathbf{y}^*)) - 2DB\sum_{j=0}^{K-1}\gamma_j - \frac{P}{2}\sum_{j=0}^{K-1}\gamma_j^2 \\ &\overset{(e)}{\geq} (G(\mathbf{y}_0) - G(\mathbf{y}^*))\exp\{-\sum_{j=0}^{K-1}\gamma_j\} - 2DB\sum_{j=0}^{K-1}\gamma_j - \frac{P}{2}\sum_{j=0}^{K-1}\gamma_j^2,\end{aligned}$$

where (e) is true since $1 - x \leq e^{-x}, \forall x \geq 0$, and $G(\mathbf{y}_0) \leq G(\mathbf{y}^*)$ holds due to the greedy nature of Algorithm 7 and monotonicity of G. In addition, Algorithm 7 ensures $\sum_{j=0}^{K-1}\gamma_j = 1$. It follows

$$G(\mathbf{y}_K) - (1 - \frac{1}{e})G(\mathbf{y}^*) \geq e^{-1}G(\mathbf{y}_0) - 2DB - \frac{P}{2}\sum_{j=0}^{K-1}\gamma_j^2. \tag{4.45}$$

This result holds for general stepsizes $0 < \gamma_j \leq 1$. The RHS of (4.45) is indeed maximized when $\gamma_j = \frac{1}{K}$, which is the assumed case in Algorithm 7. In addition, we have $\mathbf{y}_0 = \mathbf{0}$, and hence, $G(\mathbf{y}_0) = 0$. Therefore, we have

$$G(\mathbf{y}_K) - (1 - \frac{1}{e})G(\mathbf{y}^*) \geq -2DB - \frac{P}{2K}. \tag{4.46}$$

□

The theorem immediately implies that we can replace (4.29) as an estimator in Alg. 7, and attain an approximation arbitrarily close to $1 - 1/e$. Note that the computational complexity of the estimator depends on the number of terms in the W-DNF form; this, in turn, depends on L. In practice, as shown in Section 4.4, this estimator significantly outperforms sampling in both execution time and caching gain attained.

Estimation via Power Series. For arbitrary $L+1$-differentiable cost functions C_e, the estimator (4.29) can be leveraged by replacing C_e with its Taylor expansion. In the case of queue-dependent cost functions, as described in Example 4 of Section 4.1.2, the power-series (4.17) can be used instead. For example, the expected queue size (Example 1, Sec. 4.1.2), is given by $C_e(\rho_e) = \frac{\rho_e}{1-\rho_e} = \sum_{k=1}^{\infty} \rho_e^k$. In contrast to the Taylor expansion, this power series does not depend on a point ρ_e^* around which the function C_e is approximated.

4.2.3 The Role of Sparsity in Efficient Computations

The crux of our proposed method is the gradient estimator (4.35), which depends on computations of products and summations of W-DNF functions (see Lemmas 4.2.5 and 4.2.8). Note that the load over each edge ρ_e in (4.27), is a summation over only a subset of requests (defined in (4.28a)). Moreover, for each request the load depends only on a subset of variables (4.28b). In other words, the support of each load function ρ_e is *sparse*, as it only depends on a subset of variables, i.e., $\cup_{r\in\mathcal{R}_e}\mathcal{I}_e(r)$. Note that the size of each set $\mathcal{I}_e(r)$ directly depends on the length of the path p^r; in practice, every path only contains a subset of edges in the graph.

Moreover, the support for the k-th powers of these load functions, is the union over the sets $\cup_{i\in[k]}\mathcal{I}_e(r_i)$ for all sets $r_i \in \mathcal{R}_e$, due to Lemma 4.2.6. Therefore, the supports of the load functions are increased with a factor of k. Given that in practice taking powers of 2 is sufficient (see our experiments in Sec. 4.4), we see that the powers of ρ_e also have sparse supports. Therefore, the gradient estimations in (4.29) can be computed efficiently.

4.3 Beyond M/M/1 queues

As discussed in Section 4.1.1, the classes of M/M/1 queues for which the supermodularity of the cost functions arises are quite broad, and include FIFO, LIFO, and processor sharing queues. In this section, we discuss how our results extend to even broader families of queuing networks. Chapter 3 of Kelly [146] provides a general framework for a set of queues for which service times

are exponentially distributed. A large class of networks can be modeled by this framework, including networks of M/M/k queues; all such networks maintain the property that steady-state distributions have a product form. This allows us to extend our results to M/M/k queues for two cost functions C_e:

Lemma 4.3.1. *For a network of M/M/k queues, both the queuing probability*[2] *and the expected queue size are non-increasing and supermodular over sets* $\{\text{supp}(\mathbf{x}) : \mathbf{x} \in \mathcal{D}_{\boldsymbol{\lambda}}\}$.

Proof. For an arbitrary network of M/M/k queues, the traffic load on queue $(u, v) \in E$ is given as

$$a_{(u,v)}(\mathbf{x}) = \frac{\sum_{r\in\mathcal{R}:(v,u)\in p^r} \lambda^r \prod_{k'=1}^{k_{p^r}(v)} (1 - x_{p^r_{k'} i^r})}{k\mu_{(u,v)}}, \tag{4.47}$$

which is similar to that of M/M/1 queues, but normalized by the number of servers, k. Hence, $a_{(u,v)}(\mathbf{x})$ is submodular in $\mathbf{x}$. For an M/M/k queue, the probability that an arriving packet finds all servers busy and will be forced to wait in queue is given by Erlang C formula [168], which follows

$$P^Q_{(u,v)}(\mathbf{x}) = \frac{b_{(u,v)}(\mathbf{x})(ka_{(u,v)}(\mathbf{x}))^k}{k!(1 - a_{(u,v)}(\mathbf{x}))}, \tag{4.48}$$

where

$$b_{(u,v)}(\mathbf{x}) = \left[\sum_{n=0}^{k-1} \frac{(ka_{(u,v)}(\mathbf{x}))^n}{n!} + \frac{(ka_{(u,v)}(\mathbf{x}))^k}{k!(1 - a_{(u,v)}(\mathbf{x}))}\right]^{-1}, \tag{4.49}$$

is the normalizing factor. In addition, the expected number of packets waiting for or under transmission is given by

$$\mathbb{E}[n_{(u,v)}(\mathbf{x})] = ka_{(u,v)}(\mathbf{x}) + \frac{a_{(u,v)}(\mathbf{x})P^Q_{(u,v)}(\mathbf{x})}{1 - a_{(u,v)}(\mathbf{x})}. \tag{4.50}$$

Lee and Cohen [170] show that $P^Q_{(u,v)}(\mathbf{x})$ and $\mathbb{E}[n_{(u,v)}(\mathbf{x})]$ are strictly increasing and convex in $a_{(u,v)}(\mathbf{x})$, for $a_{(u,v)}(\mathbf{x}) \in [0, 1)$. In addition, a more direct proof of convexity of $\mathbb{E}[n_{(u,v)}(\mathbf{x})]$ was shown by Grassmann in [171]. Hence, Both $P(\mathbf{x}) := \sum_{(u,v)\in E} P^Q_{(u,v)}(\mathbf{x})$ and $N(\mathbf{x}) := \sum_{(u,v)\in E} \mathbb{E}[n_{(u,v)}(\mathbf{x})]$ are increasing and convex. Due to Theorem 4.2.1, we note that both functions are non-increasing and supermodular in $\mathbf{x}$, and the proof is complete. □

Product-form steady-state distributions arise also in settings where service times are not exponentially distributed. A large class of quasi-reversible queues, named *symmetric queues* exhibit this property (c.f. Section 3.3 of [146] and Chapter 10 of [148]). In the following lemma we leverage the product form of symmetric queues to extend our results to M/D/1 symmetric queues [168]:

[2] This is given by the so-called Erlang C formula [168].

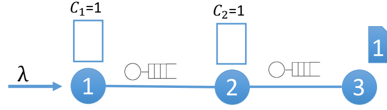


Figure 4.3: A simple network with finite-capacity queues.

Table 4.1: Results of $\rho_{u,v}(\mathbf{x})$'s for different caching configurations.

$[x_{11}, x_{21}]$	$\rho_{3,2}$	$\rho_{2,1}$
$[0,0]$	$\frac{\lambda}{\mu_{3,2}}$	$\frac{\lambda(1-p^L_{3,2})}{\mu_{2,1}}$
$[1,0]$	0	0
$[0,1]$	0	$\frac{\lambda}{\mu_{2,1}}$
$[1,1]$	0	0

Lemma 4.3.2. *For a network of M/D/1 symmetric queues, the expected queue size is non-increasing and supermodular over sets* $\{\text{supp}(\mathbf{x}) : \mathbf{x} \in \mathcal{D}_{\boldsymbol{\lambda}}\}$.

Proof. Let $\rho_{(u,v)}(\mathbf{x})$ be the traffic load on queue $(u,v) \in E$, as defined by (4.5). It can be shown that the average number of packets in queue $(u,v) \in E$ is of form [168]

$$\mathbb{E}[n_{(u,v)}(\mathbf{x})] = \rho_{(u,v)}(\mathbf{x}) + \frac{\rho^2_{(u,v)}(\mathbf{x})}{2(1-\rho_{(u,v)}(\mathbf{x}))}. \tag{4.51}$$

It is easy to see that this function is strictly increasing and convex in $\rho_{(u,v)}(\mathbf{x})$ for $\rho_{(u,v)}(\mathbf{x}) \in [0,1)$. Due to Theorem 4.2.1, $N(\mathbf{x}) := \sum_{(u,v)\in E} \mathbb{E}[n_{(u,v)}(\mathbf{x})]$ is non-increasing and supermodular in $\mathbf{x}$, and the proof is complete. □

Again, Lemma 4.3.2 and Little's theorem imply that this property also extends to network delays. It is worth noting that conclusions similar to these in Lemmas 4.3.1 and 4.3.2 are not possible for all general queues with product form distributions. In particular, we prove the following negative result:

Lemma 4.3.3. *There exists a network of M/M/1/k queues, containing a queue e, for which no strictly monotone function C_e of the load ρ_e at a queue e is non-increasing and supermodular over sets* $\{\text{supp}(\mathbf{x}) : \mathbf{x} \in \mathcal{D}_{\boldsymbol{\lambda}}\}$. *In particular, the expected size of queue e is neither monotone nor supermodular.*

Proof. Consider the network of $M/M/1/k$ queues in Fig. 4.3, where node 1 is requesting content 1 from node 3, according to a Poisson process with rate λ. For simplicity, we only consider the traffic for content 1. For queues $(2,1)$ and $(3,2)$, it is easy to verify that the probability of packet drop at queues $(u,v) \in \{(2,1),(3,2)\}$ is given by

$$p^L_{(u,v)}(\rho_{(u,v)}) = \frac{\rho_{u,v}(\mathbf{x})^k(1-\rho_{(u,v)}(\mathbf{x}))}{1-\rho_{(u,v)}(\mathbf{x})^{k+1}}, \tag{4.52}$$

where $\rho_{(u,v)}(\mathbf{x})$ is the traffic load on queue (u,v), and it can be computed for$(2,1)$ and $(3,2)$ as follows:

$$\rho_{(2,1)}(x_{11},x_{21}) = \frac{\lambda(1-x_{11})(1-p^L_{(3,2)})}{\mu_{(2,1)}}, \tag{4.53}$$

$$\rho_{(3,2)}(x_{11},x_{21}) = \frac{\lambda(1-x_{11})(1-x_{21})}{\mu_{(3,2)}}. \tag{4.54}$$

Using the results reported in Table 4.1, it is easy to verify that ρ's are not monotone in $\mathbf{x}$. Hence, no strictly monotone function of ρ's are monotone in $\mathbf{x}$. In addition, it can be verified that ρ's are neither submodular, nor supermodular in $\mathbf{x}$. To show this, let sets $A=\emptyset$, and $B=\{(1,1)\}$, correspond to caching configurations $[0,0]$ and $[1,0]$, respectively. Note that $A \subset B$, and $(2,1) \notin B$. Since $\rho_{(3,2)}(A\cup\{(2,1)\}) - \rho_{(3,2)}(A) = -\frac{\lambda}{\mu_{(3,2)}} \ngeq 0 = \rho_{(3,2)}(B\cup\{(2,1)\}) - \rho_{(3,2)}(B)$, then $\rho_{(3,2)}$ is not submodular. Consequently, no strictly monotone function of $\rho_{(3,2)}$ is submodular. Similarly, as $\rho_{(2,1)}(A\cup\{(2,1)\}) - \rho_{(2,1)}(A) = \frac{\lambda p^L_{(3,2)}}{\mu_{(2,1)}} \nleq 0 = \rho_{(2,1)}(B\cup\{(2,1)\}) - \rho_{(2,1)}(B)$, $\rho_{(2,1)}$ is not supermodular. Thus, no strictly monotone function of $\rho_{(2,1)}$ is supermodular. □

4.4 Experiments

Networks. We execute Algorithms 6 and 7 over 9 network topologies, summarized in Table 4.2. Graphs `ER` and `ER-20Q` are the same 100-node Erdős-Rényi graph with parameter $p = 0.1$. Graphs `HC` and `HC-20Q` are the same hypercube graph with 128 nodes, and graph `star` is a star graph with 100 nodes. The graph `path` is the topology shown in Fig. 4.2. The last 3 topologies, namely, `dtelekom`, `geant`, and `abilene` represent the Deutsche Telekom, GEANT, and Abilene backbone networks, respectively. The latter is also shown in Fig. 4.4.

Experimental Setup. For `path` and `abilene`, we set demands, storage capacities, and service rates as illustrated in Figures 4.2 and 4.4, respectively. Both of these settings induce an approximation ratio close to $1/2$ for greedy. For all remaining topologies, we consider a catalog of size $|\mathcal{C}|$ objects; for each object, we select 1 node uniformly at random (u.a.r.) from V to serve as the designated

Table 4.2: Graph Topologies and Experiment Parameters.

Graph	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert\mathcal{C}\rvert$	$\lvert\mathcal{R}\rvert$	$\lvert Q\rvert$	c_v	$F_{\text{PL}}(\mathbf{x}_{\text{RND}})$	$F_{\text{UNI}}(\mathbf{x}_{\text{RND}})$
ER	100	1042	300	1K	4	3	2.75	2.98
ER-20Q	100	1042	300	1K	20	3	3.1	2.88
HC	128	896	300	1K	4	3	2.25	5.23
HC-20Q	128	896	300	1K	20	3	2.52	5.99
star	100	198	300	1K	4	3	6.08	8.3
path	4	3	2	2	1	1	1.2	1.2
dtelekom	68	546	300	1K	4	3	2.57	3.66
abilene	11	28	4	4	2	1/2	4.39	4.39
geant	22	66	10	40	4	2	19.68	17.22

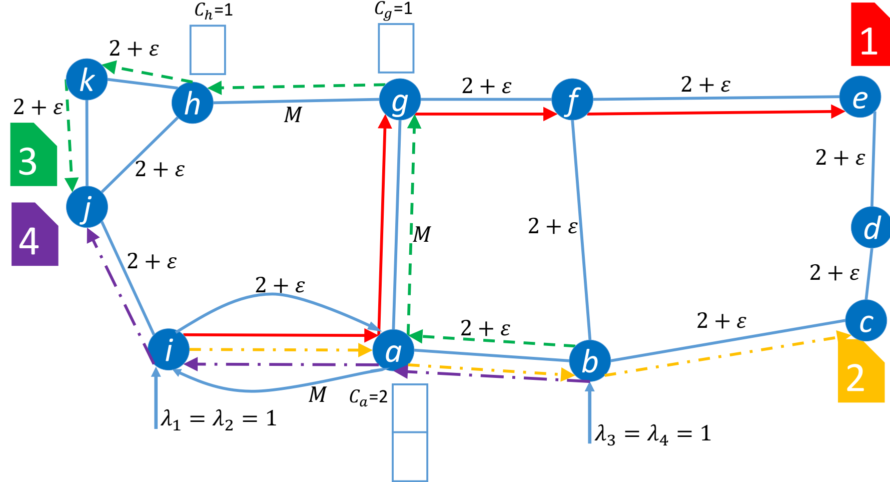
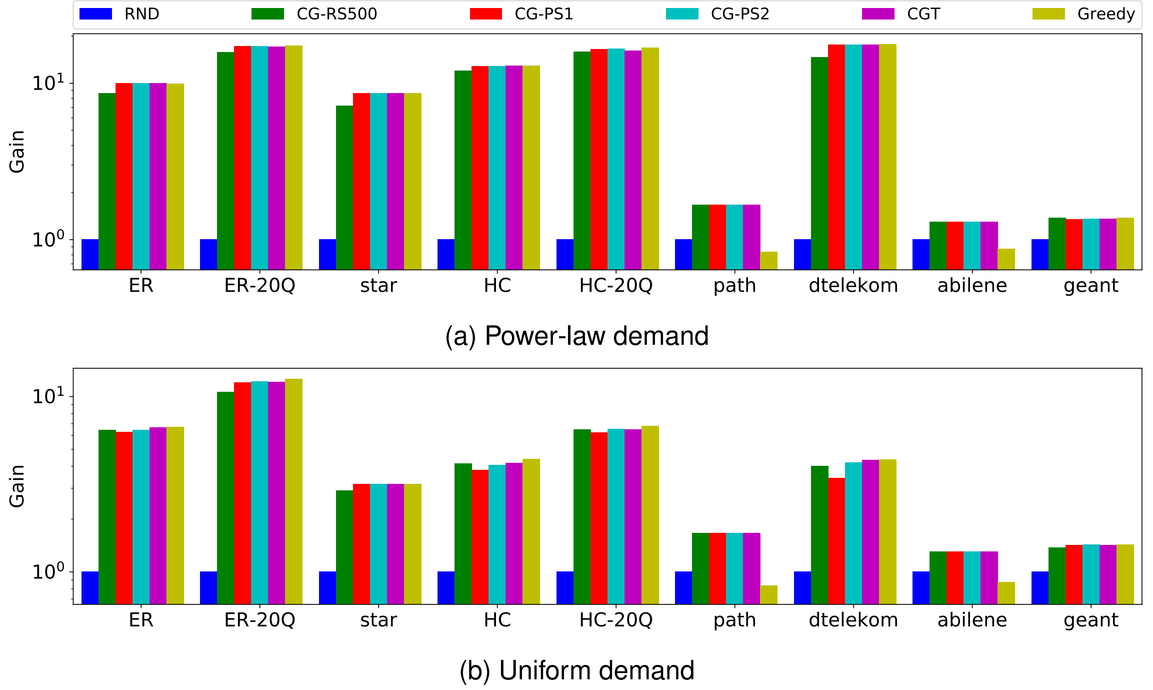


Figure 4.4: The abilene topology. We consider a catalog size of $|\mathcal{C}| = 4$ and 4 requests ($|\mathcal{R}| = 4$). Requests originate from $|Q| = 2$ nodes, b and i. Three edges have a high service rate $M \gg 1$, and the rest have a low service rate $2 + \varepsilon$. Only nodes a, g, and h can cache items, and have capacities 2, 1, and 1, respectively. We set $M = 200$ and $\varepsilon = 0.05$ in our experiments. Greedy is 0.5-approximate in this instance.

server for this object. To induce traffic overlaps, we also select $|Q|$ nodes u.a.r. that serve as sources for requests; all requests originate from these sources. All caches are set to the same storage capacity, i.e., $c_v = c$ for all $v \in V$.

We generate a set of $|\mathcal{R}|$ possible types of requests. For each request type $r \in \mathcal{R}$, $\lambda^r = 1$ request per second, and path p^r is generated by selecting a source among the $|Q|$ sources u.a.r., and routing towards the designated server of object i^r using a shortest path algorithm. We consider two ways of selecting objects $i^r \in \mathcal{C}$: in the *uniform* regime, i^r is selected u.a.r. from the catalog $\mathcal{C}$; in



(a) Power-law demand

(b) Uniform demand

Figure 4.5: Caching gains for different topologies and different arrival distributions, normalized by the gains corresponding to RND, reported in Table. 4.2. Greedy performs comparatively well. However, it attains sub-optimal solutions for `path` and `abilene`; these solutions are worse than RND. CG-RS500 has a poor performance compared to other variations of the continuous-greedy algorithm.

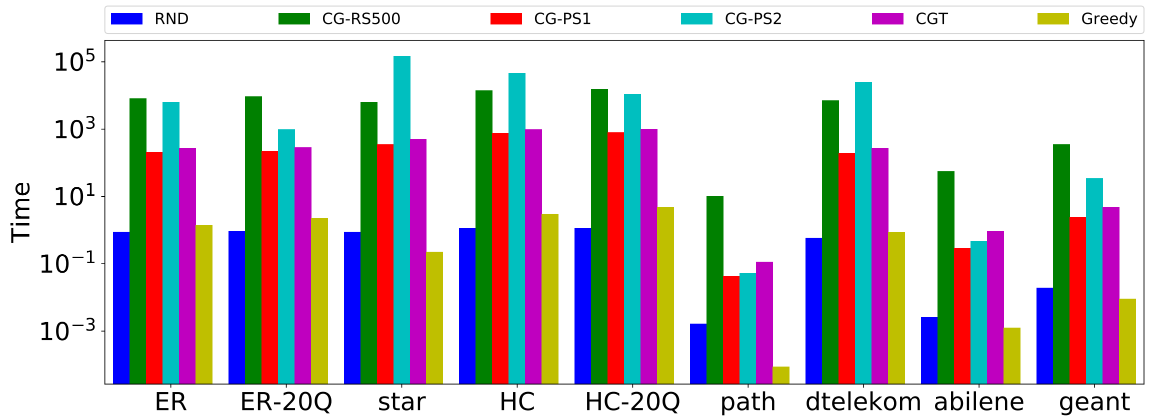


Figure 4.6: Running time for different topologies and power-law arrival distribution, in seconds. CG-RS500 is slower than power series estimation CG-PS1 and CGT, sometimes exceeding CG-PS2 as well.

the *power-law* regime, i^r is selected from the catalog $\mathcal{C}$ via a power law distribution with exponent 1.2. All the parameter values, e.g., catalog size $|\mathcal{C}|$, number of requests $|\mathcal{R}|$, number of query sources

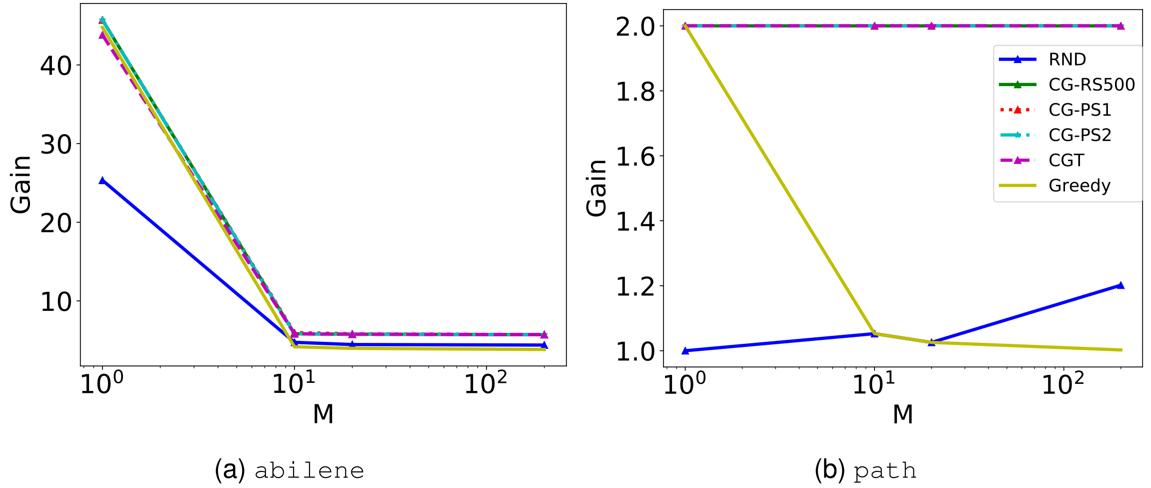
$|Q|$, and caching capacities c_v are presented in Table 4.2. We evaluate the caching gain $F(x)$ with queue size as the cost function, as defined in (4.13).

We construct heterogeneous service rates as follows. Every queue service rate is either set to a low value $\mu_e = \mu_{\text{low}}$ or a high value $\mu_e = \mu_{\text{high}}$, for all $e \in E$. We select μ_{low} and μ_{high} as follows. Given the demands $r \in \mathcal{R}$ and the corresponding arrival rates λ^r, we compute the highest load under no caching ($\mathbf{x} = \mathbf{0}$), i.e., we find $\lambda_{\max} = \max_{e \in E} \sum_{r: e \in p^r} \lambda^r$. We then set $\mu_{\text{low}} = \lambda_{\max} \times 1.05$ and $\mu_{\text{high}} = \lambda_{\max} \times 200$. We set the service rate to μ_{low} for all congested edges, i.e., edges e s.t. $\lambda_e = \lambda_{\max}$. We set the service rate for each remaining edge $e \in E$ to μ_{low} independently with probability 0.7, and to μ_{high} otherwise. Note that, as a result $\mathbf{0} \in \mathcal{D}_{\boldsymbol{\lambda}} = \mathcal{D}$, i.e., the system is stable even in the absence of caching and, on average, 30 percent of the edges have a high service rate.

Placement Algorithms. We implement several placement algorithms: (a) *Greedy*, i.e., the greedy algorithm (Alg. 6), (b) *Continuous-Greedy with Random Sampling* (CG-RS), i.e., Algorithm 7 with a gradient estimator based on sampling, as described in Sec. 4.2.1, (c) *Continuous-Greedy with Taylor approximation* (CGT), i.e., Algorithm 7 with a gradient estimator based on the Taylor expansion, as described in Sec. 4.2.2, and (d) *Continuous-Greedy with Power Series approximation* (CG-PS), i.e., Algorithm 7 with a gradient estimator based on the power series expansion, described also in Sec. 4.2.2. In the case of CG-RS, we collect 500 samples, i.e., $T = 500$. In the case of CG-PS we tried the first and second order expansions of the power series as CG-PS1 and CG-PS2, respectively. In the case of CGT, we tried the first-order expansion ($L = 1$). In both cases, subsequent to the execution of Alg. 7 we produce an integral solution in $\mathcal{D}$ by rounding via the swap rounding method [169]. All continuous-greedy algorithms use $\gamma = 0.001$. We also implement a random selection algorithm (RND), which caches c_v items at each node $v \in V$, selected uniformly at random, from the catalog $\mathcal{C}$. We repeat RND 10 times, and report the average running time and caching gain.

4.4.1 Caching Gain Across Different Topologies.

The caching gain $F(\mathbf{x})$ for $\mathbf{x}$ generated by different placement algorithms, is shown for power-law arrival distribution and uniform arrival distribution in Figures 4.5a and 4.5b, respectively. The values are normalized by the gains obtained by RND, reported in Table 4.2. Also, the running times of the algorithms for power-law arrival distribution are reported in Fig. 4.6. As we see in Fig. 4.5, Greedy is comparable to other algorithms in most topologies. However, for topologies `path` and `abilene` Greedy obtains a sub-optimal solution, in comparison to the continuous-greedy



(a) abilene (b) path

Figure 4.7: Caching gain vs. M. As the discrepancy between the service rate of low-bandwidth and high-bandwidth links increases, the performance of Greedy deteriorates.

algorithm. In fact, for `path` and `abilene` Greedy performs even worse than RND. This is precisely because it comes with a worse guarantee $(1/2)$ over these topologies. Note that the 500 samples in CG-RS500 are significantly lower than the value, stated in Theorem 4.2.4, needed to attain the theoretical guarantees of the continuous-greedy algorithm. This is quadratic in $|V||\mathcal{C}|$ ($\sim 10^8$ for, e.g., `ER`). Because of this, in Fig. 4.5, we see that the continuous-greedy algorithms with gradient estimators based on Taylor and Power series expansion, i.e., CG-PS1, CG-PS2, and CGT outperform CG-RS500 in most topologies. Despite this, from Fig. 4.6, we see that CG-RS500 runs 100 times slower than the continuous-greedy algorithms with first-order gradient estimators, i.e., CG-PS1 and CGT.

4.4.2 Varying Service Rates.

For topologies `path` and `abilene`, the approximation ratio of Greedy is ≈ 0.5. This ratio is a function of service rate of the high-bandwidth link M. In this experiment, we explore the effect of varying M on the performance of the algorithms in more detail. We plot the caching gain obtained by different algorithms for `path` and `abilene` topologies, using different values of $M \in \{M_{\min}, 10, 20, 200\}$, where $M_{\min}$ is the value that puts the system on the brink of instability, i.e., 1 and $2+\epsilon$ for `path` and `abilene`, respectively. Thus, we gradually increase the discrepancy between the service rate of low-bandwidth and high-bandwidth links. The corresponding caching gains are plotted in Fig. 4.7, as a function of M. We see that as M increases the gain attained by Greedy worsens in both topologies: when $M = M_{\min}$ Greedy matches the performance of the

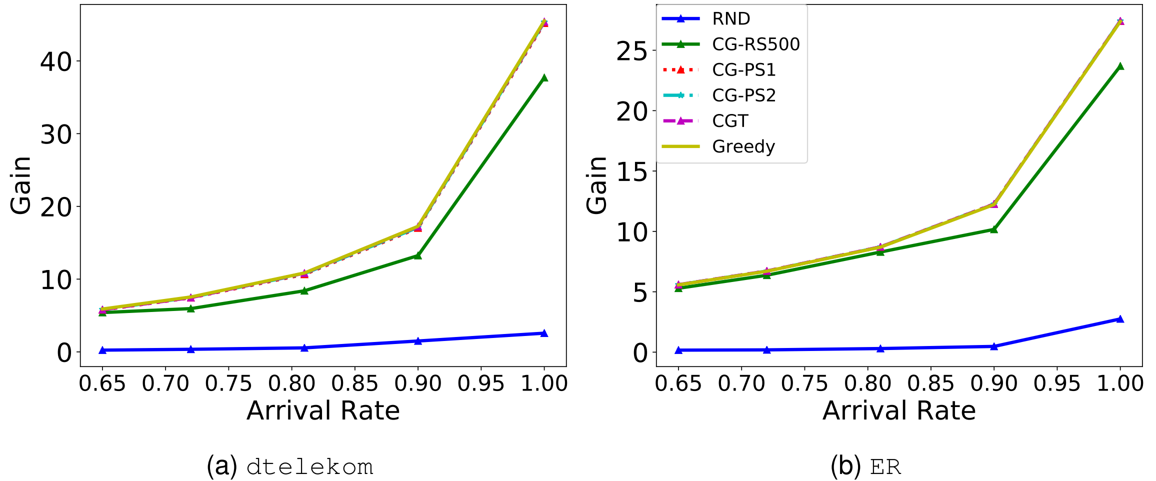


Figure 4.8: Caching gain vs. arrival rate. As the arrival rate increases caching gains get larger.

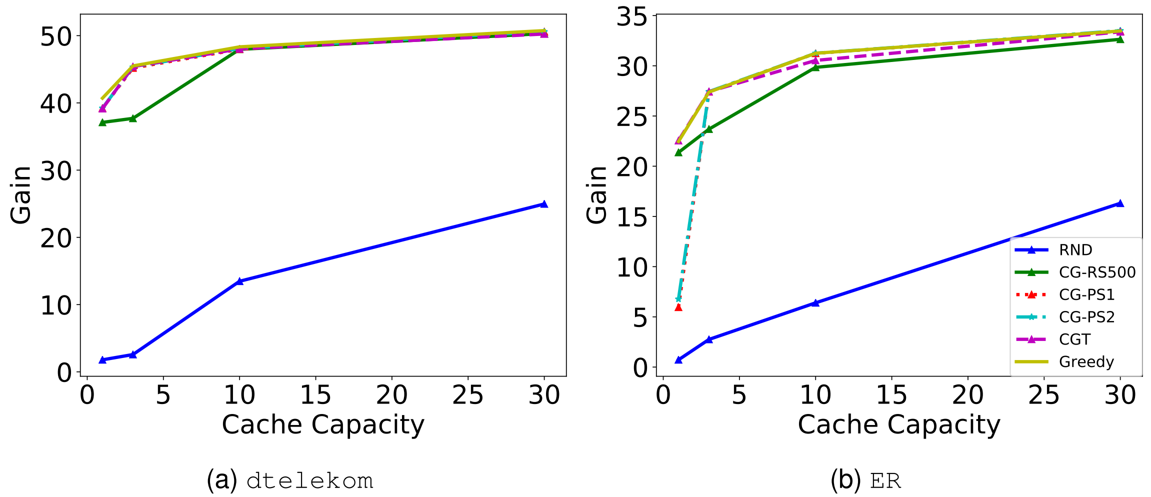
continuous-greedy algorithms, in both cases. However, for higher values of M it is beaten not only by all variations of the continuous-greedy algorithm, but by RND as well.

4.4.3 Effect of Congestion on Caching Gain.

In this experiment, we study the effect of varying arrival rates λ^r on caching gain F. We report results only for the `dtelekom` and `ER` topologies and power-law arrival distribution. We obtain the cache placements $\mathbf{x}$ using the parameters presented in Table 4.2 and different arrival rates: $\lambda^r \in \{0.65, 0.72, 0.81, 0.9, 1.0\}$, for $r \in \mathcal{R}$. Fig. 4.8 shows the caching gain attained by the placement algorithms as a function of arrival rates. We observe that as we increase the arrival rates, the caching gain attained by almost all algorithms, except RND, increases significantly. Moreover, CG-PS1, CG-PS2, CGT, and Greedy have a similar performance, while CG-RS500 achieves lower caching gains.

4.4.4 Varying Caching Capacity.

In this experiment, we study the effect of increasing cache capacity c_v on the acquired caching gains. Again, we report the results only for the `dtelekom` and `ER` topologies and power-law arrival distribution. We evaluate the caching gain obtained by different placement algorithms using the parameters of Table 4.2 and different caching capacities: $c_v \in \{1, 3, 10, 30\}$ for $v \in V$. The caching gain is plotted in Fig. 4.9. As we see, in all cases the obtained gain increases, as we increase



(a) dtelekom (b) ER

Figure 4.9: Caching gain vs. cache capacity. As caching capacities increase, caching gains rise.

the caching capacities. This is expected: caching more items reduces traffic and delay, increasing the gain.

4.5 Conclusion and Future Work

Our analysis suggests feasible object placements targeting many design objectives of interest, including system size and delay, can be determined using combinatorial techniques. Our work leaves open problems relating to stability. This includes the characterization of the stability region of arrival rates $\Lambda = \cup_{\mathbf{x}\in\mathcal{D}}\Lambda(\mathbf{x})$. It is not clear whether determining membership in this set (or, equivalently, given λ, determining whether there exists a $\mathbf{x} \in \mathcal{D}$ under which the system is stable) is NP-hard or not, and whether this region can be somehow approximated. Our recent work [172] tackles this challenge by jointly optimizing the cache placements and the arrival rates λ^r for requests. In particular, we maximize the sum of *utility functions* of the rates, subject to the stability of the network and cache constraints (4.1); however, this results in maximizing a concave function subject to non-convex and non-concave constraints. We use a variant of the *barrier method*, which moves the hard constraints to the objective as a penalty for violating the constraints.

Moreover, we propose a novel Taylor-based gradient estimator (c.f. Sec. 4.2.2), which is the crux of our efficient algorithm. Note that this method can be applied, more generally, to submodular maximization problems where the objective is a composition of a differentiable function with W-DNF polynomials. We explore this direction in our recent work [173]; in particular, we

show that a similar method for estimating the gradients can be adopted for a variety of submodular maximization applications, i.e., influence maximization, document summarization, and facility location. We obtain a theorem similar to Theorem 4.2.9, where the estimator bias B in (4.32) is specifically bounded w.r.t. Taylor expansion degree L. In fact, one of the weaknesses of the current result in Theorem 4.2.9 is that it is not clear whether the bias term goes to zero as L goes to infinity.

Our work leaves the exact characterization of approximable objectives for certain classes of queues, including M/M/1/k queues, open; providing guarantees for M/M/1/k queues, that are not amenable to an analysis via submodular maximization, is also an important open problem. Finally, all algorithms presented in this chapter are offline: identifying how to determine placements in an online, distributed fashion, in a manner that attains a design objective (as in [7, 167]), or even stabilizes the system (as in [151]), remains an important open problem.

Chapter 5

Massively Distributed Graph Distances

Graphs are ubiquitous combinatorial objects, representing real-world phenomena from social and information networks to technological, biological, chemical, and brain networks. Graph distance (or similarity) scores find applications in varied fields, such as image processing [175], chemistry [176, 177], and social network analysis [178, 179]. Graph distances are used in several graph mining tasks, including anomaly detection [180, 181], nearest neighbor and similarity search [182, 183, 184, 180, 185], pattern recognition [182, 185], transfer learning [186], and clustering [1], to name a few.

Distance scores that are *metrics*–and satisfy the triangle inequality property–exhibit significant computational advantages. From a theoretical standpoint, operations such as nearest-neighbor search [187, 188, 189], outlier detection [190], clustering [191, 192, 193], and diameter computation [194] can be computed or approximated efficiently over objects embedded in a metric space. Beyond theoretical guarantees, in practice, metrics often significantly improve performance and/or quality compared to non-metrics in a variety of tasks. For example, graph clustering algorithms are better at detecting clusters over metric spaces. This is illustrated in Fig. 5.1 in the context of graph clustering. Distances between synthetic graphs sampled from well-known random graph families are computed using both metric and non-metric distances. Classification errors by seven hierarchical clustering algorithms are significantly lower when using metrics rather than non-metrics. Metric graph distances are therefore highly desirable.

Graph matching has a long history in machine learning and pattern recognition [195, 196, 175]. Given two graphs, the graph matching problem amounts to finding a node-to-node correspondence (or mapping) that preserves edge relationships across two graphs. This relates to distance computation, as the optimal mapping can be cast as the solution of a minimum distance computation

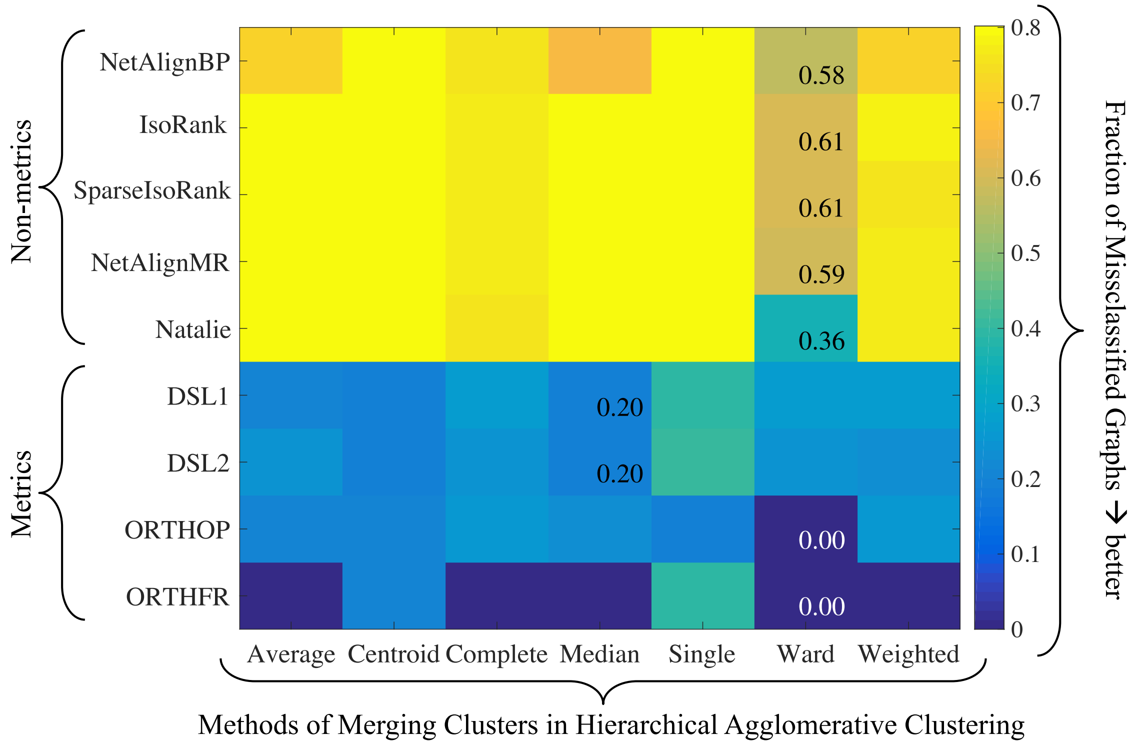


Figure 5.1: **Clustering experiment using metrics and non-metrics (from Bento and Ioannidis [1]).** The graphs with $n = 50$ nodes are sampled from six families (Barabasi Albert, Erdős-Rényi, Regular Graph, Power Law Tree, Small World and Watts Strogatz). The authors compute distances between them using nine different scores; only the last four (DSL1, DSL2, ORTHOP, and ORTHFR) are metrics. Graphs are clustered using hierarchical agglomerative clustering [174] using $Average$, $Centroid$, $Complete$, $Median$, $Single$, $Ward$, $Weighted$ as a means of merging clusters. Colors represent the fraction of misclassified graph pairs, with the minimal misclassification rate per distance labeled explicitly. Metrics outperform other distance scores across all clustering methods. DSL1 and DSL2 correspond to (5.4) for $p = 1$ and 2, respectively, with $\lambda = 0$.

problem. For example, graph matching is commonly formulated as a quadratic assignment problem [175, 197, 198, 199], which is generally NP-hard [195]. There are many works solving this problem approximately (see [175] for a thorough review). NetAlignMR [197] proposes and solves an integer linear programming relaxation. For the same linear relaxation, NetAlignBP [200] uses a more efficient belief propagation (BP) method. Natalie [199] proposes another integer linear programming relaxation. A different approach via a graduated assignment was proposed by Gold and Rangarajan [195]. IsoRank [201] finds a score matrix via a spectral algorithm. Closest to us, Lyzinski et al. [202] propose both a convex and non-convex relaxation over the set of doubly stochastic matrices: their convex relaxation is Eq. (5.2) with $\lambda = 0$ and $p = 2$, while the objective in the non-convex

relaxation is the quadratic function $\mathrm{trace}((\mathbf{AP})^\top \mathbf{PB})$. Schellewald and Schnörr [203] propose a semi-definite programming relaxation. Though highly efficient, these approaches generally do not yield distances that are metrics (see [1]).

Well-known graph distances that are metrics include the so-called *chemical* [177] and the *Chartrand-Kubiki-Shultz (CKS)* [204] distances. The chemical distance between two graphs G_A and G_B with adjacency matrices $\mathbf{A}, \mathbf{B} \in \{0,1\}^{n\times n}$, respectively, is defined as:

$$\min_{\mathbf{P}\in\mathbb{P}^n} \|\mathbf{AP} - \mathbf{PB}\|_2, \tag{5.1}$$

where $\mathbb{P}^n$ is the set of permutation matrices, and $\|\cdot\|_2$ is the $p = 2$ (a.k.a. Frobenius) norm. Intuitively, the solution to Prob. (5.1) counts the number of edges present in one graph but not the other, under a node correspondence (mapping) captured by permutation matrix $\mathbf{P}$. The CKS distance has the same formulation, replacing the adjacency matrices with matrices comprising shortest path distances. Unfortunately, both distances are computationally intractable [205].

To address this, Bento and Ioannidis [1] recently proposed a *convex relaxation* of these distances, which attains tractability while also naturally incorporating node features. In a nutshell, the authors define the distance between two n-node graphs G_A and G_B as the optimal value of the problem:

$$\min_{\mathbf{P}\in\mathcal{W}^n} \|\mathbf{AP} - \mathbf{PB}\|_p + \lambda \cdot \mathrm{trace}\left(\mathbf{P}^\top \mathbf{D}_{\mathbf{A},\mathbf{B}}\right), \tag{5.2}$$

where $\mathcal{W}^n$ is the set of doubly stochastic matrices, $\|\cdot\|_p$ is the entry-wise p-norm, $\mathbf{D}_{\mathbf{A},\mathbf{B}} \in \mathbb{R}^{n\times n}$ denotes dissimilarities between the nodes of the two graphs, and $\lambda \geq 0$ is a hyper-parameter.

The relaxation of the chemical distance defined by Prob. (5.2) has several advantages. First, it is tractable, as it involves solving a convex optimization problem. Second, Bento and Ioannidis show that the distance resulting from solving Prob. (5.2) is a metric and, in particular, satisfies the triangle inequality. This yields the aforementioned benefits of metrics in downstream tasks such as, e.g., graph clustering or nearest-neighbor search. Third, it incorporates node features via the linear trace term. This has computational advantages (which we discuss in Section 5.1.6), but is also important in practice: nodes in real-life graphs often contain such information (e.g., demographic information of users in a social network, atom properties in a molecule, etc.). Finally, Prob. (5.4) encompasses multiple p-norms and possible distance matrices $D_{A,B}$, for which both the metric property and convexity are maintained [1]. The ability to span different norms is also very important in practice, as the right value of p can be data dependent (see Tables 5.2 and 5.3 in Sec. 5.5.3).

Even though Prob. (5.2) is a convex optimization problem, the number of variables is quadratic in the graph size n; this makes traditional optimization methods for solving (5.2) prohibitive even for small n. Nevertheless, for $p = 1$, the problem can be solved in a distributed fashion via the Alternating Directions Method of Multipliers (ADMM) [13], since its objective decomposes into a sum of simpler objective functions. Unfortunately, it is not clear how to efficiently distribute the solution for $p > 1$; this is precisely because, for $p > 1$, the objective of (5.2) cannot be written as a sum of distinct terms. Our present work directly addresses this challenge: we propose a distributed algorithm solving (5.2) for arbitrary $p \geq 1$. Our solution combines ADMM with a distributed proximal operator for arbitrary p-norms, which is both novel and of independent interest. Finally, we demonstrate the applicability of our algorithm via massively distributed implementations over OpenMP and Apache Spark, which we make publicly available.[1] In summary, we make the following contributions:

- We propose an ADMM-based distributed algorithm for solving (5.4) *for all* $p \geq 1$. Our solution for the case $p > 1$ uses a nested-ADMM (Alg. 8 and 9) in combination with a distributed bisection algorithm (Alg. 10) as building blocks.
- We describe the algorithm's parallel complexity in terms of the sparsity of graphs G_A, G_B, and additional constraints we introduce in the problem. In particular, we bound message exchanges in terms of these sparsity parameters.
- We implement our algorithm in OpenMP [43] and Spark [33]. Our publicly available implementation scales to hundreds of CPUs. Over a 448 CPU cluster, we attain speedups as much as 153×.

The remainder of this chapter is organized as follows. We introduce graph distances in Sec. 5.1. We review proximal operators along with consensus ADMM in Sec. 5.2. We present our main algorithm and its complexity analysis in Sections 5.3 and 5.4, respectively. Finally, we describe our experiments in Sec. 5.5. We conclude in Sec. 5.6.

5.1 Graph Distances

In this section, we briefly review the technical material necessary for understanding this chapter. We start by introducing notations and basic definitions related to graphs and norms. We

[1] `https://github.com/neu-spiral/GraphMatching`

then review chemical distances, metrics, and proximal operators in Sections 5.1.2, 5.1.3, and 5.2.1, respectively. We finally review the consensus ADMM in Sec. 5.2.2.

Graphs. We represent a graph $G(\mathcal{V},\mathcal{E})$ with node set $\mathcal{V}=[n]\equiv\{1,\ldots,n\}$ and edge set $\mathcal{E}\subseteq[n]\times[n]$ by its *adjacency matrix*, i.e., $\mathbf{A}=[a_{ij}]_{i,j\in[n]}\in\{0,1\}^{n\times n}$ s.t. $a_{ij}=1$ iff $(i,j)\in\mathcal{E}$. A graph is *bipartite* if its node set can be partitioned into two disjoint sets $\mathcal{V}_L$ and $\mathcal{V}_R$ such that no edges exist within the same partition, i.e., $\mathcal{E}\subseteq\mathcal{V}_L\times\mathcal{V}_R$. We denote bipartite graphs by $G(\mathcal{V}_L,\mathcal{V}_R,\mathcal{E})$.

Matrix Norms and Projections. Given a matrix $\mathbf{A}=[a_{ij}]_{i,j\in[n]}\in\mathbb{R}^{n\times n}$ and $p\in\mathbb{R}_+$, where $p\geq 1$, its *entry-wise p-norm* is $\|\mathbf{A}\|_p=(\sum_{i=1}^n\sum_{j=1}^n|a_{ij}|^p)^{1/p}$. We use $\|\mathbf{A}\|_0$ to indicate the number of non-zero elements (a.k.a. the size of the support) of $\mathbf{A}$, i.e., $\|\mathbf{A}\|_0\equiv|\{(i,j):a_{ij}\neq 0\}|=|\operatorname{supp}(\mathbf{A})|$.Given a vector $\mathbf{x}\in\mathbb{R}^n$ and an ordered set $\mathcal{S}\subseteq[n]$, we denote the projection of $\mathbf{x}$ on a subset $\mathcal{S}$ of its coordinates by $\mathbf{x}_\mathcal{S}\in\mathbb{R}^{|\mathcal{S}|}$. Similarly, given a matrix $\mathbf{A}=[a_{ij}]_{i,j\in[n]}\in\mathbb{R}^{n\times n}$ and a set $\mathcal{S}\subseteq[n]\times[n]$, we define $\mathbf{A}_\mathcal{S}\in\mathbb{R}^{|\mathcal{S}|}$ to be the projection of $\mathbf{A}$ on its coordinates in $\mathcal{S}$; that is, $\mathbf{A}_\mathcal{S}$ is the $|\mathcal{S}|$-dimensional vector comprising the elements $a_{ij},(i,j)\in\mathcal{S}$. We denote by $\mathbb{P}^n=\{\mathbf{P}\in\{0,1\}^{n\times n}:\mathbf{P}\mathbf{1}=\mathbf{1},\mathbf{P}^\top\mathbf{1}=\mathbf{1}\}$ the set of *permutation* matrices and by $\mathbb{W}^n=\{\mathbf{P}\in[0,1]^{n\times n}:\mathbf{P}\mathbf{1}=\mathbf{1},\mathbf{P}^\top\mathbf{1}=\mathbf{1}\}$ the set of *doubly-stochastic* matrices (i.e., the *Birkhoff polytope*).

5.1.1 The Weisfeiler-Lehman (WL) Algorithm

The WL algorithm [206] is a heuristic for solving the graph isomorphism problem. Note that two isomorphic graphs must have the same degree distribution. More broadly, the distributions of k-hop neighborhoods in the two graphs must also be identical. We use this algorithm to generate the constraint set $\mathcal{E}$ in our experiments, in Sect. 5.5. To gain some intuition on the algorithm, note that two isomorphic graphs must have the same degree distribution. More broadly, the distributions of k-hop neighborhoods in the two graphs must also be identical. Building on this observation, to test graph isomorphism, WL *colors* the nodes of a graph $G(V,E)$ iteratively. At iteration 0, each node $v\in V$ receives the same *color* $c^0(v):=1$. Colors at iteration $k+1\in\mathbb{N}$ are defined recursively via $c^{k+1}(v):=\mathsf{hash}\left(\mathsf{sort}\left(\mathsf{clist}_v^k\right)\right)$ where hash is a perfect hash function, and $\mathsf{clist}_v^k=[c^k(u):(u,v)\in E)]$ is a list containing the colors of all of v's neighbors at iteration k. Intuitively, two nodes in V share the same color after k iterations if their k-hop neighborhoods are isomorphic. WL terminates when the partition of V induced by colors is stable from one iteration to the next.

5.1.2 Graph Distances.

A distance between two graphs can be defined naturally when they are labeled, i.e., the correspondence between their nodes is known (see, e.g., [207, 179, 208]). Two classic examples are the edit distance [209, 210] and the maximum common subgraph distance [211, 212]. Some recent works focus on distances for labeled graphs that are easy to compute (e.g in linear or quadratic time) [207, 179, 208] without maintaining the properties of a metric. We study the (harder) unlabeled setting, in which the node correspondence between graphs is unknown. Examples of distances in this setting include the chemical [177] and the Chartrand-Kubiki-Shultz (CKS) [204] distances, while the edit and the maximum common subgraph distances can also be extended to the unlabeled setting. All four [211, 212, 213, 214] are metrics and hard to compute, while existing heuristics (e.g., [215, 216]) do not satisfy the triangle inequality property. A simple approach to induce a metric over unlabeled graphs is to embed them in a common metric space and then measure the distance of these embeddings. Riesen et al. [217, 218] embed graphs into real vectors by computing their edit distances to a set of *prototype* graphs. The same embedding is also used to compute a median of graphs [219]. Other works [220, 221, 222] map graphs to spaces determined by their spectral decomposition. Such approaches are not as discriminative as the metrics considered here [1], because embeddings only summarize the graph structure.

Chemical Distance Let $\mathbf{A}, \mathbf{B} \in \{0,1\}^{n\times n}$ be the adjacency matrices of two graphs $G_A(\mathcal{V}, \mathcal{E}_A)$ and $G_B(\mathcal{V}, \mathcal{E}_B)$. Graphs G_A and G_B are *isomorphic* if and only if there exists $\mathbf{P} \in \mathbb{P}^n$ s.t. $\mathbf{P}^\top \mathbf{A}\mathbf{P} = \mathbf{B}$ or, equivalently, $\mathbf{A}\mathbf{P} = \mathbf{P}\mathbf{B}$. The *chemical distance* extends the latter relationship to capture graph distances. The chemical distance between G_A and G_B is defined via Prob. (5.1). Intuitively, Prob. (5.1) counts the number of edges present in one graph but not the other, under a node correspondence (mapping) captured by permutation matrix $\mathbf{P}$. Unfortunately, there is no poly-time algorithm for solving (5.1) [205].

5.1.3 Metrics.

Given a set Ω, a function $d : \Omega \times \Omega \to [0, \infty)$ is called a *metric*, and the pair (Ω, d) is called a *metric space*, if d satisfies the following properties for all $x, y, z \in \Omega$:

$$d(x,y) \geq 0 \qquad \text{(non-negativity)} \qquad (5.3a)$$

$$d(x,y)=0 \text{ iff } x=y \qquad \text{(pos. definiteness)} \qquad (5.3b)$$

$$d(x,y) = d(y,x) \qquad \text{(symmetry)} \qquad (5.3c)$$

$$d(x,y) \leq d(x,z)+d(z,y) \qquad \text{(triangle inequality)} \qquad (5.3d)$$

A function d is called a *pseudometric* if it satisfies (5.3a), (5.3c), and (5.3d), but the positive definiteness property (5.3b) is replaced by the (weaker) property:

$$d(x,x) = 0 \text{ for all } x \in \Omega. \qquad (5.3e)$$

If d is a pseudometric, then $d(x,y) = 0$ defines an equivalence relation $x \sim_d y$ over Ω. A pseudometric is then a metric over $\Omega/\sim_d$, the quotient space of $\sim_d$. A d that satisfies (5.3a), (5.3b), and (5.3d) *but not* the symmetry property (5.3c) is called a *quasimetric*. If d is a quasimetric, then its *symmetric extension* $\bar{d} : \Omega \times \Omega \to \mathbb{R}$, defined as $\bar{d}(x,y) = d(x,y) + d(y,x)$, is a metric over Ω.

Metrics naturally arise in data mining tasks, including clustering [223, 174], nearest neighbour search [187, 188, 189], and outlier detection [190]. Some of these tasks become tractable, or admit formal guarantees, precisely when performed over a metric space. For example, finding the nearest neighbor [187, 188, 189] or the diameter of a dataset [194] become polylogarithimic under metric assumptions; similarly, approximation algorithms for clustering (which is NP-hard) rely on metric assumptions, whose absence leads to a deterioration of known bounds [191]. Our focus on metrics is motivated by these considerations.

5.1.4 Convex Relaxation

Bento and Ioannidis [1] introduce a tractable family of distances that generalizes the chemical distance. The family can be expressed via convex optimization problems, that can be solved via, e.g., barrier methods; nevertheless, the number of variables is quadratic in the graph size n, which motivates our exploration of a distributed implementation.

Formally, given the n-node graphs $G_A(\mathcal{V}, \mathcal{E}_A)$ and $G_B(\mathcal{V}, \mathcal{E}_B)$, where $\mathcal{V} = [n]$, Bento and Ioannidis suggest computing the distance between graphs as the minimum of the following problem:

$$\text{Minimize} \quad \|\mathbf{AP} - \mathbf{PB}\|_p + \lambda \cdot \text{trace}\left(\mathbf{P}^\top \mathbf{D}_{\mathbf{A},\mathbf{B}}\right), \tag{5.4a}$$

$$\text{subj. to:} \quad \mathbf{P} \in \mathbb{W}^n, \quad p_{ij} = 0 \text{ for all } (i,j) \notin \mathcal{Q}, \tag{5.4b}$$

where $\mathcal{Q} \subseteq [n] \times [n]$ is a set of pairs constraining the support of $\mathbf{P}$, $\mathbf{D}_{\mathbf{A},\mathbf{B}} = [d_{ij}]_{(i,j)\in[n]\times[n]}$ is a matrix, s.t., d_{ij} measures the dissimilarity between some features of the nodes $i \in \mathcal{V}$ and $j \in \mathcal{V}$, and $\lambda \geq 0$ is a tuning parameter.

Intuitively, Prob. (5.4) finds a stochastic mapping between nodes that minimizes edge discrepancy, while also taking into account node feature distances as well as hard constraints. More specifically, the doubly-stochastic matrix $\mathbf{P}$ can be interpreted as a stochastic mapping, where $p_{ij} \in [0, 1]$ shows the probability that node i in G_A is mapped to node j in G_B. Prob. (5.4) thus seeks a stochastic mapping $\mathbf{P}$ that (a) minimizes the edge discrepancy between adjacency matrices, captured by term $\|\mathbf{AP} - \mathbf{PB}\|$, (b) penalizes mappings between nodes i in G_A and node j in G_B that have distinct features, captured by linear term $\text{trace}\left(P^\top \mathbf{D}_{\mathbf{A},\mathbf{B}}\right)$, and (c) further restricts mappings to have support in $\mathcal{Q}$.

We discuss examples illustrating different feature distance matrices $\mathbf{D}_{\mathbf{A},\mathbf{B}}$ and constraints $\mathcal{Q}$ below, in Sec. 5.1.5. In short, node features can be incorporated in a *soft* manner, through the linear term in objective (5.4a), or as *hard* constraints in $\mathcal{Q}$ (requiring, e.g., nodes with different categorical features to never be mapped to each other).

Computing distances via Prob. (5.4) has several important advantages. First, under mild conditions on $\mathbf{D}_{\mathbf{A},\mathbf{B}}$ and $\mathcal{Q}$, the distance computed by Prob. (5.4) is a metric; this is proved by Bento and Ioannidis [1]. Second, for arbitrary p-norms, (5.4) is a convex optimization problem. As a result, a solution can be computed using standard methods [31]. Third, the linear term $\text{trace}\left(\mathbf{P}^\top \mathbf{D}_{\mathbf{A},\mathbf{B}}\right)$ and the constraints $\mathcal{Q}$ allow us to capture auxiliary information that often exists in practice, such as node features or labels. Beyond this expressive power, both have significant computational advantages, as we show in Sec. 5.5.

5.1.5 Constraints and Node Features

In practice, graph nodes are often endowed with features or attributes that we can leverage in graph distance computations. Here, we explain how node features can be incorporated in Prob. (5.4) via either the linear term or the constraint set $\mathcal{Q}$.

Node Features in $\mathbb{R}^d$. Node attributes can be represented as, e.g, k-dimensional feature vectors in $\mathbb{R}^d$. Having access to such features, we can compute the elements of the dissimilarity matrix $\mathbf{D}_{\mathbf{A},\mathbf{B}} = [d_{ij}]_{i,j\in[n]}$ by taking, e.g., the ℓ_2 (or other vector) norm of the difference between these vectors: that is $d_{ij} = \|x_i - x_j\|_2$, where $x_i, x_j \in \mathbb{R}^k$ are the k-dimensional feature vectors for i in G_A and j in G_B.

Node features can be *exogenous*, e.g., the demographic attributes of a user in a social network, the atomic number of an atom in a molecule, etc. Alternatively, features can be *endogenous*, i.e., computed directly from the adjacency matrix: these include, e.g., a node's degree, it's centrality, its pagerank [224], node2vec representation [225, 226], or some other vector computed via graph signal processing [227, 225]. Exogenous features are often available in practical settings, while endogenous features can have computational advantages: we observe this in Sec. 5.5, where adding a linear term often accelerates convergence but also produces higher quality solutions.

Categorical Features (Colors/Labels). Rather than including categorical node features as *soft* constraints, via the trace penalty, such features can also be used to produce *hard* constraints, captured by $\mathcal{Q}$. Suppose that we are given a categorical node feature, referred to as a node's *color*. We can construct the constraint set $\mathcal{Q}$ by including only pairs (i, j) s.t. the nodes i and j across the two graphs have the same color.

Colors can again be either exogenous or endogenous/structural. As examples of exogenous colors, if the graph represents an organic molecule, the color can be the node's atomic number; then, constraint $\mathcal{Q}$ requires that identical atoms are mapped to each other across the two graphs. If the graph represents a social network, colors can correspond to different demographic attributes, (e.g., gender, age group, etc.) Structural/endogenous colors, on the other hand, can be categorical variables capturing the local neighborhood structure around a node. These can be, e.g., node degrees, the number of triangles that pass through a node, or some other discrete statistic generated from a node's k-hop neighborhood. One such statistic is the output of the so-called Weisfeiler-Lehman (WL) algorithm [206], executed after k iterations (see Appendix 5.1.1).

Using categorical variables of the above nature to construct constraint set $\mathcal{Q}$ has several advantages. First, Bento and Ioannidis show that Prob. (5.4) remains a metric, even when incorporating such constraints. Most importantly, introducing constraints can significantly decrease the number of optimization variables and, hence, the computational complexity of Prob. (5.4). As we discuss in Section 5.4, the sparsity of $\mathcal{Q}$ also dictates the communication complexity our parallel algorithm for solving Prob. (5.4).

5.1.6 Importance of p-norms and Linear Term

Given the size of both the input graphs, our goal is to produce a distributed algorithm for solving Prob. (5.4). As we discuss in Sec. 5.3, the main challenge arises from presence of the p-norm in combination with the linear term. One possible solution is to limit objective (5.4a) to the case $p = 1$. This leads to an objective paralellizable via consensus ADMM. This, on the other hand, is unsatisfactory, as the ideal norm may depend on the underlying graphs; we elaborate on this in Sec. 5.5, where we see how inherent noise can effect our norm choice (see Sec. 5.5.3). Another solution is to modify objective (5.4a), replacing $\|\cdot\|_p$ with $\|\cdot\|_p^p$. This has two significant drawbacks. First, under this modification, the distance is no longer a metric; in particular, it fails to satisfy the triangle inequality, which is a significant disadvantage for downstream applications, as mentioned earlier. Second, from an optimization standpoint, it is important to keep the two terms in objective (5.4a) balanced; this is harder in this case as $\|\cdot\|_p^p$ is not absolutely homogeneous (in contrast to both the trace and norms).

A final alternative is to remove the linear term altogether. In this case, minimizing $\|\cdot\|_p$ is equivalent to minimizing $\|\cdot\|_p^p$. This annuls any benefits of incorporating features, both in terms of modeling, e.g., exogenous node attributes, but also in terms of efficiency: as our experiments demonstrate (see, e.g., Fig. 5.3 in Sec. 5.5.2), including the linear term can significantly accelerate convergence.

5.2 Proximal Operators and Consensus ADMM

5.2.1 Proximal Operators.

The proximal operator of a lower semi-continuous function (see Sec. 2.1 for a formal definition) $f : \mathbb{R}^d \to \mathbb{R}$ is defined as follows:

$$\mathbf{x}^* := \arg\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x}) + \frac{\rho}{2}\|\mathbf{x}-\mathbf{u}\|_2^2, \tag{5.5}$$

where $\rho > 0$ and $\mathbf{u} \in \mathbb{R}^d$ is a given point. Note that in case the function f is a characteristic function $\chi_{\mathcal{D}}(\cdot)$ for a set $\mathcal{D}$, the proximal operator is equivalent to projection on the set $\mathcal{X}$. In that sense the proximal operator is a generalization of projection on sets.

We use a bisection algorithm due to Liu and Ye [68] to compute the proximal operator of p-norms. The original presentation of the algorithm was serial; we show (and exploit) in our work the fact that the algorithm can be implemented in parallel via map-reduce operations. Beyond this,

we also provide a convergence guarantee (Thm. 5.3.3), which was absent from their work. Sra [228] extends Liu and Ye's approach, proposing a bisection method for finding the proximal operators of mixed $\ell_{1,p}$ norms. The same author also provides a proximal operator algorithm for mixed $\ell_{p,q}$ norms in a follow-up work [229]. Proximal operators can be seen as generalizations of projection operators [230]; in the case of norms, they are coupled to projections via Moreau's decomposition [231]. Exploiting the latter, some works solve the problem in the dual domain via projections on the unit ball of the dual norm [230] or via gradient methods [232]. These methods are not readily parallelizable.

5.2.2 Consensus ADMM

The Alternating Direction Method of Multipliers (ADMM) [83] is a convex optimization algorithm, refer to Sec. 2.2.3 for more details. Here we focus on consensus ADMM [13], which is a classic approach to distribute optimization problems; its applications are numerous [233, 234, 198, 235, 14, 236, 237]. For strongly convex problems, its optimally-tuned convergence rate is as fast as that of the fastest first-order method [238]. Though we focus on the simplest setting, extensions include asynchronous [14] and stochastic [237] versions, adaptive ways of updating parameter ρ [236], and faster variants that solve subproblems inexactly [239]. Applying such optimizations to our work is an interesting open question.

Consensus ADMM is an iterative optimization algorithm well-suited for solving convex optimization problems in a distributed fashion. Problems amenable to a distributed solution via consensus ADMM have a specific form: their objective can be written as a sum of functions, each depending only on a few variables. Formally, consider the optimization problem:

$$\text{Minimize} \quad F(\mathbf{x}) = \textstyle\sum_{i=1}^{N} F_i(\mathbf{x}_{\mathcal{S}_i}), \tag{5.6}$$

where $\mathbf{x} \in \mathbb{R}^n$ and each term $F_i : \mathbb{R}^{|\mathcal{S}_i|} \to \mathbb{R}$ is convex and depends on a subset $\mathcal{S}_i \subseteq [n]$ of the coordinates of $\mathbf{x}$. Prob. (5.6) can be re-written with N *local* variables $\mathbf{x}_i \in \mathbb{R}^{|\mathcal{S}_i|}, i \in [N]$ and a single *consensus* variable $\mathbf{z} \in \mathbb{R}^n$ as:

$$\text{Minimize} \quad \textstyle\sum_{i=1}^{N} F_i(\mathbf{x}_i) \tag{5.7a}$$

$$\text{subj. to:} \quad \mathbf{x}_i = \mathbf{z}_{\mathcal{S}_i} \;\; i = 1, \ldots, N, \tag{5.7b}$$

where $\mathbf{z}_{\mathcal{S}_i}$ is the projection of $\mathbf{z}$ on the subset $\mathcal{S}_i$. The k-th iteration of consensus ADMM for (5.7) is

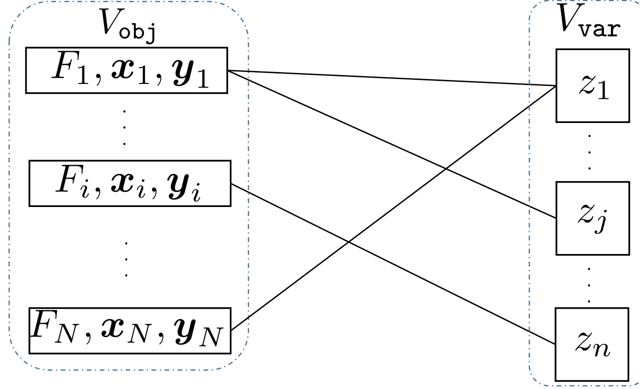


Figure 5.2: A bipartite graph $\mathcal{G}(\mathcal{V}_{\texttt{obj}}, \mathcal{V}_{\texttt{var}}, \mathcal{E}_{\mathcal{G}})$ showing the dependencies of the functions F_i on the coordinates of the global consensus variable z, as well as communication pattern during parallelism. Each node in the graph corresponds a processor. Processors in $\mathcal{V}_{\texttt{obj}}$ store F_i, $\mathbf{x}_i$, $\mathbf{y}_i$, $i \in [N]$, and perform steps (5.8a) and (5.8c), while processors in $\mathcal{V}_{\texttt{var}}$ store z_i, $i \in [n]$, and perform (5.8b).

as follows:

$$\mathbf{x}_i^{k+1} = \arg\min_{\mathbf{x}_i} F_i(\mathbf{x}_i) + \tfrac{\rho}{2}\|\mathbf{x}_i - \mathbf{z}_{\mathcal{S}_i}^k + \mathbf{y}_i^k\|_2^2, \ \forall i \in [N], \tag{5.8a}$$

$$\mathbf{z}_j^{k+1} = \frac{\sum_{i:j\in\mathcal{S}_i}\big((\mathbf{x}_i^{k+1})_{\ell_i(j)} + (\mathbf{y}_i^k)_{\ell_i(j)}\big)}{|\{i\in[N]: j\in\mathcal{S}_i\}|}, \quad \forall j \in [n], \tag{5.8b}$$

$$\mathbf{y}_i^{k+1} = \mathbf{y}_i^k + (\mathbf{x}_i^{k+1} - \mathbf{z}_{\mathcal{S}_i}^{k+1}), \quad \forall i \in [N], \tag{5.8c}$$

where $\rho > 0$ is a tuning parameter and $\mathbf{y}_i \in \mathbb{R}^{|\mathcal{S}_i|}$, $i \in [n]$, are dual variables corresponding to the constraints (5.7b). and $\ell_i : \mathcal{S}_i \to \{1, \ldots, |\mathcal{S}_i|\}$ maps coordinates in $\mathcal{S}_i$ to their "local" representations in $\mathbf{x}_i$. Under appropriate conditions [13], including the convexity of functions F_i, iterations (5.7) are guaranteed to converge to an optimal solution of (5.6). Additional details, and convergence criteria, are provided in Sec. 5.2.2.

Incorporating Constraints. We can include constraints in ADMM by adding them to the objective (5.7a) via their characteristic functions: a constraint $\mathbf{x} \in \mathcal{D}$, where $\mathcal{D}$ is a convex set, is added to (5.7a) as a term $\chi_{\mathcal{D}}(\mathbf{x})$, where $\chi_{\mathcal{D}}$ is the characteristic function of $\mathcal{D}$ (0 if $\mathbf{x} \in \mathcal{D}$, $+\infty$ o.w.). Then, the corresponding step (5.8a) becomes a Euclidean projection onto convex set $\mathcal{D}$.

A Parallel Implementation. All the above steps in (5.8) can be parallelized. To see this, suppose that we have $N + n$ processors, as illustrated in Fig. 5.2. The N processors in $\mathcal{V}_{\texttt{obj}} \equiv [N]$ are responsible for solving problems (5.8a) and performing the dual variable adaptation (5.8c), in parallel. To do so, they store functions F_i as well as "local" primal and dual variables $\mathbf{x}_i$,$\mathbf{y}_i$, $i \in [N]$. The remaining n processors $\mathcal{V}_{\texttt{var}} = [n]$ store the coefficients z_j, $j \in [n]$, of the consensus variable $\mathbf{z}$ and perform the averaging (5.8b). In each iteration, the processors in $\mathcal{V}_{\texttt{var}}$ send the consensus variables

to the corresponding processors in $\mathcal{V}_{\texttt{obj}}$. Subsequently, the latter perform adaptations (5.8c) and (5.8a), and then send their new local variables to the processors in $\mathcal{V}_{\texttt{var}}$ for averaging.

The communication complexity of each step (5.8), as well as the dependencies between steps, are determined by the bipartite graph $\mathcal{G}(\mathcal{V}_{\texttt{obj}}, \mathcal{V}_{\texttt{var}}, \mathcal{E}_{\mathcal{G}})$ shown in Fig. 5.2: each processor $i \in [N]$ on the left needs to receive the $|\mathcal{S}_i|$ consensus variables z_j, $j \in \mathcal{S}_i$ to perform (5.8a) and (5.8c), while processors $j \in [n]$ on the right need to collect $|\{i \in [N] : j \in \mathcal{S}_i\}|$ local variables $(x_i)_{\ell_i(j)}$. As a result, the number of messages exchanged is proportional to the number of edges in $\mathcal{G}$, namely, $\sum_{i\in[N]} |\mathcal{S}_i|$.

5.3 Distributed Graph Distances via ADMM

We now turn our attention to solving (5.4) via ADMM. We incorporate constraints (5.4b) in (5.4a), yielding objective:

$$\|\mathbf{AP}-\mathbf{PB}\|_p+\lambda \operatorname{trace}\left(\mathbf{P}^\top \mathbf{D}_{\mathbf{A},\mathbf{B}}\right) + \chi_{\mathcal{R}}(\mathbf{P}) + \chi_{\mathcal{C}}(\mathbf{P}), \tag{5.9}$$

where the sets

$$\mathcal{R} = \{\mathbf{P} \in [0,1]^{n\times n} : \mathbf{P1} = \mathbf{1}, p_{ij} = 0\ \forall (i,j) \notin \mathcal{Q}\}, \text{ and}$$
$$\mathcal{C} = \{\mathbf{P} \in [0,1]^{n\times n} : \mathbf{P}^\top\mathbf{1} = \mathbf{1}, p_{ij} = 0\ \forall (i,j) \notin \mathcal{Q}\},$$

correspond to the (doubly stochastic) constraints on the rows and columns, respectively. With the exception of the first term, all remaining terms in (5.9) can be written as sums. Indeed, the following lemma holds:

Lemma 5.3.1. *There exists a set $\mathcal{I} \subseteq [n] \times [n]$ as well as sets $\mathcal{S}_{ij} \subseteq [n] \times [n]$, $\mathcal{S}_i \subseteq [n] \times [n]$, $\mathcal{S}_j \subseteq [n] \times [n]$, for $i, j \in [n]$, such that the terms in* (5.9) *can be written as:*

$$\|\mathbf{AP} - \mathbf{PB}\|_p = \left(\textstyle\sum_{(i,j)\in\mathcal{I}} |f_{ij}(\mathbf{P}_{\mathcal{S}_{ij}})|^p\right)^{\frac{1}{p}}, \tag{5.10a}$$

$$\operatorname{trace}(\mathbf{P}^\top \mathbf{D}_{\mathbf{A},\mathbf{B}}) = \textstyle\sum_{(i,j)\in\mathcal{Q}} p_{ij} d_{ij}, \tag{5.10b}$$

$$\chi_{\mathcal{R}}(\mathbf{P}) = \textstyle\sum_{i\in[n]} \chi_{\mathcal{R}^{(i)}}(\mathbf{P}_{\mathcal{S}_i}), \tag{5.10c}$$

$$\chi_{\mathcal{C}}(\mathbf{P}) = \textstyle\sum_{j\in[n]} \chi_{\mathcal{C}^{(j)}}(\mathbf{P}_{\mathcal{S}_j}), \tag{5.10d}$$

where $f_{ij}(\cdot)$, $(i,j) \in \mathcal{I}$, are affine functions and

$$\mathcal{R}^{(i)} = \{\mathbf{p} \in [0,1]^{|\mathcal{S}_i|} | \mathbf{1}^\top \mathbf{p} = 1\}, \tag{5.11}$$

$$\mathcal{C}^{(j)} = \{\mathbf{p} \in [0,1]^{|\mathcal{S}_j|} | \mathbf{1}^\top \mathbf{p} = 1\}, \tag{5.12}$$

for $i \in [n], j \in [n]$, are the $|\mathcal{S}_i|$-dimensional and $|\mathcal{S}_j|$-dimensional simplices, respectively.

Proof. For $(i,j) \in [n] \times [n]$ the (i,j)-th element of $\mathbf{AP} - \mathbf{PB}$ is:

$$(\mathbf{AP} - \mathbf{PB})_{ij} = \textstyle\sum_{k:(k,j)\in\mathcal{S}_L^{(i,j)}} a_{ik}p_{kj} - \sum_{k:(i,k)\in\mathcal{S}_R^{(i,j)}} p_{ik}b_{kj},$$

where $\mathcal{S}_L^{(i,j)} = \{(k,j) \in \mathcal{Q} | (i,k) \in \mathcal{E}_A\}$ and $\mathcal{S}_R^{(i,j)} = \{(i,k) \in \mathcal{Q} | (k,j) \in \mathcal{E}_B\}$. We can write $(\mathbf{AP} - \mathbf{PB})_{ij}$ as $f_{ij}(\mathbf{P}_{\mathcal{S}_{ij}})$, where $f_{ij}(\mathbf{P}_{\mathcal{S}_{ij}}) \triangleq \sum_{k:(k,j)\in\mathcal{S}_L^{(i,j)}} a_{ik}p_{kj} - \sum_{k:(i,k)\in\mathcal{S}_R^{(i,j)}} p_{ik}b_{kj}$, where $\mathcal{S}_{ij} = \mathcal{S}_L^{(i,j)} \cup \mathcal{S}_R^{(i,j)}$ and $f_{ij} : \mathbb{R}^{|\mathcal{S}_{ij}|} \to \mathbb{R}$ is a linear function. The entry-wise p norm of $AP - PB$ is thus (5.10a), where $\mathcal{I}$ comprises pairs (i,j) for which $\mathcal{S}_{ij} \neq \emptyset$. On the other hand, $\mathrm{trace}(\mathbf{P}^\top \mathbf{D}_{\mathbf{A},\mathbf{B}}) = \sum_{(i,j)\in\mathcal{Q}} p_{ij}d_{ij}$ by the fact that $p_{ij} = 0$ for $(i,j) \notin \mathcal{Q}$. The function $\chi_{\mathcal{R}}(\mathbf{P})$ states that each row i of $\mathbf{P}$ belongs to the set: $\mathcal{R}^{(i)} = \{\mathbf{p}^{(i)} \in [0,1]^{|\mathcal{S}_i|} | \mathbf{1}^\top \mathbf{p}^{(i)} = 1\}$, where $\mathcal{S}_i = (\{i\} \times \mathcal{V}_B) \cap \mathcal{Q}$. As a result, we can write $\chi_{\mathcal{R}}(P)$ as the sum of the corresponding characteristic functions. Similarly, $\chi_{\mathcal{C}}(P)$ can be written as the sum of the characteristic functions for the sets $\mathcal{C}^{(j)} = \{\mathbf{p}^{(j)} \in [0,1]^{|\mathcal{S}_j|} | \mathbf{1}^\top \mathbf{p}^{(j)} = 1\}$, corresponding to each column $j \in [n]$ of P. $\square$

Under this characterization, Prob. 5.4 becomes:

$$\min_{\mathbf{P}\in\mathbb{W}^n} \Big[\big(\sum_{(i,j)\in\mathcal{I}} |f_{ij}(\mathbf{P}_{\mathcal{S}_{ij}})|^p \big)^{\frac{1}{p}} + \sum_{(i,j)\in\mathcal{Q}} p_{ij}d_{ij} \tag{5.13a}$$

$$+ \sum_{i\in[n]} \chi_{\mathcal{R}^{(i)}}(\mathbf{P}_{\mathcal{S}_i}) + \sum_{j\in[n]} \chi_{\mathcal{C}^{(j)}}(\mathbf{P}_{\mathcal{S}_j}) \Big]. \tag{5.13b}$$

The first term in (5.13a) (i.e., (5.10a)) *cannot* be written as a sum of functions, except when $p = 1$. Hence, it is not immediately obvious how to parallelize ADMM when $p \neq 1$. For $p = 1$, however, the entire objective can be written as a sum of constituent "local" objectives; hence, in this case, algorithm (5.8) can be directly parallelized. In all other cases however, we need a specialized implementation to parallelize the optimization of the term (5.10a).

The application of ADMM (5.8) to all the terms in Prob. (5.13) is summarized Alg. 8; primal-dual variable pairs:

$$(\mathbf{p}_{ij}, \mathbf{y}_{ij})_{(i,j)\in\mathcal{I}},\ (q_{ij}, \xi_{ij})_{(i,j)\in\mathcal{Q}},\ (\mathbf{r}_i, \psi_i)_{i\in[n]},\ (\mathbf{c}_j, \phi_j)_{j\in[n]},$$

correspond to terms (5.10a)-(5.10d), respectively. We note that Alg. 8 requires special care to handle term (5.10a) in the case $p > 1$; we describe how to address this case in the next two subsections.

Algorithm 8 Outer ADMM

1: **Input:** $A, B \in \{0,1\}^{n\times n}$, $D = D_{A,B} \in \mathbb{R}_+^{n\times n}$, $\mathcal{Q} \subseteq [n]\times[n]$
2: **Local primal & dual variables at processors in $V_{\texttt{obj}}$:**
$(\mathbf{p}_{ij}, \mathbf{y}_{ij})_{(i,j)\in\mathcal{I}}$, $(q_{ij}, \xi_{ij})_{(i,j)\in\mathcal{Q}}$, $(\mathbf{r}_i, \psi_i)_{i\in[n]}$, $(\mathbf{c}_j, \phi_j)_{j\in[n]}$
3: **Consensus variables at processors in $V_{\texttt{var}}$:** $Z = [z_{ij}]_{(i,j)\in\mathcal{Q}}$
4: Initialize consensus variables and local/dual variables to 0;
5: Send copies of consensus variables z_{ij} to processors in $V_{\texttt{obj}}$
6: **while** not converged **do**
7:     **if** $p = 1$ **then**
8:         **for all** $(i,j) \in \mathcal{I}$ **in parallel do**
9:             $\mathbf{p}_{ij} \leftarrow \underset{\mathbf{p}_{ij}\in\mathbb{R}^{|\mathcal{S}_{ij}|}}{\arg\min} \left(|f_{ij}(p_{ij})| + \frac{\rho}{2}\|\mathbf{p}_{ij} - \mathbf{Z}_{\mathcal{S}_{ij}} + \mathbf{y}_{ij}\|_2^2\right)$
10:         **end for**
11:     **else if** $p > 1$ **then**
12:         Compute $\mathbf{p}_{ij}$, $(i,j) \in \mathcal{I}$, by solving (5.14) via Alg. 9
13:     **end if**
14:     **for all** $(i,j) \in \mathcal{Q}$ **in parallel do**
15:         $q_{ij} \leftarrow \arg\min_{q_{ij}\in\mathbb{R}} \left(\lambda \cdot q_{ij} d_{ij} + (q_{ij} - z_{ij} + \xi_{ij})^2\right)$
16:     **end for**
17:     **for all** rows $i \in [n]$ and **all** columns $j \in [n]$ **in parallel do**
18:         $\mathbf{r}_i \leftarrow \arg\min_{\mathbf{r}_i\in\mathbb{R}^{|\mathcal{S}_i|}} \left(\chi_{\mathcal{R}^{(i)}}(\mathbf{r}_i) + \frac{\rho}{2}\|\mathbf{r}_i - \mathbf{Z}_{\mathcal{S}_i} + \psi_{ij}\|_2^2\right)$
19:         $\mathbf{c}_j \leftarrow \arg\min_{\mathbf{c}_j\in\mathbb{R}^{|\mathcal{S}_j|}} \left(\chi_{\mathcal{C}^{(j)}}(\mathbf{c}_j) + \frac{\rho}{2}\|\mathbf{c}_j - \mathbf{Z}_{\mathcal{S}_j} + \phi_{ij}\|_2^2\right)$
20:     **end for**
21:     Send local variables to processors in $V_{\texttt{var}}$
22:     Update z_{ij}, $(i,j) \in \mathcal{Q}$, via averaging (5.8b)
23:     Send copies of consensus variables z_{ij} to processors in $V_{\texttt{obj}}$
24:     Update all dual variables via (5.8c)
25: **end while**
26: **return** consensus variables Z

5.3.1 Distributing Consensus ADMM for $p > 1$.

Applying consensus ADMM directly on (5.9) stumbles on the fact that the first term in the objective cannot be written as a sum; although the "local" optimization step (5.8a) of ADMM can be parallelized for all other terms, (5.8a) for this term (i.e., Line 12 of Alg. 8) takes the following form:

$$\min_{\mathbf{p}_{ij}, i,j\in[n]} \Big(\sum_{(i,j)\in\mathcal{I}} |f_{ij}(\mathbf{p}_{ij})|^p\Big)^{\frac{1}{p}} + \frac{\rho}{2} \sum_{(i,j)\in\mathcal{I}} \|\mathbf{p}_{ij} - \mathbf{Z}^k_{\mathcal{S}_{ij}} + \mathbf{y}^k_{ij}\|_2^2 \tag{5.14}$$

where $\mathbf{p}_{ij} \in \mathbb{R}^{|\mathcal{S}_{ij}|}$ is the local vector containing coefficients corresponding to $\mathbf{Z}_{\mathcal{S}_{ij}}$ and $\mathbf{y}^k_{ij} \in \mathbb{R}^{|\mathcal{S}_{ij}|}$

Algorithm 9 Inner ADMM

1: **Input:** $\{\bar{z}_{ij} : (i,j) \in \mathcal{I}\}$
2: **Local primal & dual variables at $|\mathcal{I}|$ processors in $V_{\texttt{obj}}$:**
$(\mathbf{p}_{ij}, u_{ij}, v_{ij})_{(i,j)\in\mathcal{I}}$,
3: Initialize $\mathbf{p}_{ij}$ to their previous values at the outer iteration, and dual variables v_{ij} to 0
4: **while** not converged **do**
5: Compute $\mathbf{u}$ by solving (5.16a) via Alg. 10
6: **for all** $(i,j) \in \mathcal{I}$ **in parallel do**
7: $\mathbf{p}_{ij} \leftarrow \arg\min_{\mathbf{p}_{ij}\in\mathbb{R}^{|\mathcal{S}_{ij}|}} \big(\frac{\rho}{2}\|\mathbf{p}_{ij}-\bar{\mathbf{z}}_{ij}\|_2^2 + \frac{\rho'}{2}(u_{ij}-f_{ij}(\mathbf{p}_{ij}) + v_{ij})^2\big)$
8: Update dual variable v_{ij} via (5.16c).
9: **end for**
10: **end while**
11: **return** consensus variables Z

is the dual variable corresponding to $\mathbf{p}_{ij} = \mathbf{Z}_{\mathcal{S}_{ij}}$. We rewrite this as:

$$\text{Minimize: } \|\mathbf{u}\|_p + \frac{\rho}{2}\textstyle\sum_{(i,j)\in\mathcal{I}} \|\mathbf{p}_{ij} - \bar{\mathbf{z}}_{ij}\|_2^2 \tag{5.15a}$$

$$\text{subj. to: } u_{ij} = f_{ij}(\mathbf{p}_{ij}), \text{ for } (i,j) \in \mathcal{I}, \tag{5.15b}$$

where $\mathbf{u} = [u_{ij}]_{(i,j)\in\mathcal{I}} \in \mathbb{R}^{|\mathcal{I}|}$ is a vector of auxiliary variables corresponding to the the affine terms $f_{ij}(\mathbf{p}_{ij})$, and $\bar{\mathbf{z}}_{ij} \equiv \mathbf{Z}^k_{\mathcal{S}_{ij}} - \mathbf{y}^k_{ij} \in \mathbb{R}^{|\mathcal{S}_{ij}|}$, for $(i,j) \in \mathcal{I}$. As $f_{ij}(\cdot)$ are affine functions, so are the set of constraints. Then, we can also solve (5.15) w.r.t. $\mathbf{u}$ and $\mathbf{p}_{ij}$ via ADMM, where the steps are

$$\mathbf{u}^k = \arg\min_{\mathbf{u}\in\mathbb{R}^{|\mathcal{I}|}} \|\mathbf{u}\|_p + \frac{\rho'}{2}\sum_{(i,j)\in\mathcal{I}} (u_{ij} - f_{ij}(\mathbf{p}^k_{ij}) + v^k_{ij})^2 \tag{5.16a}$$

$$\mathbf{p}^{k+1}_{ij} = \arg\min_{\mathbf{p}_{ij}\in\mathbb{R}^{|\mathcal{S}_{ij}|}} \frac{\rho}{2}\|\mathbf{p}_{ij} - \bar{\mathbf{z}}_{ij}\|_2^2 + \frac{\rho'}{2}(u^{k+1}_{ij} - f_{ij}(\mathbf{p}_{ij}) + v^k_{ij})^2 \tag{5.16b}$$

$$v^{k+1}_{ij} = v^k_{ij} + (u^{k+1}_{ij} - f_{ij}(\mathbf{p}^{k+1}_{ij})) \quad (i,j) \in \mathcal{I}, \tag{5.16c}$$

where $\rho' > 0$ is a tuning parameter and $v_{ij} \in \mathbb{R}$, $i,j \in [n]$, are the dual variables corresponding to linear constraints (5.15b). Step (5.16b) comprises $|\mathcal{I}|$ quadratic problems, while (5.16c) is a simple adaptation; both can be executed *in parallel across the $|\mathcal{I}|$ processors* that store $\mathbf{p}_{ij}$, $\mathbf{y}_{ij}$, and which have received $\mathbf{Z}_{\mathcal{S}_{ij}}$ from the (outer) consensus ADMM step (line 23 of Alg. 8). In contrast, it is not apriori clear how to parallelize step (5.16a); as in the case of the outer ADMM, this is due to the $\|\cdot\|_p$ term: we present our algorithm solving (5.16a) in parallel (Alg. 10) next.

The pseudocode for this inner ADMM step is presented in Alg. 9. The code is executed in parallel across the $|\mathcal{I}|$ machines described above. Note that steps (5.16b) and (5.16c) are executed

Algorithm 10 p-norm Prox. Operator

1: **Input:** $\mathbf{w} \in \mathbb{R}^d$, $p \geq 1$, $\rho > 0$, $\varepsilon > 0$
2: Set $\hat{w}_i \leftarrow \rho|w_i|$ for $i = 1, \ldots, d$.
3: **if** $\|\hat{\mathbf{w}}\|_q \leq 1$ **then**
4:     **return** $\mathbf{u}^* \leftarrow \mathbf{0}$
5: **end if**
6: Set $\mathbf{u} \leftarrow \mathbf{0}$, $s_L \leftarrow 0$, and $s_U \leftarrow \|\hat{\mathbf{w}}\|_p$
7: **for** $k = 1, \ldots, \log_2 \left\lceil \frac{1}{\varepsilon} \right\rceil$ **do**
8:     Set $s \leftarrow (s_L + s_U)/2$
9:     Compute $u_i \leftarrow \hat{w}_i g\left(s \cdot (\hat{w}_i)^{\frac{2-p}{p-1}}\right)$ for all $i \in \text{supp}(\hat{\mathbf{w}})$;
10:     Compute $\|\mathbf{u}\|_p$;
11:     **if** $\|\mathbf{u}\|_p < s$ **then**
12:         Set $s_U \leftarrow s$
13:     **else**
14:         Set $s_L \leftarrow s$
15:     **end if**
16: **end for**
17: Set $u_i^* \leftarrow \frac{\text{sign}\, w_i}{\rho} u_i$ for $i = 1, \ldots, d$.
18: **return** $\mathbf{u}^*$

in parallel but require no communication; hence, all communication in Alg. 9 is the one needed by Alg. 10 to compute $\mathbf{u}$; as we discuss in the next section, this amounts to a logarithmic number of map and reduce operations.

5.3.2 Parallel p-norm Proximal Operator

For $p > 1$, motivated by (5.16a), we consider the problem:

$$\min_{\mathbf{u} \in \mathbb{R}^d} \|\mathbf{u}\|_p + \frac{\rho}{2}\|\mathbf{u} - \mathbf{w}\|_2^2, \tag{5.17}$$

for a given $\mathbf{w} \in \mathbb{R}^d$ where $d \equiv |\mathcal{I}|$. In doing so, we assume that, as is the case in (5.16a), the elements of vector $\mathbf{w}$ are distributed across d machines, that need to collectively solve (5.17) in parallel. Following Liu and Ye [68], we define first a non-negative vector $\hat{\mathbf{w}}$ via

$$\hat{w}_i = \rho|w_i|. \tag{5.18}$$

We then consider the following simpler problem:

$$\min_{\mathbf{u} \in \mathbb{R}_+^d} \|\mathbf{u}\|_p + \frac{1}{2}\|\mathbf{u} - \hat{\mathbf{w}}\|_2^2. \tag{5.19}$$

Note that this differs from Prob. (5.17) in that (a) $\rho = 1$ and (b) vector $\mathbf{w} \in \mathbb{R}^d$ replaced with non-negative vector $\mathbf{w} \in \mathbb{R}_+^d$, and (c) optimization happens over $\mathbf{u} \in \mathbb{R}_+^d$. Nevertheless, Prob. (5.17) is equivalent to Prob. (5.19) (see Lemma 5.3.7 below). In particular, if $\hat{\mathbf{u}}$ is the optimal solution of (5.19), the optimal solution to (5.17) is given by $\mathbf{u}^*$ such that:

$$u_i^* = \frac{\text{sign}(w_i)}{\rho}\hat{u}_i, \quad \text{for } i \in [d]. \tag{5.20}$$

We therefore turn our attention to solving Prob. (5.19). To do so, we define first an auxiliary function. Given $\alpha \in (0, \infty)$, define the function $\alpha \mapsto g(\alpha)$, as the unique solution of the following equation over $x \geq 0$:

$$(x/\alpha)^{p-1} + x - 1 = 0, \tag{5.21}$$

We extend g to $[0, \infty)$ by setting $g(0) \equiv 0$ for $\alpha = 0$, by definition. Function g is hard to express in closed form,[2] but it is well-defined. This is because, for $\alpha > 0$, the l.h.s is -1 for $x = 0$ and positive for $x = \min(1, \alpha)$. Hence, by the intermediate value theorem, Eq. (5.21) always has a positive solution between 0 and $\min(1, a)$; uniqueness is implied by the strict monotonicity of the l.h.s. of Eq. (5.21) in x. Hence, $g : \mathbb{R}_+ \to [0, 1]$ is indeed well-defined.

Having defined g, given a vector $\hat{\mathbf{w}} \in \mathbb{R}_+^d$, we define functions $g_i : \mathbb{R}_+ \to \mathbb{R}_+, i \in [d]$ as:

$$g_i(s) = \hat{w}_i \cdot g\big(s \cdot (\hat{w}_i)^{\frac{2-p}{p-1}}\big), \tag{5.22}$$

as well as function $h : \mathbb{R}_+ \to \mathbb{R}$ as:

$$h(s) = \big(\sum_{i=1}^d g_i(s)^p\big)^{\frac{1}{p}} - s. \tag{5.23}$$

The optimal solution to Prob. (5.19) can be determined w.r.t. a root of equation $h(s) = 0$. In particular, the following holds, which we prove in Sec. 5.3.3:

Theorem 5.3.2 (Liu and Ye [68])**.** *Given $\hat{\mathbf{w}} \in \mathbb{R}_+^d$ and $p > 1$, let $\hat{\mathbf{u}} \in \mathbb{R}_+^d$ be an optimal solution to Prob.* (5.19)*. Let also $q \in \mathbb{R}_+$ be such that $\frac{1}{p} + \frac{1}{q} = 1$. Then, $\hat{\mathbf{u}}$ is unique, and:*

- *If $\|\hat{\mathbf{w}}\|_q \leq 1$, then $\hat{\mathbf{u}} = \mathbf{0}$.*
- *If $\|\hat{\mathbf{w}}\|_q > 1$, then*

$$\hat{u}_i = g_i(s^*), \quad \textit{for } i \in [d], \tag{5.24}$$

where s^* is the unique value in $(0, \|\hat{\mathbf{w}}\|_p]$ s.t. $h(s^*) = 0$.

[2] Though its inverse g^{-1} is easy to describe explicitly; see Eq. (5.28).

Intuitively, the above theorem suggests that there are two cases we need to consider. The first, "easy" case, is when $\|\hat{\mathbf{w}}\|_q \leq 1$: then, the optimal solution is $\mathbf{0}$. If $\|\hat{\mathbf{w}}\|_q > 1$, i.e., on the "hard case", solving Prob. (5.19) is tantamount to finding the unique, scalar root $s^* \in (0, \|\hat{\mathbf{w}}\|_p]$ of the equation:

$$h(s) = 0, \text{ where } h \text{ is given by Eq. (5.23).}$$

This is because, once this root s^* is computed, the optimal solution $\hat{\mathbf{u}} \in \mathbb{R}^d$ can be constructed via Eq. (5.24), by computing $g_i(s^*)$ for every $i \in [d]$.

Crucially, a root of h can be found with a simple bisection algorithm, *the steps of which can be easily parallelized via map and reduce operations.* This bisection algorithm, summarized in Alg. 10, proceeds as follows: given $\hat{\mathbf{w}} \in \mathbb{R}^d_+$, we test whether the condition $\|\hat{\mathbf{w}}\|_q \leq 1$ holds; if so, we return $\mathbf{u}^* = \mathbf{0}$. Otherwise, we find s^* via bisecting $[0, \|\hat{\mathbf{w}}\|_p]$. That is, at each iteration, we maintain an upper (s_U) and lower (s_L) bound on s^*, initialized at the above values. By construction, function h alternates signs on each of the two bounds: i.e., $h(s_L)h(s_U) \leq 0$; at each iteration, we (a) compute the average $s = 0.5(s_L + s_U)$, between the two bounds, (b) find the sign of h on this average, and then (c) update the bounds accordingly. As signs alternate, by the intermediate value theorem, s^* is guaranteed to be in $[s_L, s_U]$ at all times.

Another way to get some intuition behind how Alg. 10 behaves in the "hard" case is the following. At any iteration, s is compared to p-norm of the current solution $\mathbf{u} \in \mathbb{R}^d$. If $\|\mathbf{u}\| < s$, then s is too big, and we search at a smaller value; if the opposite is true, we search for a larger value, always adjusting the bounds accordingly. At all times, we set $\mathbf{u}$ by following the trajectory in $\mathbb{R}^d$ determined by functions g_i, linking the current s to the $\mathbf{u}$.

Since Alg. 10 ensures that the root s^* is be within $[s_L, s_U]$ at all times; Liu and Ye show that the algorithm thus approximate s^* within ε accuracy by performing $\log_2 \varepsilon^{-1}$ bisections [68]. We show this implies the following convergence guarantee (see Sec. 5.3.4 for proof):

Theorem 5.3.3. *Alg. 10 outputs a solution* $\mathbf{u} \in \mathbb{R}^d$ *such that* $\|\mathbf{u} - \hat{\mathbf{u}}\|_p \leq \sqrt[p-1]{\|\hat{\mathbf{w}}\|_q} \cdot \|\hat{\mathbf{w}}\|_p \cdot \varepsilon$.

Hence, Alg. 10 can approximate the optimal solution within arbitrary accuracy within a logarithmic number of iterations. Finally, it is easy to see that the computations involved in Alg. 10 can be parallelized across the d processors that store the values w_i, $i \in [d]$. Given an s, the computation of values $\hat{u}_i$ can happen in parallel via an application of Eq. (5.24) at each $\hat{w}_i$ (Line 9). Moreover, the p-norm of u (Line 10), needed to compute the sign of $h(s)$, can be computed via a reduce; the updated value of s can subsequently be broadcast to processors, to initiate the next

iteration. We further elaborate on parallelism in Section 5.4, where we discuss the computation and communication complexity of the entire process, combining Algorithms 8,9, and 10.

5.3.3 Proof of Theorem 5.3.2

The objective of Prob. (5.19) is strongly convex. Hence, the optimal solution $\hat{\mathbf{u}} \in \mathbb{R}_+^d$ is unique. We first show that $\hat{\mathbf{u}}$ lies, coordinate-wise, between $\mathbf{0}$ and $\hat{\mathbf{w}} \in \mathbb{R}_+^d$:

Lemma 5.3.4. *Let $\hat{\mathbf{u}}$ be the optimal solution of Prob.* (5.19). *Then $\hat{u}_i \in [0, \hat{w}_i]$, for all $i \in [d]$.*

Proof. Suppose that $\hat{\mathbf{u}}$ is an optimal solution of (5.19), s.t., $\hat{u}_i < 0$ for some $i \in [d]$. Then, for the vector $\hat{\mathbf{u}}'$ with all elements equal to the elements $\hat{\mathbf{u}}$ except the i-th element with $\hat{u}'_i = 0$ we have the following $\|\hat{\mathbf{u}}'\|_p + \frac{1}{2}\|\hat{\mathbf{u}}' - \hat{\mathbf{w}}\|_2^2 < \|\hat{\mathbf{u}}\|_p + \frac{1}{2}\|\hat{\mathbf{u}} - \hat{\mathbf{w}}\|_2^2$, a contradiction. Similarly, if $\hat{u}_i > \hat{\mathbf{w}}_i$ for some $i \in [d]$, we can construct a vector $\hat{\mathbf{u}}'$, s.t., all of its elements are the same with the elements of $\hat{\mathbf{u}}$, except $\hat{u}'_i = \hat{\mathbf{w}}_i$, then again $\|\hat{\mathbf{u}}'\|_p + \frac{1}{2}\|\hat{\mathbf{u}}' - \hat{\mathbf{w}}\|_2^2 < \|\hat{\mathbf{u}}\|_p + \frac{1}{2}\|\hat{\mathbf{u}} - \hat{\mathbf{w}}\|_2^2$, a contradiction. □

The lemma implies that we only need to look for a solution in a bounded domain. Note also that, as an immediate implication of Lemma 5.3.4, if $\hat{w}_i = 0$, then necessarily also $\hat{u}_i = 0$.

The optimal solution $\hat{\mathbf{u}}$ satisfies the KKT conditions:

$$\mathbf{0} \in \{\mathbf{g} + \mathbf{u} - \hat{\mathbf{w}} - \alpha | \mathbf{g} \in \partial f_p(\mathbf{u})\}, \tag{5.25a}$$

$$\alpha_i u_i = 0 \quad \text{for all } i \in [d], \quad \mathbf{u} \geq 0, \quad \alpha \geq 0, \tag{5.25b}$$

where $\partial f_p(\mathbf{u})$ is the subdifferential[3] of the function $f_p(\mathbf{u}) = \|\mathbf{u}\|_p$ at the point $\mathbf{u}$. Eq. (5.25a) implies a condition on $\hat{\mathbf{w}}$ under which the optimal solution to (5.19) is the zero vector:

Lemma 5.3.5. *Given $p \geq 1$, let $q \geq 1$ be such that $\frac{1}{p} + \frac{1}{q} = 1$. If $\hat{\mathbf{w}} \in \mathbb{R}_+^d$ satisfies $\|\hat{\mathbf{w}}\|_q \leq 1$, the optimal solution to Prob.* (5.19) *is $\hat{\mathbf{u}} = \mathbf{0}$.*

Proof. We show that $\mathbf{u}^* = \mathbf{0}, \alpha^* = \mathbf{0}$ satisfy the KKT conditions (5.25). For $\mathbf{u}^* = 0, \alpha^* = 0$, Eq. (5.25b) is obviously satisfied. Then we need to show that $0 \in \{g - \mathbf{w} | g \in \partial f_p(\mathbf{0})\}$, or equivalently, $\mathbf{w} \in \partial f_p(\mathbf{0})$. Formally, the subdifferential at zero is the set $\partial f_p(\mathbf{0}) = \{g | g^\top \mathbf{u} \leq \|\mathbf{u}\|_p \; \forall \mathbf{u}\}$. By Holder's inequality, for every $\mathbf{u}$ we have $\mathbf{w}^\top \mathbf{u} \leq \sum_{i=1}^d |w_i||u_i| \leq \|\mathbf{w}\|_q \|\mathbf{u}\|_p \leq \|\mathbf{u}\|_p$, where the last inequality holds as $\|w\|_q \leq 1$. Hence, $\mathbf{w} \in \partial f_p(0)$, and $\mathbf{u}^* = \mathbf{0}$, $\alpha^* = \mathbf{0}$ satisfy the KKT conditions, so $\mathbf{0}$ is optimal. □

[3]Note that f_p is not differentiable at $\mathbf{u} = \mathbf{0}$, hence the need to refer to its subdifferential.

Intuitively, we prove this by verifying that if $\|\hat{\mathbf{w}}\|_q \leq 1$ then $\mathbf{u} = \alpha = \mathbf{0}$ satisfy the KKT conditions in Eq. (5.25). Lemma 5.3.5 therefore immediately implies the first ("easy") case of Theorem 5.3.2.

We therefore turn to the case where $\|\hat{\mathbf{w}}\|_q > 1$ (the "hard" case). For $\mathbf{u} \neq 0$, f_p is differentiable and its subdifferential is a singleton, i.e., $\partial f_p(\mathbf{u}) = \{\nabla f_p(\mathbf{u})\}$, where

$$\partial f_p(\mathbf{u})/\partial u_i = \left(\frac{u_i}{\|\mathbf{u}\|_p}\right)^{p-1},$$

for $i \in [d]$. Suppose that the i-th element of the optimal point $\hat{\mathbf{u}}$ is positive, i.e., $\hat{u}_i > 0$. Then, $\alpha_i = 0$ by Eq. (5.25b), and Eq. (5.25a) implies that $\hat{\mathbf{u}}$ satisfies the following equation:

$$\big(\hat{u}_i/\|\hat{\mathbf{u}}\|\big)^{p-1} + (\hat{u}_i - \hat{w}_i) = 0, \text{ for all } i \text{ s.t. } \hat{u}_i > 0. \tag{5.26}$$

The optimality condition (5.26) can equivalently be written as

$$\hat{u}_i = \hat{w}_i g\left(\|\hat{\mathbf{u}}\|_p (\hat{w}_i)^{\frac{2-p}{p-1}}\right), \quad \text{for all } i \text{ s.t. } \hat{u}_i > 0. \tag{5.27}$$

where $g : \mathbb{R}_+ \to \mathbb{R}_+$ is defined via Eq. (5.21). Recall that Lemma 5.3.4 implies that, if $i \notin \operatorname{supp}(\hat{\mathbf{w}})$, then necessarily $\hat{u}_i = 0$. We can therefore consider w.l.o.g. a vector $\hat{\mathbf{w}}$ for which $\operatorname{supp}(\hat{\mathbf{w}}) = [d]$; if not, we can compute the optimal solution by setting $\hat{u}_i = 0$ for $i \notin \operatorname{supp}(\hat{\mathbf{w}})$, and focus on what happens on the remainder of the coordinates, that have full support. Not surprisingly, the optimal solution in this case is characterized by Eq. (5.27). In particular, the following lemma holds:

Lemma 5.3.6. *Consider a* $\mathbf{w} \in \mathbb{R}^d_+$ *s.t. (a)* $\operatorname{supp}(\hat{\mathbf{w}}) = [d]$, *and (b)* $\|\mathbf{w}\|_q > 1$, *where* $\frac{1}{p} + \frac{1}{q} = 1$. *Let* $g_i : \mathbb{R} \to \mathbb{R}^d$, $i \in [d]$, *and* $h : \mathbb{R}_+ \to \mathbb{R}$ *be given by Eqs.* (5.22) *and* (5.23)*, respectively. Then, the unique solution* $\hat{\mathbf{u}}$ *to Prob.* (5.19) *is given by* $\hat{u}_i = g_i(s^*)$*, for all* $i \in [d]$, *where* s^* *is the unique value in* $(0, \|\mathbf{w}\|_p]$ *s.t.* $h(s^*) = 0$.

Proof. The inverse g^{-1} of function g is given by

$$g^{-1}(x) = x(1-x)^{-1/(p-1)}. \tag{5.28}$$

As g^{-1} is monotone and continuous in $[0, 1)$, we have that g is also monotone and continuous in $\mathbb{R}_+$. Hence, function h is continuous in the interval $[0, \|\hat{\mathbf{w}}\|_p]$. Given that, by the intermediate value theorem, $g(a) \in [0, 1]$, we have that $h(\|\hat{\mathbf{w}}\|_p) \leq 0$. Since, by definition, $g(0) = 0$, we also have that $h(0) = 0$. Moreover,

$$\frac{dh(0)}{ds} = \lim_{\delta\to 0} \frac{\left(\sum_{i=1}^d (g_i(\delta))^p\right)^{\frac{1}{p}} - \delta}{\delta} = \lim_{\delta\to 0} \left(\frac{\sum_{i=1}^d (g_i(\delta))^p}{\delta^p}\right)^{\frac{1}{p}} - 1.$$

By Taylor's theorem, the first-degree Taylor approximation of g_i at 0 is $g_i(\delta) = g_i'(0)\delta + o(\delta^2)$. The partial derivative $g_i'(0)$ is given by: $g_i'(0) = \hat{w}_i^{\frac{1}{p-1}} g'(0)$. Note that $g'(0) = \left(\frac{dg^{-1}(0)}{dx}\right)^{-1} = 1$, and, as a result,

$$g_i'(0) = \hat{w}_i^{\frac{1}{p-1}}. \tag{5.29}$$

Therefore, we have:

$$\frac{\partial h(0)}{\partial s} = \lim_{\delta \to 0} \Big(\sum_{i=1}^{d} (\hat{w}_i^{\frac{1}{p-1}} \delta + O(\delta^2))^p / \delta^p \Big)^{\frac{1}{p}} - 1 = \|\hat{\mathbf{w}}\|_q^{\frac{1}{p-1}} - 1 > 0,$$

as $\|\hat{\mathbf{w}}\|_q > 1$ for $q = \frac{p}{p-1}$ by the hypothesis of the theorem. Hence, $h(s)$ is positive in a neighborhood of 0. As h is continuous, and $h(\|\mathbf{w}\|_p) \leq 0$, by the intermediate value theorem h must contain an $s^* \in (0, \|\hat{\mathbf{w}}\|_p]$ s.t, $h(s^*) = 0$. Such a solution satisfies Eq. (5.27) for all $i \in [d]$ and, as such, it satisfies the KKT conditions of Prob. (5.19); therefore, it is an optimal solution. Strong convexity implies its uniqueness. ☐

Lemma 5.3.6, along with our observation on cases where $i \notin \text{supp}(\hat{\mathbf{w}})$, immediately imply Theorem 5.3.2. To see this, note first that for $i \notin \text{supp}(\hat{\mathbf{w}})$, $g_i(s) = 0$ for all $s \in \mathbb{R}_+$. Moreover, these 0 coordinates do not contribute to $\|\mathbf{u}\|_p$ or $\|\hat{\mathbf{w}}\|_p$; as such, they do not affect h and, thereby, the root s^*: the latter is fully determined by only elements in $\text{supp}(\hat{\mathbf{w}})$. Hence, Eq. (5.24) holds for all coordinates in $[d]$. ☐

5.3.4 Proof of Theorem 5.3.3

We begin by showing the equivalence of Problems (5.17) and (5.19):

Lemma 5.3.7. *Let $\hat{\mathbf{u}}$ be an optimal solution to Prob.* (5.19)*, where $\hat{\mathbf{w}}$ is given by Eq.* (5.18)*, then $\mathbf{u}^*$, given by Eq.* (5.20)*, is an optimal solution to Prob.* (5.17)*.*

Proof. We have that: $\min_{\mathbf{u}\in\mathbb{R}^d} \rho(\|\mathbf{u}\|_p + \frac{\rho}{2}\|\mathbf{u}-\mathbf{w}\|_2^2) = \min_{\mathbf{u}'\in\mathbb{R}^d} \|\mathbf{u}'\|_p + \frac{1}{2}\|\mathbf{u}'-\mathbf{w}'\|_2^2$ for $\mathbf{u}' = \rho\mathbf{u}$, $\mathbf{w}' = \rho\mathbf{w}$. One can show that the coordinates of the optimal solution to the latter problem will have the same sign as the coordinates of $\mathbf{w}$. Let $\odot : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ indicate the element-wise multiplication between two vectors, and $\text{sign} : \mathbb{R}^d \to \{-1, +1\}^d$ be the vector resulting from element-wise application of the sign operator. Then, under the transformation $\mathbf{v} = \text{sign}(\mathbf{w}) \odot \mathbf{u}' \in \mathbb{R}_+^d$, Prob. (5.17) is equivalent to: $\min_{\mathbf{v}\in\mathbb{R}_+^d} \|\text{sign}(w) \odot \mathbf{v}\|_p + \frac{1}{2}\big\|\text{sign}(\mathbf{w}) \odot \mathbf{v} - \text{sign}(\mathbf{w}) \odot \hat{\mathbf{w}}\big\|_2^2 = \min_{\mathbf{v}\in\mathbb{R}_+^d} \|\mathbf{v}\|_p + \frac{1}{2}\|\mathbf{v} - \hat{\mathbf{w}}\|_2^2$,as $\|e \odot y\|_p = \|y\|_p$ for all $e \in \{-1, 1\}^d$, $y \in \mathbb{R}^d$, and $p \geq 1$.

Hence, given an optimal solution $\hat{\mathbf{u}}$ to Prob. (5.19), the optimal solution to Prob. (5.17) is given by $\mathbf{u}^* = \frac{1}{\rho}\left(\mathrm{sign}(\mathbf{w}) \odot \hat{\mathbf{u}}\right)$. □

Armed with this result, we next show that Alg. 10 correctly bounds the root s^*, whenever the corresponding for-loop is executed:

Lemma 5.3.8. *If $\|\hat{\mathbf{w}}\|_q > 1$, at every iteration of Alg. 10, $s^* \in [s_L, s_U]$.*

Proof. By Lemma 5.3.6, there must exist a unique root of $h(s) = 0$ in $(0, \|\hat{\mathbf{w}}\|_p]$. This, along with the strict positivity of h in the vicinity of 0 when $\|\hat{\mathbf{w}}\|_q > 1$, implies that if $h(\|\hat{\mathbf{w}}\|_p) = 0$, all values $h(s)$ for $s \in (0, \|\hat{\mathbf{w}}\|_p)$ are strictly positive, and the bisection will repeatedly update the lower bound but never the upper bound. The lemma therefore holds. If, on the other hand $h(\|\hat{\mathbf{w}}\|_p]) < 0$, s^* must be in $(0, \|\hat{\mathbf{w}}\|_p)$ by Lemma 5.3.6, so the lemma holds for the first iteration. We can show, by induction on iterations, that $h(s_L) \geq 0$, with equality holding only if $s_L = 0$, and $h(s_U) < 0$. If $h(s_L) > 0$, the lemma follows from the intermediate value theorem. If $h(s_L) = 0$, the lemma again follows from the intermediate value theorem, and the fact that h is strictly positive in the vicinity of 0. □

Observe that the distance between the two bounds is halved at each iteration. Hence, at the last ($\log_2 \left\lceil \frac{1}{\varepsilon} \right\rceil$) iteration,

$$|s - s^*| \leq |s_L - s_U| \leq \|\hat{\mathbf{w}}\|_p \varepsilon. \tag{5.30}$$

On the other hand, functions $g_i : \mathbb{R}_+ \to \mathbb{R}_+$ are Lipschitz:

Lemma 5.3.9. *Each function $g_i : \mathbb{R}_+ \to \mathbb{R}_+$ is Lipschitz continuous with Lipschitz parameter $\hat{w}_i^{\frac{1}{p-1}}$.*

Proof. The inverse of g is given by Eq. (5.28), which is strictly increasing, differentiable, and convex in $[0, 1)$; the latter follows from the fact that the second derivative is non-negative for $p > 1$, $x \in [0, 1)$. Hence, g is strictly increasing, differentiable, and concave. Each function g_i, $i \in [d]$, consists of a composition of g with an affine function, and a multiplication with a non-negative scalar, so it is also concave. Hence, it is Lipschitz continuous with parameter given by $g_i'(0)$; the latter is characterized by Eq. (5.29). □

Lemma 5.3.9 immediately implies that, for $\mathbf{u}$ the output of the algorithm, and s the last estimate of the root: $\left(\sum_{i\in[d]}(u_i - \hat{u}_i)^p\right)^{1/p} \leq \left(\sum_{i\in[d]} \hat{w}_i^{\frac{p}{p-1}}\right)^{1/p} \cdot |s - s^*|$, and the theorem follows from Eq. (5.30), as $q = \frac{p}{p-1}$. □

5.4 Parallel Complexity

ADMM is a first-order method, and its convergence is $O(\frac{1}{k})$ [240]. All dual variable adaptations are linear in their input sizes and so are averaging operations involved in consensus variable computations; both are parallelized. All primal variable adaptations are convex optimization problems with self-concordant objectives, either unconstrained or linearly constrained; as such, they can generically be solved within accuracy ϵ by interior point methods in steps that are polylogarithmic in $1/\epsilon$, with each step being polynomial in the input size. In particular:

- When $p = 1$, updating $\mathbf{p}_{ij}$, $(i, j) \in \mathcal{I}$ involves solving a generalized lasso regression problem (Line 9 of Alg. 8), which can be solved using the algorithm proposed by Tibshirani [241]. Alternatively, the inner ADMM Eq. (5.16) can again be applied; Eq. (5.16a) then amounts to $|\mathcal{I}|$ soft-max operations (each at $O(n)$ cost for computing f_{ij}). All all trivially parallelizable via a `map`.
- The row and column updates (Lines 18 and 19 of Alg. 8) amount to orthogonal projections on the simplex; we use the strongly polynomial algorithm by Michelot [242], which has complexity $O(n \log n)$. There are a total of n (one per row/column) such operations, all of which can again be parallelized via a `map` applying Michelot's algorithm.
- The optimization of the trace term involves $|\mathcal{Q}|$ one-dimensional quadratic problems (Line 15 of Alg. 8), which have a closed form and can be computed in $O(1)$ time. Again, these operations can be parallelized via a `map`; in practice, however, we avoid these computations altogether by "completing the squares" and incorporating these terms along with the column and row projections, as adjustments to the vectors projected to the simplices.
- The update of $\mathbf{p}_{ij}$ in Alg. 9 is an unconstrainted convex quadratic program that has a closed form solution. Each of these $|\mathcal{I}|$ such operations is equivalent to solving a linear system that can be computed in $O(|S_{ij}|^{2.376}) = O(n^{2.376})$ time [243]; again, they can be parallelized over $|\mathcal{I}|$ processors via a `map`.
- The norm computations in Alg. 10 (Lines 3 and 10) depend on vector size $|\mathcal{I}|$ and can be parallelized via a `reduce`. Updates in Line 9 are $O(1)$ for each of the $|\mathcal{I}|$ coordinates;[4] this is parallelizable over $|\mathcal{I}|$ processors, via a `map`.

[4]Function g can be computed efficiently at an arbitrary accuracy as it is strictly monotone and g^{-1} has a closed form.

- There are a total of $|\mathcal{I}| + |\mathcal{Q}| + 2n$ outer and $|\mathcal{I}|$ inner dual adaptations, they are all $O(1)$ and parallelizable via a `map`.

- The consensus averaging step involves $|\mathcal{V}_{\texttt{var}}| = |\mathcal{Q}|$ scalar summations, adding in total of $|\mathcal{E}_{\mathcal{G}}|$ terms, where $\mathcal{G}(\mathcal{V}_{\texttt{obj}}, \mathcal{V}_{\texttt{var}}, \mathcal{E}_{\mathcal{G}})$ is the bipartite graph (illustrated in Fig. 5.2) induced by our problem. Parallelizing this involves message passing between the nodes storing all $\mathcal{V}_{\texttt{obj}}$ objectives and the $|\mathcal{Q}|$ processors storing the consensus values, with the total number of messages passed being $|\mathcal{E}_{\mathcal{G}}|$. Below, we establish bounds on all these quantities.

Putting everything together, assuming the number of iterations of the outer and inner ADMM are $k_1, k_2 \in \mathbb{N}$, respectively, and that the accuracy used in Alg. 10 is ε, the serial complexity of our algorithm is:

$$k_1 k_2 O\big(|\mathcal{I}|\big(n^{2.376} + \log_2 \frac{1}{\varepsilon}\big)\big) + k_1\big[O\big(n^2 \log n\big) + O(|\mathcal{E}_{\mathcal{G}}|)\big]. \tag{5.31}$$

Assuming access to $\max(|\mathcal{I}|, n, |\mathcal{Q}|)$ processors, each inner iteration (first term in Eq. (5.31)) can be fully implemented via a constant number of `map` and `reduce` operations over these processors, with maps involving operations of at most $O(n \log n)$ complexity, and reduces terminating within $O(\log |\mathcal{I}|)$ rounds. On the other hand, the consensus step (second step in Eq. (5.31)) can be done via message passing between the processors corresponding to nodes of graph $\mathcal{G}$. The parallel complexity of the algorithm depends on the size of set $\mathcal{E}_{\mathcal{G}}$. In particular, we would like to determine conditions under which $\mathcal{G}$ is sparse. We therefore turn our attention to bounding the sparsity of $\mathcal{G}$ of the problem input size.

5.4.1 Characterizing the Sparsity of $\mathcal{G}$

The induced bipartite graph $\mathcal{G}(\mathcal{V}_{\texttt{obj}}, \mathcal{V}_{\texttt{var}}, \mathcal{E}_{\mathcal{G}})$, as illustrated in Fig. 5.2, depends on the number of terms that appear in the problem objective (determining $V_{\texttt{obj}}$) as well as on the number of times each variable appears in each such term (determining $\mathcal{E}_{\mathcal{G}}$). We first bound the size of $\mathcal{V}_{\texttt{obj}}$:

Lemma 5.4.1. *Let $\mathbf{E} \in \{0,1\}^{n\times n}$ be the binary matrix whose support is $\mathcal{Q}$, and let $m_0 \equiv \|\mathbf{AE} + \mathbf{EB}\|_0$. Then, the summation inside the first term* (5.10a) *of objective* (5.9) *contains $|\mathcal{I}| \leq m_0$ terms; collectively, the remaining three terms* (5.10b)-(5.10d) *contain at most $|\mathcal{Q}| + 2n$ terms.*

Proof. The number of non-zero elements in the matrix $\mathbf{AP} - \mathbf{PB}$ is at most the number of the non-zero elements in $\mathbf{AP} + \mathbf{PB}$, where $\mathbf{P}$ is a matrix has the full support under constraints $\mathcal{G}$,

e.g., $\mathbf{P} = \mathbf{E}$. Each set $\mathcal{S}_{ij}$ defines the support of the (i,j)-th element of $\mathbf{AP} - \mathbf{PB}$; therefore, we conclude that the total number of non-empty sets $\mathcal{S}_{ij}$ is upper-bounded by m_0. For the terms (5.10c) and (5.10d), it is easy to see that we have $|\mathcal{V}_A| = n$ sets $\mathcal{S}_i$ and $|\mathcal{V}_B| = n$ sets $\mathcal{S}_j$, each corresponding to the nodes $i \in \mathcal{V}_A$ and $j \in \mathcal{V}_B$, respectively. □

Lemma 5.4.1 immediately implies that the bipartite graph $\mathcal{G}(\mathcal{V}_{\texttt{obj}}, \mathcal{V}_{\texttt{var}}, \mathcal{E}_\mathcal{G})$ satisfies:

$$|\mathcal{V}_{\texttt{obj}}| \leq m_0 + |\mathcal{Q}| + 2n \qquad \text{and} \qquad |\mathcal{V}_{\texttt{var}}| = |\mathcal{Q}|. \tag{5.32}$$

Our next lemma characterizes $|\mathcal{E}_\mathcal{G}|$:

Lemma 5.4.2. *The supports of $f_{ij}(\cdot), \chi_\mathcal{R}(\cdot), \chi_\mathcal{C}(\cdot)$ satisfy:*

$$\textstyle\sum_{(i,j)\in\mathcal{I}} |\mathcal{S}_{ij}| \leq \min(n|\mathcal{E}_A|, n|\mathcal{Q}|) + \min(n|\mathcal{E}_B|, n|\mathcal{Q}|), \tag{5.33a}$$

$$\textstyle\sum_{i\in[n]} |\mathcal{S}_i| \leq |\mathcal{Q}|, \;\; \sum_{j\in[n]} |\mathcal{S}_j| \leq |\mathcal{Q}|. \tag{5.33b}$$

The support of functions $f_{ij}(\cdot)$ is also bounded by:

$$\textstyle\sum_{i,j\in[n]} |\mathcal{S}_{ij}| \leq m_0 \left(\max(d_A, d_\mathcal{Q}) + \max(d_B, d_\mathcal{Q})\right), \tag{5.33c}$$

where d_A, d_B, and $d_\mathcal{Q}$ denote the maximum degrees of graphs G_A, G_B, and $G([n],[n],\mathcal{Q})$, respectively.

Proof. Let $\mathcal{S}_L^{(i,j)} = \{(k,j) \in \mathcal{Q}\} \cap \{(i,k) \in \mathcal{E}_A\}$ and $\mathcal{S}_R^{(i,j)} = \{(i,k) \in \mathcal{Q}\} \cap \{(k,j) \in \mathcal{E}_B\}$. Then $\sum_{i,j\in[n]} |\mathcal{S}_L^{(i,j)}| = \sum_{j\in[n]} \sum_{i\in[n]} |\mathcal{S}_L^{(i,j)}| \leq \sum_{j\in[n]} \min(|\mathcal{E}_A|, nd_j) \leq \min(n|\mathcal{E}_A|, \sum_{j\in[n]} nd_j) = \min(n|\mathcal{E}_A|, n|\mathcal{Q}|)$, where d_j is the node degree for $j \in \mathcal{V}_B$.

Similarly, we can show that $\sum_{i,j\in[n]} |\mathcal{S}_R^{(i,j)}| \leq \min(n|\mathcal{E}_B|, n|\mathcal{Q}|)$. As a result, $\sum_{i,j\in[n]} |\mathcal{S}_{ij}| \leq \sum_{i,j\in[n]} |\mathcal{S}_L^{(i,j)}| + |\mathcal{S}_R^{(i,j)}| \leq \min(n|\mathcal{E}_A|, n|\mathcal{Q}|) + \min(n|\mathcal{E}_B|, n|\mathcal{Q}|)$.

Now we prove (5.33c). For each set $\mathcal{S}_{ij}$ we have that: $|\mathcal{S}_{ij}| \leq |\mathcal{S}_L^{(i,j)}| + |\mathcal{S}_R^{(i,j)}| \leq \max(d_i, d_\mathcal{Q}) + \max(d_j, d_\mathcal{Q}) \leq \max(d_A, d_\mathcal{Q}) + \max(d_B, d_\mathcal{Q})$. From this and Lemma 5.4.1, which shows that $\mathcal{I} \leq m$, we get $\sum_{i,j\in[n]} |\mathcal{S}_{ij}| \leq m \max_{i,j}(|\mathcal{S}_{ij}|) \leq m(\max(d_A, d_\mathcal{Q}) + \max(d_B, d_\mathcal{Q}))$. For $\mathcal{S}_i, i \in [n]$ we have that: $\bigcap_i \mathcal{S}_i = \emptyset$, and $\bigcup_i \mathcal{S}_i = \{(k,j) \in \mathcal{Q} | j \in \mathcal{V}_B\} \subseteq \mathcal{Q}$. Therefore, for the total size we have: $\sum_i |\mathcal{S}_i| = |\cup_i \mathcal{S}_i| \leq |\mathcal{Q}|$.

□

Lemma 5.4.2 implies that the number of edges in $\mathcal{G}$ is:

$$|\mathcal{E}_\mathcal{G}| \leq M + 3|\mathcal{Q}|, \tag{5.34}$$

where M is the minimum among the bounds in (5.33a) and (5.33c). Hence, Eq. (5.32) and (5.34) together provide conditions under which when $\mathcal{G}$ is sparse. This happens if, e.g., $m_0 = O(n^2)$ and both G_A and G_B are sparse: by Eq. (5.33a), graph $\mathcal{G}$ would then have a number of edges that is $O(\mathcal{V}_{\texttt{obj}} + \mathcal{V}_{\texttt{var}})$. Alternatively, the same occurs when d_A, d_B, and d_Q are bounded (by Eq. (5.33c)).

5.5 Experiments

5.5.1 Experimental Setup

Execution environment. We run Spark on a local cluster that comprises 8 machines. Each machine has 2 Intel(R) Xeon(R) CPUs (E5-2680 v4) with 14 cores, and the cluster has $8 \times 28 = 224$ cores in total. We run OpenMP on the Google Cloud Platform[5] and on a `n1-standard-96` machine with 96 (virtual) cores and 360GB RAM.

Metrics. We report the objective as well as the primal and dual residuals as the iterations of our ADMM algorithm progress. The latter measure convergence. We evaluate the optimality of our solution by a parameter $\epsilon \in \mathbb{R}$ defined as: $\epsilon = \max\left(\frac{\|r^K\|_2}{\sqrt{|\mathcal{I}|}}, \frac{\|s^K\|_2}{\sqrt{|\mathcal{I}|}}\right)$, where r^K and s^K are the primal and dual residuals [13] at the last iteration . The smaller ϵ is, the closer the solution is to the optimal.

Datasets. We experiment on several real graphs from the Network Repository[6] and the Stanford Large Network Dataset Collection[7], which we summarize in Table 5.1. Four graphs `bnm1`, `bnm2`, `bnc1`, and `bnc2` are brain networks. `ptn1` and `ptn2` are biological networks, while `rt1` and `rt2` are re-tweet networks. The nodes in `dzr1` and `dzr2` are users of the music streaming service Deezer for two different countries, where edges show friendship. The graphs `sld1` and `sld2` represent social interactions between the users of the website Slashdot. We also experiment on synthetic Erdős Rényi, ER(n, q), graphs with n nodes and edge probability q, where n ranges from 2^6 to 2^{17}.

Preprocessing. For real graphs we use 4 node features: the size of the first-hop and second-hop neighborhoods, the number of paths of length 2, and their pagerank. For synthetic graphs, in addition to these 4 features, we also compute the number of paths and cycles of length 3 along with the size of the third-hop neighborhood. Given two graphs, we construct the dissimilarity matrix $\mathbf{D_{A,B}}$ using the ℓ_2 distance between features. We construct the constraint set $\mathcal{Q}$ for real graphs by one of the following methods, which we report along with the size of the set $\mathcal{Q}$ in Table 5.1.

[5] `https://cloud.google.com`
[6] `http://networkrepository.com`
[7] `https://snap.stanford.edu/data/index.html`

pair	(G_A, G_B)	$\lvert\mathcal{V}_A\rvert, \lvert\mathcal{V}_B\rvert$	$\lvert\mathcal{E}_A\rvert, \lvert\mathcal{E}_B\rvert$	$\mathcal{Q}$	$\lvert\mathcal{Q}\rvert$	$\lvert\mathcal{I}\rvert$
`cortex`	$(\texttt{bnc1}, \texttt{bnc2})$	91, 93	1.9K, 2.6K	`all`	8.6K	8.6K
`monkey`	$(\texttt{bnm1}, \texttt{bnm2})$	242, 91	4K, 628	`all`	58.5K	58.5K
`protein`	$(\texttt{ptn1}, \texttt{ptn2})$	1.5K, 1.8K	2K, 4K	`degree`	3.3M	3.3M
`retweet`	$(\texttt{rt1}, \texttt{rt2})$	3.2K, 3.2K	3.4K, 3.9K	`degree`	5.9M	2.7M
`deezer`	$(\texttt{dzr1}, \texttt{dzr2})$	41.8K, 47.5K	125.8, 222.8K	`WL2`	1.1M	2.9M
`slashdot`	$(\texttt{sld1}, \texttt{sld2})$	77K, 82K	828K, 870K	`WL3`	98K	2M

Table 5.1: A summary of real graph pairs along with the preprocessing method for generating $\mathcal{Q}$.

- `all`: $\mathcal{Q}$ includes all pairs $(i, j) \in \mathcal{V}^2$, i.e., $|\mathcal{Q}| = n^2$.
- `degree`: $\mathcal{Q}$ includes pairs $(i, j) \in \mathcal{V}^2$, s.t., the nodes i and j have the same node degree.
- `WL`k: $\mathcal{Q}$ includes pairs $(i, j) \in \mathcal{V}^2$, s.t. i, j have the same color: we generate node colors by running the Weisfeiler-Lehman (WL) algorithm [206] (see Sec. 5.1.1 and Sec. 2 in [1]) for k iterations.

For synthetic graphs we use `all` for generating $\mathcal{Q}$.

Implementation. We implemented Alg. 8 over Spark (version 2.3.2), an open-source cluster-computing framework [33], via its Python interface (version 2.7.15). We also implemented Alg. 8 in Ansi C (glibc version 2.23), using OpenMP (version 4.0) and Atlas (version 3.10.2).

5.5.2 Linear Term

We begin by studying the effects of the linear term in (5.4) on convergence. G_A is an ER$(64, 0.1)$ and G_B is a random permutation of G_A. We show the trace of residuals and the norm throughout iterations of ADMM for $\lambda = 0, 0.1, 10$ in Fig. 5.3. We run ADMM for each λ value for a fixed number of iterations (160). In all cases the optimal solution is the permutation matrix used to generate G_B from G_A, so the optimal objective value is 0. Note that in Fig. 5.3 all of the objective values as well as the residuals indeed converge to zero. We see that non-zero λ values significantly accelerate convergence, as the linear term directs the algorithm faster to the correct solution. The logarithm-scaled plot accentuates the faster linear convergence of ADMM for $\lambda = 0.1, 10$ in comparison with slower sub-linear convergence for $\lambda = 0$.

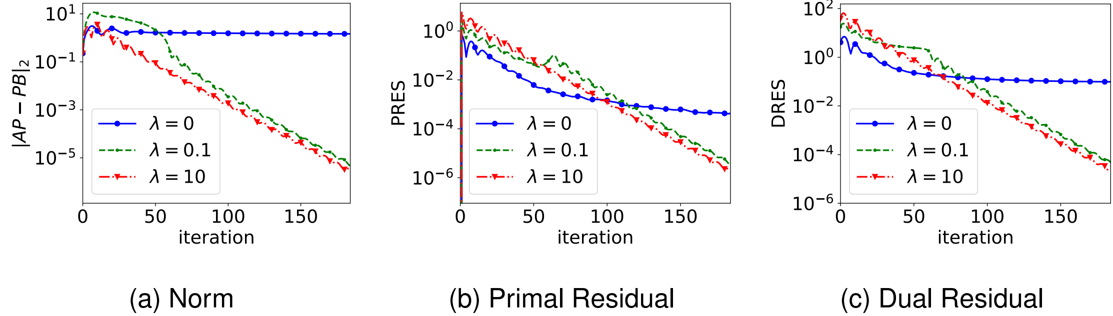


Figure 5.3: Effects of adding the linear term on the convergence. The plot shows the traces for the objective and the primal and dual residual throughout ADMM iterations (Alg. 8 for $p = 2$) for different λ. We observe that giving a larger coefficient to the linear term makes the convergence considerably faster.

5.5.3 p-norms

We next assess the quality of the computed doubly stochastic matrix $\mathbf{P}$ w.r.t. p-norms. We let again G_A be ER$(64, 0.1)$. We generate another graph G_B by adding different noise types to G_A according to the following scenarios:

- Bernoulli noise (BRN): We flip each element in $\mathbf{A}$ with probability 0.01.
- Outliers (OUTk): We choose k nodes (*outliers*) from G_A uniformly at random and make them connected to every other node in G_B. We experimented with $k = 1, 2$ outliers, (i.e, OUT1 and OUT2, respectively).
- Gaussian noise (GSS): We add i.i.d. zero-mean Gaussian noise with variance 0.01 to the elements of A.
- Laplacian noise (LPC): We add i.i.d. zero-mean Laplacian noise with variance 0.01 to the elements of A.
- Mixture of Gaussian and Laplacian noise (MIX): We add a mixture of i.i.d. zero-mean Gaussian and Laplacian random variables with 0.01 variance; the mixture coefficients are equally set to $\sqrt{0.5}$ so that the variance is 0.01.

We obtain the matrix $\mathbf{P}$ by solving (5.4) for different pair (p, λ) values. As G_B is a perturbed version of G_A, i.e., generated via the addition of noise, we measure the quality of the resulting solution by how far it deviates from the identity. As metrics, we report the Diagonal Probability Mass ($\mathbf{DPM}$) of $\mathbf{P}$, defined as the sum of the diagonal elements normalized by $n = 64$,

	BRN		OUT1		OUT2	
(p, λ)	DPM	DPMP	DPM	DPMP	DPM	DPMP
(1, 0.0)	**0.99**	**1.0**	0.32	1.0	0.08	0.98
(1, 0.1)	0.99	1.0	0.33	1.0	0.05	0.45
(1, 1.0)	0.97	0.97	0.33	1.0	0.05	0.45
(1.5, 0.0)	0.14	1.0	0.42	1.0	**0.24**	**0.99**
(1.5, 0.1)	0.27	0.98	**0.43**	**1.0**	0.23	0.98
(1.5, 1.0)	0.49	0.97	0.38	1.0	0.10	0.82
(2, 0.0)	0.12	1.0	0.35	1.0	0.18	0.98
(2, 0.1)	0.25	0.97	0.08	0.97	0.06	0.90
(2, 1.0)	0.3	0.95	0.04	0.45	0.06	0.91
(3, 0.0)	0.11	1.0	0.25	0.98	0.04	0.24
(3, 0.1)	0.29	0.97	0.01	0.03	0.01	0.05
(3, 1.0)	0.42	0.82	0.01	0.02	0.01	0.05
(5, 0.0)	0.097	0.98	0.3	1.0	0.04	0.24
(5, 0.1)	0.31	0.92	0.01	0.006	0.01	0.04
(5, 1.0)	0.36	0.58	0.01	0.01	0.01	0.04
(N/A, 1.0)	0.32	0.32	0.02	0.02	0.01	0.009

Table 5.2: Comparison of different p-norms and λ coefficients for the linear term for BRN, OUT1, and OUT2. When considering only the linear term, we denote p value by N/A and report the corresponding results in the last row.

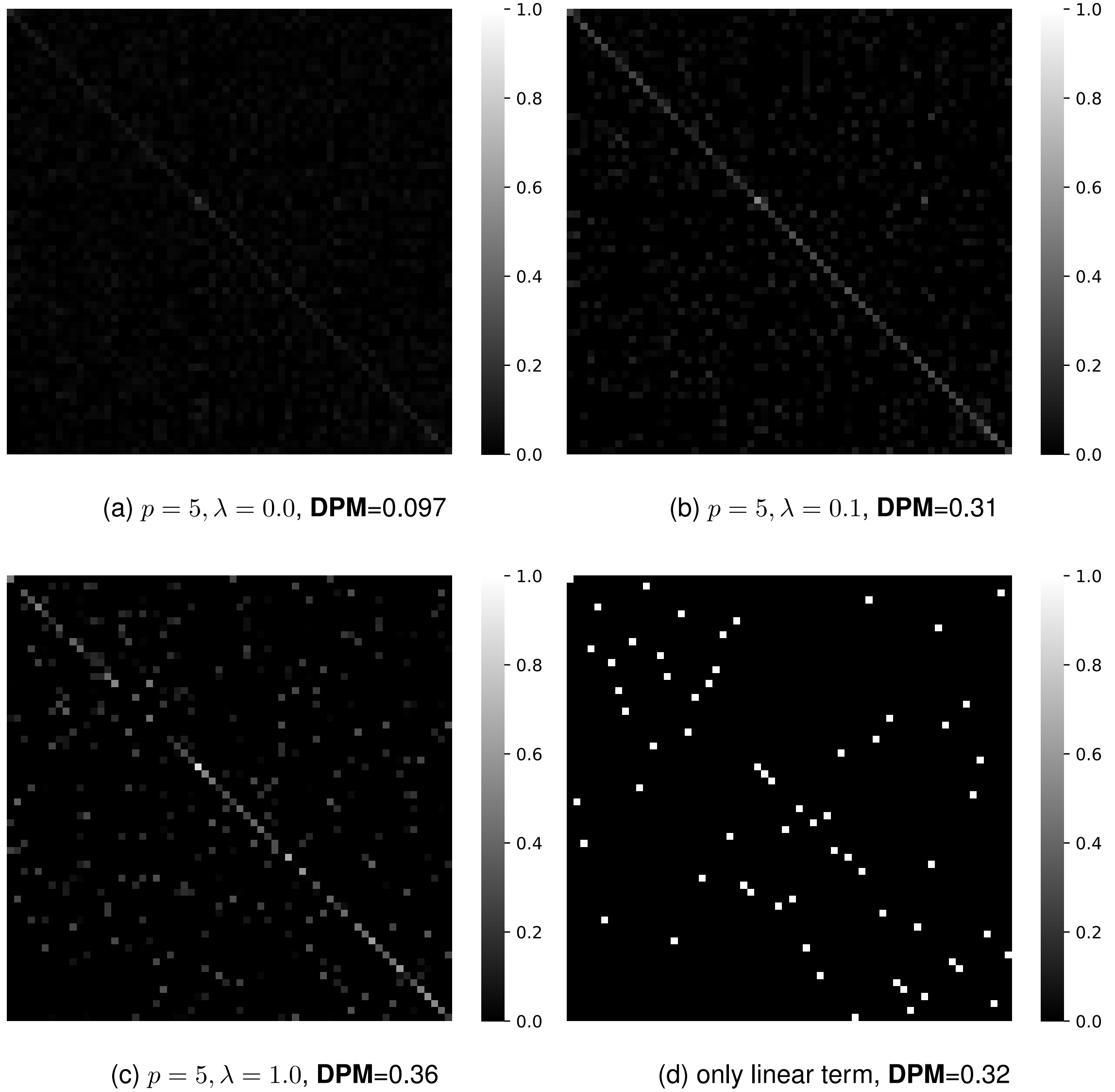
	GSS		LPC		MIX	
(p, λ)	DPM	DPMP	DPM	DPMP	DPM	DPMP
(1, 0.0)	0.12	0.99	0.08	0.87	0.16	1.0
(1.5, 0.0)	0.14	1.0	0.8	1.0	0.19	1.0
(2, 0.0)	0.15	1.0	0.09	1.0	0.20	1.0
(3, 0.0)	0.16	1.0	0.09	0.98	0.21	1.0
(5, 0.0)	**0.163**	**1.0**	**0.1**	**1.0**	**0.22**	**1.0**

Table 5.3: Comparison of different norms for GSS, LPC, and MIX. Here we do not report results for $\lambda \neq 0$, as the second graph is weighted and fully connected.

and the Diagonal Probability Mass after Projection ($\mathbf{DPMP}$) on the set of permutation matrices $\mathbb{P}^n$, i.e., $\mathbf{DPM} = \frac{1}{n}\sum_{i\in[n]} \mathbf{P}_{ii}$ and $\mathbf{DPMP} = \frac{1}{n}\sum_{i\in[n]} \mathbf{P}^{\pi}_{ii}$, where $\mathbf{P}^{\pi} = \arg\min_{\mathbf{P}'\in\mathbb{P}^n} \|\mathbf{P} - \mathbf{P}'\|_2^2$.

We report results for the first three cases with unweighted edges, i.e., BRN, OUT1, and OUT2 in Table 5.2. Note that for other cases, i.e., GSS, LPC, and MIX, the edges in G_B are weighted and thusly G_B is fully connected, so we do not add the linear term ($\lambda = 0$). We report results for the latter in Table 5.3. The reported results are averaged over 5 random runs; graphs G_A are generated independently at random for each run, then G_B is generated following the scenarios outlined above.

We make the following observations from Table 5.2. We first concentrate on BRN, reported in the first column of the table. We see that in the absence of the linear term ($\lambda = 0$), $p = 1$ has a superior performance and recovers the identity matrix. We further observe that for other p values



(a) $p = 5, \lambda = 0.0$, **DPM**=0.097

(b) $p = 5, \lambda = 0.1$, **DPM**=0.31

(c) $p = 5, \lambda = 1.0$, **DPM**=0.36

(d) only linear term, **DPM**=0.32

Figure 5.4: Effects of adding linear term on solutions for $p = 5$ and BRN noise. We assess that adding the linear term $\lambda = 0.1$ increases the values on the diagonal. However, increasing λ further to 1.0 makes solution highly biased on the extracted node features; this is obvious from the non-diagonal gray elements.

adding the linear term ($\lambda = 0.1, 1$) improves the solution. For instance, by comparing metrics for $\lambda = 0$ and $\lambda = 0.1$ in the first column (BRN case) we see that adding the linear term generally increases $\mathbf{DPM}$, while $\mathbf{DPMP}$ stays almost the same (above 0.9). However, increasing λ further to 1 decreases $\mathbf{DPMP}$ significantly. The reason is that increasing λ makes the solution highly biased on the extracted features. Motivated by this observation we also tested the case with only the linear

term, where p is denoted by N/A, we see that $\mathbf{DPM}$ and $\mathbf{DPMP}$ values are grossly inferior, in comparison to other cases. These observations suggest that there is a trade-off between the first norm term, and the second linear term in (5.4); the linear term can improve the solution by incorporating node features; however, using a high λ values makes the results highly dependent on the crafted node features.

To make this point more vivid we visualize solutions as heatmaps for $p = 5$ and different λ values and the case with only linear term in Fig. 5.4; from the figure we see that increasing λ from 0 to 0.1 increases the overall diagonal values (see Fig 5.4a and 5.4b). We see that increasing λ further to 1 slightly increases the diagonal mass but also increases non-diagonal values (see Fig. 5.4c); by comparing Fig. 5.4c and Fig. 5.4d we see that the non-diagonal elements in the former corresponds to the solution generated by adding the linear term (see the non-diagonal elements in Fig. 5.4c and Fig. 5.4d).

For the two types of outliers (OUT1 and OUT2) reported in next columns of Table 5.2, we see that $p = 1.5$ outperforms other p values. This is in contradiction to the previous case (BRN), where $p = 1$ outperformed other p values. We further observe that despite the case of BRN adding the linear term only deteriorates solutions. The reason is that adding outlier nodes adversely interfere with the extracted node features that are all dependent on degree and neighborhood information.

Finally we report results for other noise types with weighted edges, i.e., GSS, LPC, and MIX in Table 5.3. Here all p norms have comparable performances; they all achieve $\mathbf{DPMP} = 1$, but $\mathbf{DPM}$ is slightly higher for higher p values.

5.5.4 Scalability

We evaluate the scalability of our proposed ADMM algorithm w.r.t. the graph size n and the number of CPUs. We report the results for two different objectives $\|\mathbf{AP} - \mathbf{PB}\|_2^2$ and $\|\mathbf{AP} - \mathbf{PB}\|_2$ with $\lambda = 0$, in Tables 5.4 and 5.5, respectively. The two problems are mathematically equivalent. However, $\|\mathbf{AP} - \mathbf{PB}\|_2^2$ has a separable form, therefore, Alg. 8 skips the inner loop (Alg. 9); in this case, Line 9 in Alg. 8 is an unconstrained quadratic problem, which has a closed form solution. In particular, we report setup time $t_{\texttt{SU}}$ and iteration time $t_{\texttt{IT}}$. $t_{\texttt{SU}}$ includes the time spent creating and initializing the variables, i.e., Lines 2 to 4 in Alg. 8. $t_{\texttt{IT}}$ is the average iteration time of Alg. 8, i.e., Lines 7 to 24. ADMM is a first-order-method and it usually needs ≈100 iterations to converge. Therefore, the iteration time dominates the setup time, but for completeness we report setup times too. In this experiment G_A, G_B, and $\mathcal{Q}$ are Erdős Rényi graphs. We experiment with two settings,

	Dense						Sparse									
n	2^{10}		2^{11}		2^{12}		2^{13}		2^{14}		2^{15}		2^{16}		2^{17}	
$\lvert\mathcal{E}_A\rvert,\lvert\mathcal{E}_B\rvert$	5.2K,5.2K		20.9K,21K		84K,83.8K		49K,48.7K		103K,104K		220K,221K		464K,465K		980K,981K	
$\lvert\mathcal{Q}\rvert,\lvert\mathcal{I}\rvert$	10.4K,194.9K		42K,1.4M		168K,9.4M		48.7K,1.1M		103K,2.5M		219K,5.8M		465K,13M		979K,29M	
CPUs	t_{SU}(s)	t_{IT}(s)	t_{SU}(s)	t_{IT}(s)	t_{IT}(s)	t_{IT}(s)	t_{SU}(s)	t_{IT}(s)	t_{SU}(s)	t_{IT}(s)	t_{SU}(s)	t_{IT}(s)	t_{SU}(s)	t_{IT}(s)	t_{SU}(s)	t_{IT}(s)
OpenMP																
1	8	4	126	39	1076	412	△		-	-	-	-	-	-	-	-
10	10	0.4	145	4	1074	43	△		-	-	-	-	-	-	-	-
20	9	0.23	121	2	1045	19	△		-	-	-	-	-	-	-	-
30	9	0.17	121	1	953	12	△		-	-	-	-	-	-	-	-
Spark																
1	803	680	6095	9570	■		4173	2891	11155	6339	22183	15250	■		■	
10	288	88	639	1139	■		2147	357	6419	769	12009	1896	■		■	
20	257	56	370	688	■		2029	216	6208	463	11435	1175	■		■	
30	257	56	302	559	■		1984	177	5869	392	10762	975	■		■	
56	294	79	293	402	■		1963	154	6000	338	10713	910	■		■	
448	75	26	52	62	1123	437	290	34	806	67	1409	135	4200	250	10990	790

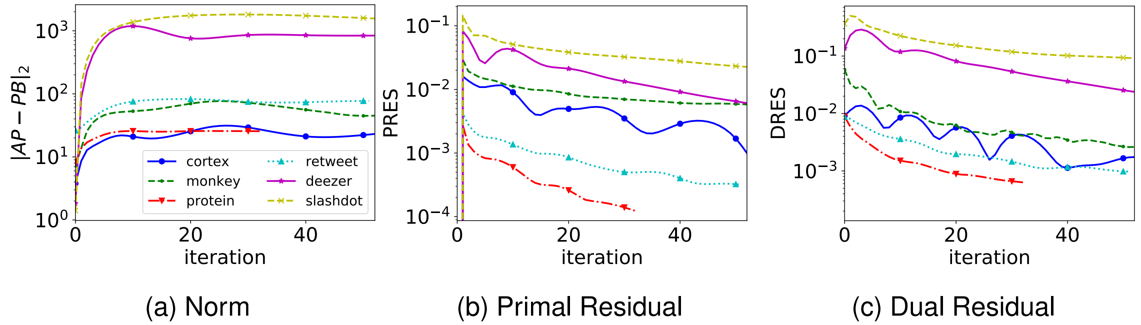
Table 5.4: Scaling results for Alg. 8 and $\|\mathbf{AP}-\mathbf{PB}\|_2^2$ objective ($\lambda=0$). In this case Alg. 8 skips the inner loop Alg. 9 as the objective is separable. We run our ADMM algorithms using both OpenMP and Spark implementations on the synthetic graphs. t_{IT} is the average over 5 iterations. We denote the cases that the execution ran out of memory by ■ and ones that produced a segmentation fault with △.

	Dense						Sparse									
n	2^{10}		2^{11}		2^{12}		2^{13}		2^{14}		2^{15}		2^{16}		2^{17}	
$\lvert\mathcal{E}_A\rvert,\lvert\mathcal{E}_B\rvert$	5.2K,5.2K		20.9K,21K		84K,83.8K		49K,48.7K		103K,104K		220K,221K		464K,465K		980K,981K	
$\lvert\mathcal{Q}\rvert,\lvert\mathcal{I}\rvert$	10.4K,194.9K		42K,1.4M		168K,9.4M		48.7K,1.1M		103K,2.5M		219K,5.8M		465K,13M		979K,29M	
CPUs	t_{SU}(s)	t_{IT}(s)	t_{SU}(s)	t_{IT}(s)	t_{SU}(s)	t_{IT}(s)	t_{SU}(s)	t_{IT}(s)	t_{SU}(s)	t_{IT}(s)	t_{SU}(s)	t_{IT}(s)	t_{SU}(s)	t_{IT}(s)	t_{SU}(s)	t_{IT}(s)
56	63	642	1566	11468	■		101	3040	218	6475	495	15359	■		■	
448	21	222	60	1490	584	5463	35	455	44	907	98	2070	196	4327	565	9121

Table 5.5: Scaling results for Alg. 8 and $\|\mathbf{AP}-\mathbf{PB}\|_2$ objective ($\lambda=0$). In comparison to Table 5.4, we see that in general the iteration time is longer because each iteration of Alg. 8 executes the inner ADMM loop Alg 9 (for 60 iterations).

i.e., (a) "dense" graphs ER$(n, 0.01)$ ($n=2^{10}$ to 2^{12}) and (b) "sparse" graphs ER$(n, 1.1\log n/n)$ for $n=2^{13}$ to 2^{17}.

For $\|\mathbf{AP}-\mathbf{PB}\|_2^2$ we use both the OpenMP (in C) and the Spark (in Python) implementations. For Spark, when running with 56 cores or less, we use a single machine out of the cluster. From Table 5.4 we see that for dense graphs OpenMP has excellent speedup; for example, we see that t_{IT} for 30 CPUs is 30× smaller than t_{IT} for 1 CPU, matching the level of parallelism. However, the Spark implementation is slower for these dense graphs. This is due to both the high-level programming language (Python) and the Spark overheads, e.g., the cost of communicating the consensus variables across machines, which is more considerable in the dense graphs. We report speedups for running over a Spark cluster with 480 CPUs based on Gustafoson's law [244] that computes speedup as follows, $s_{speedup} = 1-\gamma+\gamma s^{par}_{speedup}$, where $\gamma \in [0,1]$ is the portion of the serial program that can be



(a) Norm (b) Primal Residual (c) Dual Residual

Figure 5.5: Traces of our ADMM algorithm for $p = 2$ and real graphs pairs. For `cortex`, `retweet`, `deezer`, and `slashdot` pairs λ is set to 0.001, while for `monkey` and `protein` it is 0.1. We run the inner loop (Alg. 9) for 60 iterations. The average iteration time (of Alg. 8) for `cortex`, `monkey`, `protein`, `retweet`, `deezer`, and `slashdot` is 364(s), 712(s), 3836(s), 4375(s), 3797(s), and 2290(s), respectively.

parallelized (ADMM iterations in Alg. 8) and $s^{par}_{speedup}$ is the speedup for the portion of the program that benefits from parallelism. We compute γ as the ratio of running time for ADMM iterations (we consider 100 iterations) to the total running time for serial execution (1 CPU), i.e., $\gamma = \frac{100 \times t_{\text{IT}}}{100 \times t_{\text{IT}} + t_{\text{SU}}}$, where $t_{\text{IT}}, t_{\text{SU}}$ correspond to 1 CPU in Table 5.4. We compute $s^{par}_{speedup}$ by comparing t_{IT} values for 1 CPU and 480 CPUs. The speedups for Table 5.4 are 26, 153, 84, 92, and, 110, respectively for $n = 2^{10}, 2^{11}, 2^{13}, 2^{14}$, and 2^{15}.

As expected from Lemma 5.4.2, the Spark scales better for sparse graphs. For each n, by increasing the number of CPUs from 1 to 56, on a single machine, we see a consistent speedup in both t_{IT} and t_{SU}. Moreover, when running over cluster (448CPUs) it is 4.5, 5, and 6.7 times faster then a single machine (56CPUs) for $n = 2^{13}, 2^{14}$, and 2^{15}, respectively.

For $\|\mathbf{AP} - \mathbf{PB}\|_2$ we only report the results for Spark implementation in Table 5.5. By comparing the running times for 448 CPUs and 56 CPUs we see speedups of 2.89 and 7.69, for the dense graphs with $n = 2^{10}$ and 2^{11}, respectively, and speedups of 6.68, 7.13, and 7.41 for the sparse graphs of size $n = 2^{13}, 2^{14}$, and 2^{15}, respectively. We again observe that for sparse graphs our algorithm scales better than dense graphs: for spares graphs the running times almost consistently double as we double n. In comparison to Table 5.4, we see that setup times are lower as for the case of $\|\mathbf{AP} - \mathbf{PB}\|_2^2$ our implementation pre-computes some matrices.

5.5.5 Real Graph Pairs

We use our proposed ADMM algorithm to compute distances for the real graph pairs summarized in Table 5.1. We force a pair of graphs G_A and G_B to have the same number of nodes

$n = \max(|\mathcal{V}_A|, |\mathcal{V}_B|)$ by adding isolated (degree 0) dummy nodes to the smaller graphs.

For brevity, we only report results for $p = 2$. Fig. 5.5 shows the trace of residuals and norm. We see that our algorithm converges for all these graph pairs; for `cortex`, `monkey`, `protein`, `retweet`, `deezer`, and `slashdot`, the parameter ϵ is $0.01, 0.005, 0.0006, 0.001, 0.3$, and 0.06, respectively. The average iteration time of Alg. 8 for these pairs are 364(s), 712(s), 3836(s), 4375(s), 3797(s), and 2290(s), respectively, scaling well with $|\mathcal{Q}|$ and $|\mathcal{I}|$.

5.6 Conclusion

We present a massively parallel algorithm for graph distance computation via ADMM. We can consider penalty terms beyond the trace. Accelerating this method further, via, e.g., optimally partitioning the data, are important open problems. Our approach allows introducing additional penalty terms beyond the trace we considered here. Identifying means of accelerating this method further, as well as how to optimally partition the data, are important open problems.

The key in our distributed algorithm is Inner ADMM steps (Alg. 9) for solving the non-separable problem (5.14), which uses the efficient p-norm proximal operator as a building block. In the next chapter, we show that a similar algorithm can be adopted for minimizing the composition of the ℓ_p norm with a linear function plus a quadratic term-as is the case in (5.14)- and is highly efficient, in comparison with gradient methods.

We also showed that considering different ℓ_p norms in the objective is important, as different norms have better performance under different conditions, e.g., the type of noise added to data. In the next chapter we explore this further and propose to replace the commonly used mean squared error with the ℓ_p norm; we show that this objective is more robust to outliers and generally leads to better performance, e.g., higher accuracy in classification tasks.

Chapter 6

Robust Regression via Model Based Optimization

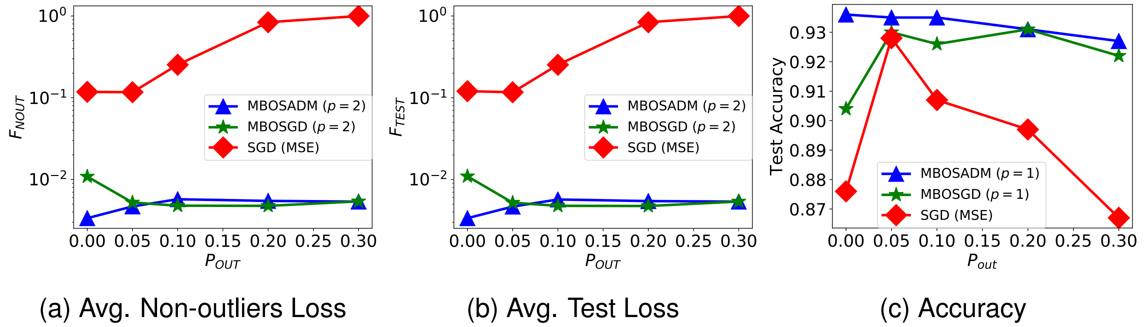
Mean Squared Error (MSE) loss problems are ubiquitous in machine learning and data mining. Such problems have the following form:

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \|F(\theta; \mathbf{x}_i)\|_2^2 + g(\theta), \tag{6.1}$$

where function $F : \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}^N$ captures the contribution of a sample $\mathbf{x}_i \in \mathbb{R}^m, i = 1, \ldots, n$, to the objective under the parameter $\theta \in \mathbb{R}^d$ and $g : \mathbb{R}^d \to \mathbb{R}$ is a regularizer. Example applications include training auto-encoders [44, 45], matrix factorization [46], and multi-target regression [47].

The MSE loss in (6.1) is computationally convenient, as the resulting problem is smooth and can thus be optimized efficiently via gradient methods, such as stochastic gradient descent (SGD). However, it is well-known that the MSE loss is not *robust to outliers* [2, 49, 50, 51, 52], i.e., samples far from the dataset mean. Intuitively, when squaring the error, outliers tend to dominate the objective.

To improve the sensitivity of MSE to outliers, Ding et al. [49] first suggested replacing the MSE with the ℓ_2 norm in the context of Principal Component Analysis (PCA). This motivated a line of research for developing robust algorithms using the ℓ_2 norm in different applications, e.g., non-negative matrix factorization [51], feature selection [53, 50], training autoencoders [44], and k-means clustering [54]. Attaining robustness via the ℓ_1 norm has also been used in matrix factorization [55, 56, 52], PCA [57, 58, 59], and regression [60]. Robustness of the ℓ_1 norm can be



(a) Avg. Non-outliers Loss (b) Avg. Test Loss (c) Accuracy

Figure 6.1: Robustness of ℓ_p norms vs. MSE to outliers introduced to MNIST when training an autoencoder. Figures 6.1a and 6.1b show the average loss over the non-outliers and the test set, respectively; values in each figure are normalized w.r.t. the largest value. The test accuracy of a logistic regression on the latent features is shown in Fig. 6.1c. We see that, under MSE, both for the loss values and classification accuracy are significantly affected by the fraction of outliers $P_{\texttt{out}}$. Robust embeddings under $p = 1, 2$ norms optimized via our proposed MBO methods exhibit almost constant behavior w.r.t. $P_{\texttt{out}}$.

linked to robustness of median to outliers in comparison to average value (see, e.g., Friedman et al. [2]).

Motivated by this approach, we study the following robust variant of Problem (6.1):

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \|F(\theta; \mathbf{x}_i)\|_p + g(\theta), \tag{6.2}$$

where $\|\cdot\|_p$ denotes an ℓ_p norm ($p \geq 1$). We are particularly interested in cases where F is not affine and, in general, Problem (6.2) is non-convex. This includes, e.g., feature selection [50], matrix factorization [51, 55], auto-encoders [44], and deep multi-target regression [60, 47]. Our problem includes robust variations considered in, e.g., [60, 50, 44, 51, 55], as special cases. However, these earlier algorithms are tailored to specific ℓ_p norms and/or do not generalize beyond the studied objective or application (some works, e.g., [50, 60], only consider convex problems). In contrast, we unify these variations for different applications as a non-convex and non-smooth problem, and present a general optimization algorithm for arbitrary ℓ_p norms.

A significant challenge behind solving Prob. (6.2) is that its objective is not smooth, precisely because the ℓ_p norm is not differentiable at $\mathbf{0} \in \mathbb{R}^N$. For non-convex and non-smooth problems of the form (6.2), where the objective contains a composite function, *Model-Based Optimization* (MBO) methods [61, 62, 63, 64, 65, 66] come with good experimental performance as well as theoretical guarantees. In particular, these MBO methods define a convex (but non-smooth) approximation of the main objective, called the *model function*. They then iteratively optimize this model function

plus a proximal quadratic term. Under certain conditions, MBO converges to a stationary point of the non-convex problem [61].

In this work, we use MBO to solve Problem (6.2) for arbitrary ℓ_p norms. In particular, each MBO iteration results in a convex optimization problem. We solve these sub-problems using a novel stochastic variant of the Online Alternating Direction Method (OADM) [67], which we call *Stochastic Alternating Direction Method* (SADM). Using SADM is appealing, as its resulting steps have efficient gradient-free solutions; in particular, we exploit a bi-section method [68, 20] for finding the proximal operator of ℓ_p norms. We provide theoretical guarantees for SADM. As an additional benefit, SADM comes with a stopping criterion, which is hard to obtain for gradient methods when the objective is non-smooth [69].

Overall, we make the following contributions:

- We study a general outlier-robust optimization that replaces the MSE with ℓ_p norms. We show that such problems can be solved via Model-Based Optimization (MBO) methods.
- We propose SADM, i.e., a stochastic version of OADM, and show that under strong convexity of the regularizer g, it converges with a $O(\log T/T)$ rate when solving the sub-problems arising at each MBO iteration.
- We conduct extensive experiments on training auto-encoders and multi-target regression. We show (a) the higher robustness of ℓ_p norms in comparison with MSE and (b) the superior performance of MBO, against stochastic gradient methods, both in terms of minimizing the objective and performing down-stream classification tasks. In some cases, we see that the MBO variant using SADM obtains objectives that are 29.6$\times$ smaller than the ones achieved by the competitors.

The performance of our MBO approach is illustrated in Fig. 6.1. An autoencoder trained via SGD over the MSE objective is significantly affected by the presence of outliers; in contrast, our MBO methods applied to ℓ_p objectives are robust to outliers. These relative benefits are also evident in a downstream classification task over the latent embeddings. The remainder of this paper is organized as follows. We introduce our robust formulation along with its applications in Sec. 6.1. We describe the instance of MBO applied to our problem in Sec. 6.2. We introduce SADM and its convergence analysis in Sec. 6.3 and present our experiments in Sec. 6.4. We finally conclude in Sec. 6.5.

6.1 Robust Regression and Applications

Notations. Lowercase boldface letters represent vectors, while capital boldface letters represent matrices. We also use the notation $[n] \triangleq \{1, 2, \ldots, n\}$.

Robust Regression. We first extend Prob. (6.2) to include constraints via:

$$\min_{\theta} \frac{1}{n} \sum_{i \in [n]} \|F(\theta; \mathbf{x}_i)\|_p + g(\theta) + \chi_{\mathcal{C}}(\theta), \tag{6.3}$$

where, again, $F : \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}^N$ is smooth, $|| \cdot ||_p$ is the ℓ_p norm, $g : \mathbb{R}^d \to \mathbb{R}$ is a convex regularizer such that $\inf g > -\infty$, while $\chi_{\mathcal{C}} : \mathbb{R}^d \to \{0, \infty\}$ is the indicator function of the convex set $\mathcal{C} \subseteq \mathbb{R}^d$. In practice, we are often interested in cases where either the regularizer or the constraint is absent.

Applications. For the sake of concreteness, we introduce some applications of Prob. (6.3). Function g is typically either the lasso (i.e., the ℓ_1 norm $g(\theta) = \|\theta\|_1$) or ridge regularizer (i.e., the ℓ_2 norm squared $g(\theta) = \|\theta\|_2^2$). We thus focus on the definition of $F(\cdot; \cdot)$ and constraint set $\mathcal{C}$ in each of these applications.

Auto-encoders [44]. Given n data points $\mathbf{x}_i \in \mathbb{R}^m$, $i \in [n]$, auto-encoders embed them in a $m'-$dimensional space, $m' \ll m$, as follows. The mapping to $\mathbb{R}^{m'}$ is done by a possibly non-linear function (e.g., a neural network) with $d_{\texttt{enc}}$ parameters $F_{\texttt{enc}} : \mathbb{R}^{d_{\texttt{enc}}} \times \mathbb{R}^m \to \mathbb{R}^{m'}$, called the *encoder*. An inverse mapping, the *decoder* $F_{\texttt{dec}} : \mathbb{R}^{d_{\texttt{dec}}} \times \mathbb{R}^{m'} \to \mathbb{R}^m$ with $d_{\texttt{dec}}$ parameters re-constructs the original points given latent embeddings. Both the encoder and the decoder are trained jointly over a dataset $\{\mathbf{x}_i\}_{i=1}^n$ by minimizing the reconstruction error; cast in our robust setting, this amounts to minimizing (6.3) with

$$F(\theta; \mathbf{x}_i) = \mathbf{x}_i - F_{\texttt{dec}}\left(\theta_{\texttt{dec}}; F_{\texttt{enc}}(\theta_{\texttt{enc}}; \mathbf{x}_i)\right), \tag{6.4}$$

where $\theta = [\theta_{\texttt{dec}}; \theta_{\texttt{enc}}] \in \mathbb{R}^{d_{\texttt{enc}}+d_{\texttt{dec}}}$ comprises the parameters of the encoder and the decoder. Robustness here aims to ameliorate the effect of outliers in the dataset $\{\mathbf{x_i}\}_{i=1}^n$. The constraint set can be $\mathbb{R}^d$ (i.e., the problem is unconstrained) or an ℓ_p-norm ball (i.e., $\{\theta \mid \|\theta\|_p \leq r\}$, for some $r > 0$, $p \geq 1$), when the magnitute of parameters is constrained; this can be used instead of a ℓ_1 or ℓ_2 norm regularizer. In stacked denoising autoencoders [48], the encoder and decoder are shallow and satisfy the additional constraint $\theta_{\texttt{enc}} = \theta_{\texttt{dec}}$.

Multi-target Regression [245]. We are given a set of n data points $\mathbf{x}_i \in \mathbb{R}^m$, $i \in [n]$ and the corresponding target labels $\mathbf{y}_i \in \mathbb{R}^{m'}$. The goal is to train a (again possibly non-linear) function

$f : \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}^{m'}$, with d parameters, to predict target values for a given vector $\mathbf{x} \in \mathbb{R}^m$. This maps to Prob. (6.3) via:

$$F(\theta; \mathbf{x}_i, \mathbf{y}_i) = \mathbf{y}_i - f(\theta; \mathbf{x}_i). \tag{6.5}$$

Robustness in this setting corresponds to ameliorating the effect of outliers in the *label* space, i.e., among labels $\{\mathbf{y}_i\}_{i=1}^n$. The constraint set can again be $\mathbb{R}^d$ or defined through an ℓ_p-norm ball (instead of the corresponding regularizer).

Matrix Factorization [246]. Given a matrix $\mathbf{X} \in \mathbb{R}^{n\times m}$., the goal is to express it a the product of two matrices $\mathbf{G}$,$\mathbf{H}$. Cast in our setting, each row $\mathbf{x}_i \in \mathbb{R}^m, i \in [n]$, of $\mathbf{X}$ is mapped to a lower dimensional sub-space as a vector $\mathbf{h}_i \in \mathbb{R}^{m'}$, where the sub-space basis is defined by the rows of the matrix $\mathbf{G} \in \mathbb{R}^{m\times m'}$. Function F is then given by $F(\theta; \mathbf{x}_i) = \mathbf{x}_i - \mathbf{G}\mathbf{h}_i$, where $\theta = (\mathbf{G}, \mathbf{H})$, where the rows of the matrix $\mathbf{H} \in \mathbb{R}^{n\times m'}$ are the low-dimensional embeddings $\mathbf{h}_i$. Robustness here limits sensitivity to ourliers in rows; a similar problem can be defined in terms of robustness to outliers in columns. Beyond usual boundedness constraints, additional constraints are introduced in so-called *non-negative matrix factorization* [246, 247], where matrices $\mathbf{G}$ and $\mathbf{H}$ are constrained to be non-negative.

For all three applications, we assume that F is smooth; this requires, e.g., smooth activation functions in deep models. Moreover, in all three examples, Prob. (6.3) is non-convex and non-smooth, as $\|\cdot\|_p$ is non-differentiable at $\mathbf{0} \in \mathbb{R}^N$.

6.2 Robust Regression via MBO

In this section, we outline how our non-smooth and non-convex problem in (6.3) can be solved via MBO. More broadly, non-smooth and non-convex optimization problems arise in many applications, such as non-negative matrix factorization [46], compressed sensing with non-convex norms [248], and ℓ_p norm regularized sparse regression problems with [249, 250]. Before presenting MBO for our problem, we review relavnt non-smoot non-convex optimization approaches.

A class of non-smooth non-convex optimization problems, known as *weakly convex problems* [251], i.e., problems in which the objective functions are summation of a convex function and a quadratic function, have attracted a lot of attention [61, 252, 65, 64, 253, 69]. Mai and Johansson [69] provided novel theoretical guarantees on the convergence of stochastic gradient descent with momentum for weakly-convex functions. However, in our experiments in Sec. 6.4, we

show that model-based methods considerably outperform these stochastic gradient methods with momentum.

Our approach falls under the class of *prox-linear* methods [61, 252, 63, 65, 64, 253], that solve problems where the objective is a composition of a non-smooth convex function and a smooth function, exactly as in Prob. (6.3). Such methods iteratively minimize the composition of the non-smooth function with the first-order approximation of the smooth function [61, 252, 65, 64]. Lewis and Wright [61] prove convergence to a stationary point while Drusvyatskiy et al. prove linear convergence [63] and obtain sample complexity guarantees [65]. Ochs et al. [66, 254] generalize prox-linear methods by proposing *Model-Based Optimization* (MBO) for both smooth and non-smooth non-convex problems. MBO reduces to a prox-linear method when the objective has a composite form, as in our case. Ochs et al. further considered non-quadratic proximal penalties in sub-problems and complemented MBO with an Armijo-like line search. We leverage both their line search and theoretical guarantees (c.f. Prop. 6.2.0.1); our main technical departure is in solving sub-problems per iteration via SADM, which we discuss next.

6.2.1 MBO

We now outline how non-smooth, non-convex Prob. (6.3) can be solved via *model-based optimization* (MBO) [66]. MBO relies on the use of a model function, which is a convex approximation of the main objective. In short, the algorithm proceeds iteratively, approximating function $F(\cdot;\cdot)$ by it's 1st order Taylor expansion at each iteration. This approximation is affine in θ, and results in a convex optimization problem per iteration.

In more detail, cast into our setting, MBO proceeds as follows. Starting with a feasible solution $\theta^0 \in \mathcal{C}$, it performs the following operations in each step $k \in \mathbb{N}$:

$$\tilde{\theta}^k = \arg\min_{\theta} F_{\theta^k}(\theta) + \frac{h}{2}\|\theta - \theta^k\|_2^2, \tag{6.6a}$$

$$\theta^{k+1} = (1-\eta^k)\theta^k + \eta^k\tilde{\theta}^k, \tag{6.6b}$$

where $h > 0$ is a regularization parameter, $\eta^k > 0$ is a step size, and function $F_{\theta^k} : \mathbb{R}^d \to \mathbb{R}$ is the so-called *model function* at θ^k, defined as:

$$F_{\theta^k}(\theta) \triangleq \frac{1}{n}\sum_{i\in[n]} \|F(\theta^k; \mathbf{x}_i) + \mathbf{D}_{F_i}(\theta^k)(\theta - \theta^k)\|_p + g(\theta) + \chi_{\mathcal{C}}(\theta), \tag{6.7}$$

Algorithm 11 Model-based Minimization (MBO)

1: **Input: Initial solution** $\theta^0 \in$ **dom**F**, iteration number** K **set** $\delta, \gamma \in (0, 1)$, **and** $\tilde{\eta} > 0$
2: **for** $k \in [K]$ **do**
3: $\quad \tilde{\theta}^k := \arg\min_\theta F_{\theta^k}(\theta) + \frac{h}{2}\|\theta - \theta^k\|_2^2$
4: $\quad$ Find γ^k via Armijo search rule
5: $\quad \theta^{k+1} := (1 - \eta^k)\theta^k + \eta^k\tilde{\theta}^k$
6: **end for**

where $\mathbf{D}_{F_i}(\theta) \in \mathbb{R}^{N\times d}$ is the Jacobian of $F(\theta; \mathbf{x}_i)$ w.r.t. θ. Put differently, in each step, MBO minimizes the 1st-order Taylor approximation augmented with a proximal penalty; the resulting $\tilde{\theta}^k$ is interpolated with the current solution θ^k. The above steps are summarized in Alg. 11.

Ochs et al. [66] allows to use more general Bregman divergences for the second term (c.f. Sec. 5.2 of [66]). The specific model functions we study fall under Example 5.3 in [66] (see also [61, 64].) Moreover, Ochs et al. allow (6.6a) to be solved inexactly; at each iteration, the solution $\tilde{\theta}^k$ only needs to improve the model function value by, i.e.,

$$\Delta_k \triangleq F_{\theta^k}(\tilde{\theta}^k) + \frac{h}{2}\|\tilde{\theta}^k - \theta^k\|_2^2 - F_{\theta^k}(\theta^k) < 0. \tag{6.8}$$

The step-size η^k is found via an Armijo line search algorithm. In particular, the line search algorithm finds a step-size, s.t., θ^{k+1} improves the current objective comparable with the model improvement Δ^k, i.e.,

$$\frac{1}{n}\sum_{i\in[n]}\|F(\theta^{k+1}; \mathbf{x}_i)\|_p + g(\theta^{k+1}) - \left(\frac{1}{n}\sum_{i\in[n]}\|F(\theta^k; \mathbf{x}_i)\|_p + g(\theta^k)\right) \leq \gamma\eta^k\Delta^k,$$

where $\gamma \in (0, 1)$ is a hyper-parameter of the linear search algorithm.

The exact Line Search Algorithm from Ochs et al. [66] is summarized in Alg. 12. Note that Ochs et al. prove that LSA is guaranteed to finish within finite number of iterations. The following proposition shows asymptotic convergence of MBO to a stationary point using an inexact solver

Proposition 6.2.0.1. *(Theorem 4.1 of [66]) Suppose θ^* is the limit point of the sequence θ^k generated by Alg. 11. Assume $F_{\theta^k}(\tilde{\theta}^k) + \frac{h}{2}\|\tilde{\theta}^k - \theta^k\|_2^2 - \inf_{\tilde{\theta}} F_{\theta^k}(\tilde{\theta}) + \frac{h}{2}\|\tilde{\theta} - \theta^k\|_2^2 \leq \epsilon^k$, for all iterations k, and that $\epsilon^k \to 0$. Then θ^* is a stationary point of Prob.* (6.3).

Proof. We show that all assumptions for Theorem 4.1 of Ochs et al. [66] are satisfied, therefore the result holds. We solve convex problems (6.6a) via OADM iterations. Since we established the $O(\frac{\log T}{T})$ convergence rate of OADM in Theorem 6.3.1, we can solve (6.6a) with an arbitrary

Algorithm 12 Line Search Algorithm (LSA)

1: **Input:** Solutions θ^k, $\tilde{\theta}^k$, and parameters $\delta, \gamma \in (0,1), \tilde{\eta} > 0$
2: Initialize $\tilde{\theta} := (1-\tilde{\eta})\theta^k + \tilde{\eta}\tilde{\theta}^k$
3: **while** $\frac{1}{n}\sum_{i\in[n]} \|F(\tilde{\theta};\mathbf{x}_i)\|_p + g(\tilde{\theta}) > \frac{1}{n}\sum_{i\in[n]} \|F(\theta^k;\mathbf{x}_i)\|_p + g(\theta^k) + \gamma\tilde{\eta}\big[F_{\theta^k}(\tilde{\theta}^k) - F_{\theta^k}(\theta^k) + \frac{h}{2}\|\tilde{\theta}^k - \theta^k\|_2^2\big]$ **do**
4: Set $\tilde{\eta} := \delta\tilde{\eta}$
5: Set $\tilde{\theta} := (1-\tilde{\eta})\theta^k + \tilde{\eta}\tilde{\theta}^k$
6: **end while**
7: **Return:** $\tilde{\eta}$

accuracy ϵ, which goes to zero for $T \to \infty$; therefore, Assumption 4.1 of [66] is satisfied. Moreover, due to the choice of the Euclidean norm as our Bergman distance function, Assumption 4.2 of [66] is satisfied (see Section 5 of [66]). Finally, our model function (6.7) is precisely Example 5.3 in [66]; it is written as the form $f_0 + h \circ F$, where $f_0(\theta) = \chi_{\mathcal{C}}(\theta) + g(\theta)$, $h(\mathbf{F}) = \|\mathbf{F}\|_{p,1}$, and $F : \mathbb{R}^d \to \mathbb{R}^{n\times N}$. Therefore, Assumption 4.3 is also satisfied. In addition, the domain of Euclidean distance is $\mathbb{R}^n$, which implies that Condition (ii) in Theorem 4.1 of [66] is satisfied for any limit point. Thus, all the conditions in Theorem 4.1 of Ochs et al. are satisfied, and every limit point of Alg. 11 is a stationary point. □

Problem (6.6a) is convex but still non-smooth; we discuss how it can be solved efficiently via SADM in the next section.

6.3 Stochastic Alternating Direction Method of Multipliers

After dealing with convexity via MBO, there are still two challenges behind solving the constituent sub-problem (6.6a). The first is the non-smoothness of $\|\cdot\|_p$; the second is scaling in n, which calls for a the use of a stochastic optimization method, akin to SGD (which, however, is not applicable due to the lack of smoothness). We address both through the a novel approach, namely, SADM, which is a stochastic version of the OADM algorithm by Wang and Banerjee [67, 255]. Most importantly, our approach reduces the solution of Prob. (6.6a) to several gradient-free optimization sub-steps, which can be computed efficiently. In addition, using an SADM/ADMM variant comes with clear stopping criteria, which is challenging for traditional stochastic subgradient methods [69].

6.3.1 OADM

The Alternating Direction Method of Multipliers (ADMM) [13] is a convex optimization algorithm that provides efficient methods for non-smooth problems. Applying ADMM often results in sub-problems that can be solved efficiently via proximal operators [13, 256, 257]. To speed up ADMM, stochastic variants [258, 259, 260] have been proposed for minimizing sum-like objectives. These stochastic variants, similar to SGD, update solutions using the gradients of a small batch of terms in the objective, at each iteration. Another group of works proposed online variants of ADMM [67, 261, 262]. In these variants, the goal is to minimize the summation of loss functions that are revealed by an adversary.

Wang and Banerjee [67] proposed the first online variant of ADMM, as Online Direction Method of Multipliers (OADM). Here, we propose a stochastic version of OADM, Stochastic Alternating Direction Method (SADM), for solving inner-problems in MBO iterations. SADM is similar to OADM with the difference that functions are sampled uniformly at random and are not given by an adversary. We prove that SADM converges with a $O(\log T/T)$ rate when the regularizer is strongly convex. Other existing stochastic or online ADMM variants either require a smooth objective [258, 259] or bounded sub-gradients [260, 261], neither of which apply for the inner-problems we solve. In contrast, we show that the subsequent steps of SADM admit gradient-free efficient solutions due to a bi-section method for finding proximal operators of ℓ_p norms [68, 20].

6.3.2 SADM

We first describe how our SADM can be applied to solve Prob. (6.6a). We introduce the following notation to make our exposition more concise:

$$F^{(k)}(\theta;\mathbf{x}_i) \triangleq \|F(\theta^k;\mathbf{x}_i) + \mathbf{D}_{F_i}(\theta^k)(\theta-\theta^k)\|_p + \frac{h}{2}\|\theta-\theta^k\|_2^2, \tag{6.9a}$$

$$F^{(k)}(\theta) \triangleq \frac{1}{n}\sum_{i\in[n]} F^{(k)}(\theta,\mathbf{x}_i), \tag{6.9b}$$

$$G(\theta) \triangleq g(\theta) + \chi_{\mathcal{C}}(\theta). \tag{6.9c}$$

We can then rewrite Prob. (6.6a) as the following equivalent problem:

$$\text{Minimize} \quad F^{(k)}(\theta_1) + G(\theta_2) \tag{6.10a}$$

$$\text{subject to:} \quad \theta_1 = \theta_2, \tag{6.10b}$$

where $\theta_1, \theta_2 \in \mathbb{R}^d$ are auxiliary variables.

Note that the objective in (6.10a) is equivalent to $F^{(k)}(\theta_1) + G(\theta_2)$. SADM starts with initial solutions, i.e., $\theta_1^0 = \theta_2^0 = \mathbf{u}^0 = 0$. At the t-th iteration, the algorithm performs the following steps:

$$\theta_1^{t+1} := \arg\min_{\theta_1} F^{(k)}(\theta_1; \mathbf{x}_t) + \frac{\rho_t}{2}\|\theta_1 - \theta_2^t + \mathbf{u}^t\|_2^2 + \frac{\gamma_t}{2}\|\theta_1 - \theta_1^t\|_2^2, \tag{6.11a}$$

$$\theta_2^{t+1} := \arg\min_{\theta_2} G(\theta_2) + \frac{\rho_t}{2}\|\theta_1^{t+1} - \theta_2 + \mathbf{u}^t\|_2^2, \tag{6.11b}$$

$$\mathbf{u}^{t+1} := \mathbf{u}^t + \theta_1^{t+1} - \theta_2^{t+1}, \tag{6.11c}$$

where variables $\mathbf{x}_t$ are sampled uniformly at random from $\{\mathbf{x}_i\}_{i=1}^n$, $\mathbf{u}^t \in \mathbb{R}^d$ is the dual variable, the $\rho_t, \gamma_t > 0$ are scaling coefficients at the t-th iteration. We explain how to set ρ_t, γ_t in Thm. 6.3.1.

The solution to Problem (6.11b) amounts to finding the proximal operator of function G. In general, given that g is smooth and convex, this is a strongly convex optimization problem and can be solved via standard techniques. Nevertheless, for several of the practical cases we described in Sec. 6.1 this optimization can be done efficiently with gradient-free methods. For example, in the case where the regularizer g is the either a ridge or lasso penalty, and $\mathcal{C} = \mathbb{R}^d$, is is well-known that proximal operators for ℓ_1 and ℓ_2 norms have closed-form solutions [13]. For general ℓ_p norms, the efficient (gradient-free) bi-section method due to Liu and Ye [68], which we introduced in the previous chapter (see Sec. 5.3.2) can be used to compute the proximal operator. Moreover, in the absence of the regularizer, the proximal operator for the indicator function $\chi_{\mathcal{C}}$ is equivalent to projection on the convex set $\mathcal{C}$. This again has closed-form solution, e.g., when $\mathcal{C}$ is the simplex [242] or an ℓ_p-norm ball [231, 68].

Problem (6.11a) is harder to solve; we show however that it can also reduced to the (gradient-free) bi-section method due to Liu and Ye [68] in the next section.

6.3.3 Inner ADMM

We solve Problem (6.11a) using another application of ADMM. In particular, note that (6.11a) assumes the following general form:

$$\min_{\mathbf{x}} \quad \|\mathbf{A}\mathbf{x} + \mathbf{b}\|_p + \lambda\|\mathbf{x} - \mathbf{c}\|_2^2, \tag{6.12}$$

where $\mathbf{A} = \mathbf{D}_{F_t}(\theta^{(k)})$, the constituent parameter vectors are $\mathbf{c} = \frac{\rho_t}{\rho_t+\gamma_t+h}(\theta_2^t - \mathbf{u}^t) + \frac{\gamma_t}{\rho_t+\gamma_t+h}\theta_1^t + \frac{h}{\rho_t+\gamma_t+h}\theta^{(k)}$, $\mathbf{b} = F(\theta^{(k)}; \mathbf{x}_t) - \mathbf{D}_{F_t}(\theta^{(k)})\theta^{(k)}$, and $\lambda = \frac{\rho_t+\gamma_t+h}{2}$.

We solve (6.12) via ADMM by reformulating it as the following problem:

$$\min \quad \|\mathbf{y}\|_p + \lambda\|\mathbf{x} - \mathbf{c}\|_2^2 \tag{6.13a}$$

$$\text{s.t} \quad \mathbf{A}\mathbf{x} + \mathbf{b} - \mathbf{y} = 0. \tag{6.13b}$$

The ADMM steps at the k-th iteration for (6.13) are the following:

$$\mathbf{y}^{k+1} := \arg\min_{\mathbf{y}} \|\mathbf{y}\|_p + \rho'/2\|\mathbf{y} - \mathbf{A}\mathbf{x}^k - \mathbf{b} + \mathbf{z}^k\|_2^2 \tag{6.14a}$$

$$\mathbf{x}^{k+1} := \arg\min_{\mathbf{x}} \lambda\|\mathbf{x} - \mathbf{c}\|_2^2 + \rho'/2\|\mathbf{y}^{k+1} - \mathbf{A}\mathbf{x} - \mathbf{b} + \mathbf{z}^k\|_2^2 \tag{6.14b}$$

$$\mathbf{z}^{k+1} := \mathbf{z}^k + \mathbf{y}^{k+1} - \mathbf{A}\mathbf{x}^{k+1} - \mathbf{b}, \tag{6.14c}$$

where $\mathbf{z}^k \in \mathbb{R}^N$ denotes the dual variable at the k-th iteration and $\rho' > 0$ is a hyper-parameter of ADMM.

Problem (6.14a) is again equivalent to computing the proximal operator of the p-norm, which, as mentioned earlier, has closed-form solution for $p = 1, 2$. Moreover, for general ℓ_p-norms the proximal operator can be computed via the bi-section algorithm by Liu and Ye [68]. This bi-section method yields a solution with an ϵ accuracy in $O(\log_2(1/\epsilon))$ rounds [20, 68] (see Sec. 5.3.2).

6.3.4 Convergence

To attain the convergence guarantee of MBO given by Proposition 6.2.0.1, we need to solve the inner problem (6.8) within accuracy ϵ^k at iteration k, where $\epsilon^k \to 0$. As our major technical contribution, we ensure this by proving the convergence of SADM when solving Prob. (6.8).

Consider the sequence $\{\theta_1^t, \theta_2^t, \mathbf{u}^t\}_{t=1}^T$ generated by our SADM algorithm (6.11), where $\mathbf{x}_t$, $t \in [T]$, are sampled u.a.r. from $\{\mathbf{x}_i\}_{i=1}^n$.. Let also

$$\bar{\theta}_1^T \triangleq \frac{1}{T}\sum_{t=1}^{T}\theta_1^t,\ \bar{\theta}_2^T \triangleq \frac{1}{T}\sum_{t=1}^{T}\theta_2^{t+1}, \tag{6.15}$$

denote the time averages of the two solutions. Let also $\theta^* = \theta_1^* = \theta_2^*$ be the optimal solution of Prob. (6.10). Finally, denote by

$$R^T \triangleq F^{(k)}(\bar{\theta}_1^T) + G(\bar{\theta}_2^T) - F^{(k)}(\theta^*) - G(\theta^*) \tag{6.16}$$

the residual error of the objective from the optimal.

Then, the following theorem holds:

Theorem 6.3.1. *Assume that $\mathcal{C}$ is convex, closed, and bounded, and that $g(0) = 0$, $g(\theta) \geq 0$, and $g(\cdot)$ is both Lipschitz continuous and β-strongly convex over $\mathcal{C}$. Moreover, assume that both the function $F(\theta; \mathbf{x}_i)$ and its Jacobian $\mathbf{D}_{F_i}(\theta)$ are bounded on the set $\mathcal{C}$, for all $i \in [n]$. We set $\gamma_t = ht$ and $\rho_t = \beta t$. Then,*

$$\|\bar{\theta}_1^T - \bar{\theta}_2^T\|_2^2 = O\left(\frac{\log T}{T}\right) \tag{6.17a}$$

$$\mathbb{E}[R^T] = O\left(\frac{\log T}{T}\right) \tag{6.17b}$$

$$\mathbb{P}\left(R^T \geq k_1 \frac{\log T}{T} + k_2 \frac{M}{\sqrt{T}}\right) \leq e^{-\frac{M^2}{16}} \text{ for all } M > 0, T \geq 3, \tag{6.17c}$$

where $k_1, k_2 > 0$ are constants (see (6.35) *in Sec. 6.3.5 for exact definitions).*

We prove Theorem 6.3.1 next in Sec. 6.3.5. The theorem has the following important consequences. First, (6.17a) implies that the infeasibility gap between θ_1 and θ_2 decreases as $O(\frac{\log T}{T})$ *deterministically*. Second, by (6.17b) the residual error R^T decreases as $O(\frac{\log T}{T})$ in expectation. Finally, (6.17c) shows that the tail of the residual error as iterations increases is exponentially bounded. In particular, given a desirable accuracy ϵ_k, (6.17c) gives the number of iterations necessary be within ϵ_k of the optimal with any probability $1 - \delta$. Therefore, according to Proposition 6.2.0.1, using SADM will result in convergence of Algorithm 11 with high probability. Finally, we note that, although we write Theorem 6.3.1 for updates using only one random sample per iteration, the analysis and guarantees readily extend to the case where a batch selected u.a.r. is used instead. More formally, we have the following:

Corollary 6.3.1.1. *(Batch Setting) Assume the assumptions of Theorem 6.3.1 and let the sequence $\{\theta_1^t, \theta_2^t, \mathbf{u}^t\}, t \in [T]$ be generated by OADM algorithm, i.e.,* (6.11)*, where the step* (6.11a) *is replaced with the following step:*

$$\theta_1^{t+1} := \arg\min_{\theta_1} \frac{1}{J} \sum_{j=1}^{J} F^{(k)}(\theta_1; \mathbf{x}_t^j) + \frac{\rho}{2}\|\theta_1 - \theta_2^t + \mathbf{u}^t\|_2^2 + \frac{\gamma}{2}\|\theta_1 - \theta_1^t\|_2^2, \tag{6.18}$$

where at each iteration $t \in [T]$, the points $\mathbf{x}_t^j \in \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ are i.i.d. samples drawn uniformly at random. Then all the results of Theorem 6.3.1 in (6.17) *hold.*

Proof. We only need to adapt the definition in (6.9a) to this batch setting as follows: (6.9a) as follows

$$F^{(k)}(\theta; [\mathbf{x}^j]_{j\in[J]}) \triangleq \frac{1}{J} \sum_{j=1}^{J} F_{\theta^{(k)}}(\theta, \mathbf{x}^j) + \frac{h}{2}\|\theta - \theta^{(k)}\|_2^2. \tag{6.19}$$

Then all the results follow similar to the proof presented in Sec. 6.3.5. □

6.3.5 Proof of Theorem 6.3.1

Since we assume that the constraint set $\mathcal{C}$ is convex, closed, and bounded, we take the diameter of the set $\mathcal{C}$ to be

$$D_{\mathcal{C}} = \max_{\theta_1, \theta_2 \in \mathcal{C}} \|\theta_1 - \theta_2\|_2. \tag{6.20}$$

In addition, since function F and its Jacobian $\mathbf{D}_{F_i}(\theta)$ are bounded on the set $\mathcal{X}$, for all $i \in [n]$, there exist $M_F, M_D < \infty$, s.t.,

$$\|F(\theta; \mathbf{x})\|_\infty \leq M_F \;\; \forall \theta \in \mathcal{C}, \mathbf{x} \in \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \tag{6.21a}$$

$$\|\mathbf{D}_{F_i}(\theta)\|_\infty \leq M_D \;\; \forall \theta \in \mathcal{C}, i \in [n]. \tag{6.21b}$$

We now paraphrase Theorem 6 of [255] as the following lemma.

Lemma 6.3.2 (Theorem 6 in [255])**.** *If the assumptions of Theorem 6.3.1 hold, for the sequence* $\{\theta_1^t, \theta_2^t, \mathbf{u}^t\}, t \in [T]$ *generated by OADM algorithm for any sequence of the variables* $\mathbf{x}_t \in \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}, t \in [T]$ *the following holds*

$$\sum_{t=1}^{T} \left(F^{(k)}(\theta_1^t; \mathbf{x}_t) + G(\theta_2^{t+1}) \right) - \sum_{t=1}^{T} \left(F^{(k)}(\theta^*; \mathbf{x}_t) + G(\theta^*) \right)$$
$$\leq \frac{(N^{1/p} d M_D + h D_{\mathcal{C}})^2}{2h} \log(T+1) + \frac{\beta + h}{2} D_{\mathcal{C}}^2 \tag{6.22a}$$

$$\sum_{t=1}^{T} \|\theta_1^{t+1} - \theta_2^{t+1}\|_2^2 + \|\theta_2^{t+1} - \theta_2^t\|_2^2$$
$$\leq \frac{2}{\beta}(\sqrt{d} M_G + h D_{\mathcal{C}} + L_G) \log(T+1) + (1 + \frac{h}{\beta}) D_{\mathcal{C}}^2, \tag{6.22b}$$

where $\theta^* = \theta_1^* = \theta_2^*$ *is the optimal solution for* (6.10)*, and* L_G *is the Lipschitz coefficient for* $g(\cdot)$*, and* $M_G \triangleq N^{1/p} \sqrt{d} M_D$*.*

Proof. We show that Assumption 3 of [255] is satisfied, thus the results follow from Theorem 6 of [255]. For case (a), we need to show that the subgradient of the functions $F^{(k)}(\theta; \mathbf{x}_t)$ are bounded.

For any subgradient $\mathbf{g} \in \partial \left(\|F(\theta^{(k)}; \mathbf{x}_t) + \mathbf{D}_t^{(k)}(\theta - \theta^{(k)})\|_p \right)$ and for all $\theta \in \mathbb{R}^d$, we have

$$\begin{aligned}
\mathbf{g}^\top(\theta - \theta_1) &\leq \|F(\theta^{(k)}; \mathbf{x}_t) + \mathbf{D}_t^{(k)}(\theta - \theta^{(k)})\|_p - \|F(\theta^{(k)}; \mathbf{x}_t) + \mathbf{D}_t^{(k)}(\theta_1 - \theta^{(k)})\|_p \\
&\leq \|\mathbf{D}_t^{(k)}(\theta - \theta_1)\|_p \\
&= \left(\sum_{i=1}^{N} |\mathbf{D}_i^\top(\theta - \theta_1)|^p \right)^{1/p} \\
&\overset{\text{Cauchy Schwarz Ineq.}}{\leq} \left(\sum_{i=1}^{N} (\|\mathbf{D}_i\|_2 \|\theta - \theta_1\|_2)^p \right)^{1/p} \\
&\overset{(6.21b)}{\leq} N^{1/p}\sqrt{d} M_D \|\theta - \theta_1\| = M_G \|\theta - \theta_1\|,
\end{aligned}$$

where $\mathbf{D}_i$ is the i-th row of $\mathbf{D}_t^{(k)}$. Now given that the above holds for all θ, we can show that for every element $i \in [d]$, $\mathbf{g}_i$ is bounded by M_G; to see this set $\theta = \theta_1 + e_i$ and it follows that $\mathbf{g}_i \leq M_G$. We therefore conclude that the subgradients $\partial F^{(k)}(\theta, \zeta_t) = \{\mathbf{g} + h(\theta - \theta^{(k)}) | \mathbf{g} \in \partial F_{\theta^{(k)}}(\theta_1, \zeta_t)\}, \theta \in \mathcal{C}$ are bounded:

$$\begin{aligned}
\|\mathbf{g}_F\|_2 = \|\mathbf{g} + h(\theta - \theta^{(k)})\|_2 &\leq \|\mathbf{g}\|_2 + h\|(\theta - \theta^{(k)})\|_2 \\
&\leq \sqrt{d} M_G + h D_{\mathcal{C}} \;\; \forall \mathbf{g}_F \in \partial F^{(k)}(\theta; \mathbf{x}_t), \qquad (6.23)
\end{aligned}$$

where the last inequality is due to the fact that $D_{\mathcal{C}} = \max_{\theta_1, \theta_2 \in \mathcal{C}} \|\theta_1 - \theta_2\|_2$.

Case (b) is satisfied with the choice of ℓ_2 norm squared for Bregman distance, i.e., the term $\gamma \|\theta_1 - \theta_1^t\|_2^2$. Since we assumed the constraint set $\mathcal{C}$ is convex, closed and bounded, we take the diameter of the set $\mathcal{C}$ to be

$$D_{\mathcal{C}} = \max_{\theta_1, \theta_2 \in \mathcal{C}} \|\theta_1 - \theta_2\|_2.$$

Thus, case (c) is satisfied as a result of initialization ($\theta_1^1 = \theta_2^1 = \mathbf{u}^1 = 0$), and the fact that $\|\theta_1^1 - \theta^*\|_2 \leq D_{\mathcal{C}}$ and $\|\theta_2^1 - \theta^*\|_2 \leq D_{\mathcal{C}}$. Case (d) is directly included in the assumption of Theorem 6.3.1. Finally, for Case (e) we have

$$\begin{aligned}
&\left| F^{(k)}(\theta_1^{t+1}; \mathbf{x}_t) + G(\theta_2^{t+1}) - \left(F^{(k)}(\theta^*; \mathbf{x}_t) + G(\theta^*) \right) \right| \\
&\leq \left| F^{(k)}(\theta_1^{t+1}, \zeta_t) - F^{(k)}(\theta^*, \zeta_t) \right| + \left| G(\theta_2^{t+1}) - G(\theta^*) \right| \\
&\overset{(a)}{\leq} (\sqrt{d} M_G + h D_{\mathcal{C}}) \|\theta_1^{t+1} - \theta^*\|_2 + L_G \|\theta_2^{t+1} - \theta^*\|_2 + \\
&|\chi_{\mathcal{C}}(\theta_2^{t+1}) - \chi_{\mathcal{C}}(\theta^*)| \\
&\overset{(b)}{\leq} (\sqrt{d} M_G + h D_{\mathcal{C}} + L_G) D_{\mathcal{C}},
\end{aligned}$$

where we derive (a) using the Lipschitz continuity of $g(\cdot)$ along with Lemma 2.6 from [263], which states that $F^{(k)}$ is Lipschitz continuous if and only if some ℓ_p norm of its subgradients is bounded (that we showed in (6.23)). Moreover, in deriving (b) we use the fact that the constraint set $\mathcal{C}$ is convex, closed, and bounded, and that $\chi_{\mathcal{C}}(\theta_2^{t+1}) = \chi_{\mathcal{C}}(\theta^*) = 0$, as $\theta_2^{t+1}, \theta^* \in \mathcal{C}$. So far we have shown that all cases in Assumption 3 of [255] are satisfied. Also, both $F^{(k)}$ and G are h and β strongly convex, respectively; the former is due to the quadratic term and the latter is explicitly stated in our assumptions. Therefore, we have shown that all assumptions in Theorem 6 [255] are satisfied and the results in (6.22) follow from the theorem. □

From (6.22b) we obtain the following

$$\sum_{t=1}^{T} \|\theta_1^t - \theta_2^t\|_2^2 = O(\log T) \tag{6.24a}$$

$$\sum_{t=1}^{T} \|\theta_2^{t+1} - \theta_2^t\|_2^2 = O(\log T). \tag{6.24b}$$

Now we derive the result

$$\begin{aligned}
\|\bar{\theta}_1^T - \bar{\theta}_2^T\|_2^2 &= \|\frac{1}{T}\sum_{t=1}^{T}(\theta_1^t - \theta_2^{t+1})\|_2^2 \\
&= \|\frac{1}{T}\sum_{t=1}^{T}(\theta_1^t - \theta_2^t + \theta_2^t - \theta_2^{t+1})\|_2^2 \\
&\leq \|\frac{2}{T}\sum_{t=1}^{T}(\theta_1^t - \theta_2^t)\|_2^2 + \|\frac{2}{T}\sum_{t=1}^{T}(\theta_2^t - \theta_2^{t+1})\|_2^2 \\
&\leq \frac{2}{T}\sum_{t=1}^{T}\|\theta_1^t - \theta_2^t\|_2^2 + \frac{2}{T}\sum_{t=1}^{T}\|\theta_2^t - \theta_2^{t+1}\|_2^2 \\
&\overset{(6.24)}{=} O(\frac{\log T}{T}),
\end{aligned}$$

where in deriving the first inequality we have used the fact that

$$\|\mathbf{x} + \mathbf{y}\|_2^2 \leq 2\|\mathbf{x}\|_2^2 + 2\|\mathbf{y}\|_2^2 \; \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^d.$$

Now we prove the second part of theorem about the optimality of solutions. Using

convexity of $F^{(k)}$ and G we have that

$$
\begin{aligned}
&F^{(k)}(\bar{\theta}_1^T) + G(\bar{\theta}_2^T) - F^{(k)}(\theta^*) - G(\theta^*) \\
&\leq \frac{1}{T}\sum_{t=1}^{T}\Big(F^{(k)}(\theta_1^t) - F^{(k)}(\theta^*) + G(\theta_2^{t+1}) - G(\theta^*)\Big) \\
&= \frac{1}{T}\sum_{t=1}^{T}\Big(F^{(k)}(\theta_1^t;\mathbf{x}_t) - F^{(k)}(\theta^*;\mathbf{x}_t) + G(\theta_2^{t+1}) - G(\theta^*)\Big) \\
&+ \frac{1}{T}\sum_{t=1}^{T}\Big(F^{(k)}(\theta_1^t) - F^{(k)}(\theta_1^t;\mathbf{x}_t) - F^{(k)}(\theta^*) + F^{(k)}(\theta^*;\mathbf{x}_t)\Big) \\
&\overset{(6.22a)}{\leq} \frac{(N^{1/p}dM_D + hD_{\mathcal{C}})^2}{2hT}\log(T+1) + \frac{\beta+h}{2T}D_{\mathcal{C}}^2 + \frac{1}{T}\sum_{t=1}^{T}\delta_t,
\end{aligned}
\tag{6.25}
$$

where the first inequality is due to the Jensen's inequality and

$$
\delta_t \triangleq F^{(k)}(\theta_1^t) - F^{(k)}(\theta_1^t;\mathbf{x}_t) - F^{(k)}(\theta^*) + F^{(k)}(\theta^*;\mathbf{x}_t).
$$

As the variables $\mathbf{x}_t$ and θ_1^t are independent, we have

$$
\begin{aligned}
&\mathbb{E}[\delta_t|\mathbf{x}_1,\ldots\mathbf{x}_{t-1}] \\
&= \mathbb{E}\Big[F^{(k)}(\theta_1^t) - F^{(k)}(\theta_1^t;\mathbf{x}_t) - F^{(k)}(\theta^*) + F^{(k)}(\theta^*;\mathbf{x}_t)|\mathbf{x}_1,\ldots\mathbf{x}_{t-1}\Big] = 0.
\end{aligned}
$$

Therefore, we obtain

$$
\mathop{\mathbb{E}}_{\mathbf{x}_t, t\in[T]}[\delta_t] = 0. \tag{6.26}
$$

Now taking expectations of both sides of (6.25) w.r.t. the sequence $\zeta_t, t\in[T]$ and noting (6.26) we have that:

$$
\begin{aligned}
&\mathop{\mathbb{E}}_{\mathbf{x}_t, t\in[T]}\Big[F^{(k)}(\bar{\theta}_1^T) + G(\bar{\theta}_2^T) - F^{(k)}(\theta^*) - G(\theta^*)\Big] \\
&\leq \frac{(N^{1/p}dM_D + hD_{\mathcal{C}})^2}{2hT}\log(T+1) + \frac{\beta+h}{2T}D_{\mathcal{C}}^2 \\
&= O(\frac{\log T}{T})
\end{aligned}
$$

We first derive the following for all $\theta \in \mathcal{C}, \mathbf{x}_i \in \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$:

$$
\begin{aligned}
&|F^{(k)}(\theta) - F^{(k)}(\theta; \mathbf{x}_i)| \\
&= \left| F_{\theta^{(k)}}(\theta; \mathbf{x}_i) - \frac{1}{n} \sum_{j=1}^{n} F_{\theta^{(k)}}(\theta; \mathbf{x}_j) \right| \\
&\leq \frac{1}{n} \sum_{j \neq i} |F_{\theta^{(k)}}(\theta; \mathbf{x}_i) - F_{\theta^{(k)}}(\theta; \mathbf{x}_j)| \\
&= \frac{1}{n} \sum_{j \neq i} \mid \|F(\theta^{(k)}; \mathbf{x}_i) + \mathbf{D}_{F_i}(\theta^{(k)})(\theta - \theta^{(k)}))\|_p \\
&- \|F(\theta^{(k)}; \mathbf{x}_j) + \mathbf{D}_{F_j}(\theta^{(k)})(\theta - \theta^{(k)})\|_p \mid \\
&\leq \frac{1}{n} \sum_{j \neq i} \|F(\theta^{(k)}; \mathbf{x}_i) + \mathbf{D}_{F_i}(\theta^{(k)})(\theta - \theta^{(k)})) - F(\theta^{(k)}; \mathbf{x}_j) + \mathbf{D}_{F_j}(\theta^{(k)})(\theta - \theta^{(k)})\|_p \\
&\leq \frac{1}{n} \sum_{j \neq i} \|F(\theta^{(k)}; \mathbf{x}_i) - F(\theta^{(k)}; \mathbf{x}_j)\|_p + \frac{1}{n} \sum_{j \neq i} \left\| \left(\mathbf{D}_{F_i}(\theta^{(k)}) - \mathbf{D}_{F_j}(\theta^{(k)}) \right) (\theta - \theta^{(k)}) \right\|_p \\
&\overset{(6.21a)}{\leq} \sqrt[p]{N} M_F + \frac{1}{n} \sum_{j \neq i} \left(\sum_{i'=1}^{N} \mid \left(\mathbf{D}_{F_i}(\theta^{(k)}) - \mathbf{D}_{F_j}(\theta^{(k)}) \right)_{i'}^{\top} (\theta - \theta^{(k)}) \mid|^p \right)^{1/p} \\
&\leq \sqrt[p]{N} M_F + \frac{1}{n} \sum_{j \neq i} \left(\sum_{i'=1}^{N} \left(\left\| \left(\mathbf{D}_{F_i}(\theta^{(k)}) - \mathbf{D}_{F_j}(\theta^{(k)}) \right)_{i'} \right\|_2 \|\theta - \theta^{(k)}\|_2 \right)^p \right)^{1/p} \\
&\overset{(6.21b),(6.20)}{\leq} \sqrt[p]{N} M_F + \sqrt[p]{N} \sqrt{d} M_D D_{\mathcal{C}}.
\end{aligned} \tag{6.27}
$$

In order to show the rest of the results we first show that the variance of δ_t are bounded for all $t \in [T]$

$$
\begin{aligned}
\delta_t^2 &= \left(F^{(k)}(\theta_1^t) - F^{(k)}(\theta_1^t, \zeta_t) - F^{(k)}(\theta^*) + F^{(k)}(\theta^*; \mathbf{x}_t) \right)^2 \\
&\leq 2 \left(F^{(k)}(\theta_1^t) - F^{(k)}(\theta_1^t; \mathbf{x}_t) \right)^2 + 2 \left(-F^{(k)}(\theta^*) + F^{(k)}(\theta^*; \mathbf{x}_t) \right)^2 \\
&\overset{(6.27)}{\leq} 4 \left(\sqrt[p]{N} M_F + \sqrt[p]{N} \sqrt{d} M_D D_{\mathcal{X}} \right)^2 \\
&\equiv \sigma^2.
\end{aligned} \tag{6.28}
$$

From (6.28) it is obvious that

$$
\left(\frac{\delta_t^2}{\sigma^2} \right) \leq \exp(1) \;\; \forall t \in [T]. \tag{6.29}
$$

Now we show that the following holds

$$
\mathbb{E}[\exp(\alpha \delta_t) | \mathbf{x}_1, \ldots, \mathbf{x}_{t-1}] \leq \exp(\alpha^2 \sigma^2) \;\; \forall \alpha > 0. \tag{6.30}
$$

To show (6.30), we follow results from [264]; similar to them, we use the fact that $\exp(x) \leq x + \exp(x^2)$. Then we have that

$$\begin{aligned}
\mathbb{E}[\exp(\alpha\delta_t)|\mathbf{x}_1,\ldots,\mathbf{x}_{t-1}] &\leq \mathbb{E}[\exp(\alpha^2\delta_t^2)|\mathbf{x}_1,\ldots,\mathbf{x}_{t-1}] \\
&= \mathbb{E}\left[\left(\exp(\frac{\delta_t^2}{\sigma^2})\right)^{\alpha^2\sigma^2} \mid \mathbf{x}_1,\ldots,\mathbf{x}_{t-1}\right] \\
&\overset{(6.29)}{\leq} \exp(\alpha^2\sigma^2)
\end{aligned}$$

Now for the sum $\sum_{t=1}^{T}\delta_t$ we have that

$$\begin{aligned}
\mathbb{E}\left[\exp(\alpha\sum_{t=1}^{T}\delta_t)\right] &= \mathbb{E}\left[\exp(\alpha\sum_{t=1}^{T-1}\delta_t)\exp(\alpha\delta_T)\right] \\
&= \mathop{\mathbb{E}}_{\mathbf{x}_1,\ldots,\mathbf{x}_{T-1}}\left[\exp(\alpha\sum_{t=1}^{T-1}\delta_t)\mathop{\mathbb{E}}_{\mathbf{x}_T}[\alpha\delta_T|\mathbf{x}_1,\ldots,\mathbf{x}_{T-1}]\right] \\
&\overset{(6.30)}{\leq} \exp(\alpha^2\sigma^2)\,\mathbb{E}\left[\exp(\alpha\sum_{t=1}^{T-1}\delta_t)\right].
\end{aligned} \tag{6.31}$$

Having (6.31) for all T and $\mathbb{E}[\exp(\alpha\delta_1)] \leq \exp(\alpha^2\sigma^2)$ by induction we obtain that:

$$\mathbb{E}\left[\exp(\alpha\sum_{t=1}^{T}\delta_t)\right] \leq \exp(T\alpha^2\sigma^2). \tag{6.32}$$

Now applying the Markov's inequality we have that for all $\alpha > 0, M_\delta$:

$$p(\sum_{t=1}^{T}\delta_t \geq M_\delta) \leq \frac{\mathbb{E}\left[\exp(\alpha\sum_{t=1}^{T}\delta_t)\right]}{\exp(\alpha M_\delta)} \overset{(6.32)}{\leq} \frac{\exp(T\alpha^2\sigma^2)}{\exp(\alpha M_\delta)}. \tag{6.33}$$

Now for deriving bounds for our solution we have that for any $M > 0$:

$$\begin{aligned}
&P\left(F^{(k)}(\bar{\theta}_1^T) + G(\bar{\theta}_2^T) - F^{(k)}(\theta^*) - G(\theta^*) \geq \frac{(N^{1/p}dM_D + hD_{\mathcal{C}})^2}{2hT}\log(T+1) + \frac{\beta+h}{2T}D_{\mathcal{C}}^2 + \frac{M}{\sqrt{T}}\right) \\
&\overset{(6.25)}{\leq} P\left(\frac{1}{T}\sum_{t=1}^{T}\delta_t \geq \frac{M}{\sqrt{T}}\right) \\
&= P\left(\sum_{t=1}^{T}\delta_t \geq M\sigma\sqrt{T}\right) \\
&\overset{(6.33)}{\leq} \exp\left(-\frac{M^2}{4}\right),
\end{aligned} \tag{6.34}$$

where for deriving the last inequality we set $M_\delta = M\sigma\sqrt{T}$ and $\alpha = \frac{M}{2\sigma\sqrt{T}}$ in (6.33). Eq. (6.34) is equivalent to (6.17c), where we set

$$k_1 \triangleq \max\left(\frac{(N^{1/p}dM_D + hD_\mathcal{C})^2\log(3)}{2h\log(2)}, \frac{\beta+h}{2}D_\mathcal{C}^2\right) \tag{6.35a}$$

$$k_2 \triangleq M. \tag{6.35b}$$

6.4 Experiments

Algorithms. We run two variants of MBO; the first one, which we call `MBOSADM`, uses SADM (see Sec. 6.3) for solving the inner problems (6.6a). The second one, which we call `MBOSGD`, solves inner problems via a sub-gradient method. We also apply stochastic gradient descent with momentum directly to Prob. (6.3); we refer to this algorithm as `SGD`. This corresponds to the algorithm by [69], applied to our setting. We also solve the problem instances with an MSE objective using `SGD`, as the MSE is smooth and `SGD` is efficient in this case. Details about hyperparameters are given below.

For all algorithms we use a batch size of 8; we avoid using larger batch sizes as the computation time increases. Additional details and stopping criteria used for each algorithm are as follows:

- `MBOSADM`: We run Alg. 11 for 20 iterations, however we stop earlier if we do no see an improvement in the objective. In the k-th iteration of Alg. 11, for solving the sub-problems (Line 3), we run iterations of SADM (6.11) for a maximum of 200 rounds or until the primal and dual residuals are less then $0.95^k\epsilon$, where we set $\epsilon = 0.1$ and 0.001 for training auto-encoders and multi-target regression, respectively.

- `MBOSGD`: Again, we run Alg. 11 for 20 iterations and stop earlier if the objective does not improve. For solving the sub-problems (Line 3), we run the stochastic gradient descent (SGD) with the learning rate of 10^{-5} and 10^{-3}, respectively, for training auto-encoders and multi-target regression. In all cases we set the momentum parameter to 0.9 and run SGD for 500 iterations.

- `SGD`: We run the stochastic gradient descent (SGD) algorithm with momentum with the learning rate of 10^{-6} and the momentum parameter set to 0.9. This corresponds to the algorithm by Mai and Johanssen [69] applied to our setting. We run `SGD` for 10^3 and 10^4 iterations for training auto-encoders and multi-target regression, respectively; we observe that

		MBOSADM					MBOSGD					SGD			
P_{out}	p	F_{NOUTL}	F_{OBJ}	F_{TEST}	T(h)	T^*(h)	F_{NOUTL}	F_{OBJ}	F_{TEST}	T(h)	T^*(h)	F_{NOUTL}	F_{OBJ}	F_{TEST}	T(h)
MNIST															
0.0	2.0	**2.50**	**2.51**	**2.50**	5.69	0.14	8.08	8.08	8.12	64.17	6.47	9.21	9.22	9.30	9.83
0.0	1.5	**2.63**	**2.63**	**2.63**	11.67	0.79	20.19	20.20	20.39	65.67	59.98	20.35	20.36	20.57	14.71
0.0	1.0	**3.46**	**3.47**	**3.44**	17.82	3.09	102.79	102.80	104.24	81.53	NA	102.44	102.46	103.89	11.50
0.05	2.0	**3.48**	**5.35**	**3.46**	6.33	0.31	3.89	6.36	3.86	54.92	38.31	8.03	12.52	8.09	13.96
0.05	1.5	**4.10**	**9.74**	**4.08**	45.32	2.08	5.86	11.70	5.82	57.03	25.12	20.34	34.69	20.57	14.60
0.05	1.0	**5.23**	**20.20**	**5.24**	44.03	5.61	27.68	73.53	27.56	32.76	9.67	102.40	236.43	103.90	11.70
0.1	2.0	4.27	**7.77**	4.23	11.67	1.34	**3.56**	7.83	**3.54**	64.20	33.70	7.02	11.64	7.04	13.97
0.1	1.5	**4.18**	**11.84**	**4.17**	68.74	0.29	5.50	13.77	5.45	67.04	9.88	20.34	48.79	20.57	14.04
0.1	1.0	**5.90**	**36.02**	**5.92**	37.73	8.20	30.08	109.79	30.16	39.77	6.72	102.36	368.22	103.90	11.81
0.2	2.0	4.07	8.97	4.04	51.69	4.39	**3.54**	**8.23**	**3.52**	57.08	19.19	7.48	16.44	7.51	14.25
0.2	1.5	**3.90**	**11.58**	**3.89**	195.69	1.56	7.00	20.63	6.95	45.46	6.44	20.36	77.78	20.59	15.06
0.2	1.0	**3.85**	**28.25**	**3.83**	36.98	5.15	40.12	224.47	40.11	19.71	2.13	102.37	639.32	103.90	8.25
0.3	2.0	**3.99**	14.21	**3.98**	9.83	4.93	4.02	**10.55**	3.99	55.60	24.14	7.46	20.63	7.48	13.53
0.3	1.5	20.55	**22.56**	20.78	159.92	39.24	**7.22**	24.30	**7.16**	42.65	15.09	23.90	58.52	23.89	16.25
0.3	1.0	102.70	**99.36**	104.27	51.48	0.87	**56.60**	438.89	**56.17**	20.48	3.32	102.34	910.68	103.90	8.52
Fashion-MNIST															
0.0	2.0	**3.51**	**3.51**	**3.51**	4.33	0.31	5.01	5.01	5.01	42.13	14.80	8.72	8.73	8.70	9.78
0.0	1.5	**6.13**	**6.14**	**6.14**	14.35	2.18	8.87	8.88	8.89	63.39	12.80	22.62	22.63	22.56	14.70
0.0	1.0	**10.59**	**10.61**	**10.56**	29.69	2.63	41.24	41.26	41.35	50.41	3.32	224.26	224.28	224.89	9.72
0.05	2.0	**3.80**	**5.82**	**3.80**	14.70	1.40	4.53	6.75	4.54	67.29	19.15	8.30	11.98	8.27	9.71
0.05	1.5	**7.38**	14.57	**7.40**	96.73	2.56	7.91	**11.97**	7.93	64.25	16.16	20.88	27.22	20.83	10.94
0.05	1.0	**16.64**	**30.51**	**16.68**	43.55	16.04	65.01	109.94	65.31	29.60	9.70	158.65	227.48	158.27	9.97
0.1	2.0	**4.05**	**6.73**	**4.06**	14.66	2.35	4.28	7.90	4.29	65.06	16.46	8.96	14.35	8.94	13.67
0.1	1.5	11.08	32.69	11.10	20.07	NA	**8.46**	**15.23**	**8.49**	65.78	9.92	17.98	31.41	17.95	10.63
0.1	1.0	**9.79**	**27.96**	**9.81**	35.50	2.43	58.70	126.18	58.90	45.06	2.08	235.02	452.45	234.49	13.32
0.2	2.0	6.07	10.19	6.08	14.25	3.67	**4.77**	**9.28**	**4.77**	69.84	30.16	5.71	14.34	5.71	9.31
0.2	1.5	28.51	53.69	28.49	39.42	NA	**10.94**	**27.12**	**10.97**	39.11	16.09	19.36	42.97	19.36	10.87
0.2	1.0	**10.50**	**27.95**	**10.50**	94.72	6.57	140.00	390.02	140.08	17.03	3.33	204.88	644.99	205.13	14.72
0.3	2.0	6.63	23.18	6.63	32.87	NA	**5.84**	**13.04**	**5.85**	50.52	29.95	7.45	20.12	7.46	13.65
0.3	1.5	**7.08**	**22.51**	**7.10**	86.27	30.02	11.09	24.73	11.12	52.41	12.08	19.52	58.26	19.56	11.05
0.3	1.0	**14.43**	**50.91**	**14.46**	95.08	19.48	404.77	893.56	404.52	9.51	NA	410.82	522.50	411.84	10.74
SCMD1d															
0.0	2.0	2.88	2.88	3.02	1.82	0.12	**2.85**	**2.85**	**2.99**	0.36	0.04	3.62	3.63	3.72	1.37
0.0	1.5	4.23	4.24	**4.39**	7.22	0.43	**4.22**	**4.23**	4.44	0.36	0.04	5.47	5.47	5.60	1.58
0.0	1.0	**9.78**	**9.79**	**10.18**	7.13	0.47	9.86	9.86	10.32	0.37	0.04	12.95	12.95	13.25	1.22
0.05	2.0	2.88	3.13	2.99	2.52	0.13	**2.86**	**3.11**	3.00	0.54	0.05	3.64	3.89	3.71	1.31
0.05	1.5	4.23	4.61	**4.37**	10.23	4.61	**4.22**	**4.59**	4.46	0.50	0.05	5.50	5.87	5.61	1.23
0.05	1.0	**9.69**	**10.52**	**10.09**	0.59	0.18	9.86	10.66	10.35	0.51	0.05	13.03	13.87	13.29	1.17
0.1	2.0	2.90	3.41	3.01	2.22	0.13	**2.84**	**3.34**	**3.00**	0.46	0.05	3.63	4.12	3.69	1.30
0.1	1.5	4.23	4.99	4.42	9.42	0.68	**4.18**	**4.90**	**4.40**	0.50	0.05	5.52	6.11	5.62	1.15
0.1	1.0	**9.56**	**10.99**	**10.18**	9.92	0.78	9.77	11.11	10.54	0.54	0.07	13.09	13.72	13.32	1.11
0.2	2.0	2.93	3.90	3.03	1.83	0.95	**2.86**	**3.79**	**3.02**	0.5	0.3	3.63	3.97	3.66	1.15
0.2	1.5	4.23	5.60	**4.37**	8.17	3.60	**4.21**	**5.48**	4.47	0.36	0.2	5.50	5.83	5.56	1.17
0.2	1.0	**9.46**	**11.00**	**10.04**	6.55	1.50	9.82	11.30	10.60	0.45	0.20	13.09	13.38	13.28	1.11
0.3	2.0	2.93	4.32	3.03	1.80	NA	**2.85**	4.10	**3.03**	0.46	NA	3.61	**3.90**	3.64	1.18
0.3	1.5	4.25	6.05	**4.44**	8.19	NA	**4.21**	5.73	4.49	0.50	NA	5.43	**5.69**	5.43	1.18
0.3	1.0	**9.53**	11.07	**10.00**	6.44	2.72	9.68	**11.05**	10.32	0.51	0.28	12.95	13.13	12.96	1.11

Table 6.1: Time and Objective Performance. We report objective and time metrics for algorithms, under different outlier ratios and different p-norms. We observe from the table that MBOSADM significantly outperforms other competitors in terms of objective metrics. In terms of running time, SGD is generally fastest, due to cheap gradient updates. However, we see that the time MBO variants take to get to the same or better objective value (i.e., T^*), ware comparable to running time of SGD.

the algorithm achieves its minimum within this number of iterations. At each iteration, SGD evaluates the objective F_{OBJ} and outputs the solution for the best observed objective.

Implementation. We implement all the algorithms in Python 3.7 and using the PyTorch backend. We run all algorithms on CPU machines that have Intel(R) Xeon(R) CPUs (E5-2680 v4) with 2.4GHz clock speed.

Applications. For both applications, we set the regularizer as $g(\theta) = \frac{0.001}{2}\|\theta\|_2^2$ and do not consider a constraint set, i.e., $\mathcal{C} = \mathbb{R}^d$.

- **Training Autoencoders**. We use a neural network with two convolutional and two de-convolutional layers; the convolutional layers have 8 and 4 output channels, respectively, and 3×3 kernel weights. The de-convolutional layers exactly mirror the convolutional layers. We do not apply zero padding or dilation for any of the layers and use a convolution step size of 1. We apply a soft-plus activation function after each layer.

- **Multi-target Regression**. We use a network with two layers; the first layer is a 1-dimensional convolutional layer with the kernel size of 3 and no zero padding or dilation and the step size of 1. The second layer is a fully-connected layer with 278 hidden units and output size of 16 (the target size). We again apply the soft-plus activation after each layer.

Note that we choose soft-plus activation, i.e., a smooth version of the ReLu, to make sure that the functions $F(\theta; \mathbf{x})$ are smooth.

Datasets. For each of the applications we use two datasets:

- **Multi-Target Regression.** We use `SCM1d`, from the collection of regression data made available by [245]. `SCM1d`[1] is a supply chain management dataset comprising 9803 samples with 280 predictors and 16 targets. We use 80 percent of data for training, i.e., solving (6.3), and the rest of it as the test set.

- **Auto-encoders.** We use two well-known datasets `MNIST`[2] and `Fashion-MNIST`[3]. They both have greyscale 28×28 images with 60,000 training samples and 10,000 testing samples. `MNIST` contains handwritten digits with 10 classes and `Fashion-MNIST` contains images of clothing items from 10 classes.

Outliers. We denote the outliers ratio with $P_{\texttt{out}}$; each datapoint $\mathbf{x}_i$, $i \in [n]$, is independently corrupted with outliers with probability $P_{\texttt{out}}$. The probability $P_{\texttt{out}}$ ranges from 0.0 to 0.3 in our experiments. In particular, we corrupt training samples by replacing them with samples randomly

[1] http://users.auth.gr/espyromi/mtr/mtr-datasets.zip
[2] https://pytorch.org/vision/stable/datasets.htm#mnist
[3] https://pytorch.org/vision/stable/datasets.html#fashion-mnist

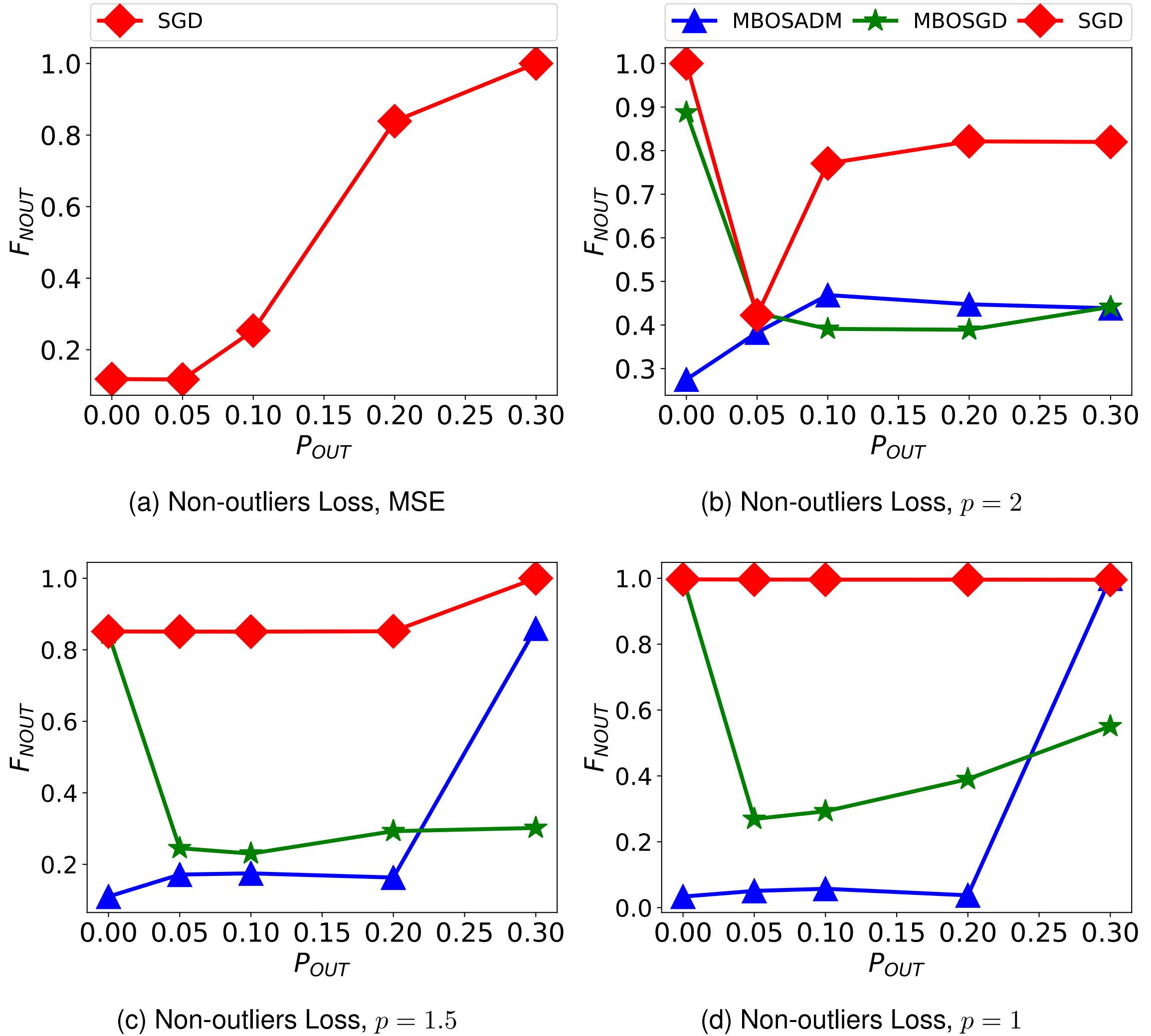
drawn from a Gaussian distribution whose mean is α away from the original data and its standard deviation equals that of the original dataset. For `MNIST` and `FashionMNIST`, we set α to 1.5 times the original standard deviation, while for `SCM1d`, we set α to 2.5 times the standard deviation.

Metrics. We evaluate the solution obtained by different algorithms by using the following three metrics. The first is F_{OBJ}, the regularized objective of Prob. (6.3) evaluated over the training set. The other two are: $F_{\mathtt{NOUTL}} \triangleq \frac{\sum_{i \notin \mathcal{S}_{\mathtt{OUTL}}} \|F(\theta;\mathbf{x}_i)\|_p}{n-|\mathcal{S}_{\mathtt{OUTL}}|}$, and $F_{\mathtt{TEST}} \triangleq \frac{\sum_{i \in \mathcal{S}_{\mathtt{TEST}}} \|F(\theta;\mathbf{x}_i)\|_p}{|\mathcal{S}_{\mathtt{TEST}}|}$, where $\mathcal{S}_{\mathtt{OUTL}}$, $\mathcal{S}_{\mathtt{TEST}}$ are the outlier and test sets, respectively. Metric $F_{\mathtt{NOUTL}}$ measures the robustness of algorithms w.r.t. outliers; ideally, $F_{\mathtt{NOUTL}}$ should remain unchanged as the fraction of outliers increases. Metric $F_{\mathtt{TEST}}$ evaluates the generalization ability of algorithms on unseen (test) data, which also does not contain outliers; ideally, $F_{\mathtt{TEST}}$ be similar $F_{\mathtt{NOUTL}}$. Moreover, we report total running time (T) of all algorithms. For the two variants of MBO, we additionally report the time (T^*) until the they reach the optimal value attained by `SGD` (N/A if never reached). Finally, for autoencoders, we also use dataset labels to train a logistic regression classifier over latent embeddings, and also report the prediction accuracy on the test set.

6.4.1 Time and Objective Performance Comparison

We evaluate our algorithms w.r.t. both objective and time metrics, which we report for different outlier ratios $P_{\mathtt{out}}$ and p-norms in Table 6.1. By comparing objective metrics, we see that `MBOSADM` and `MBOSGD` significantly outperform `SGD`. `SGD` achieves a better $F_{\mathtt{OBJ}}$ in only 2 out of 45 cases, i.e., `SCM1d` dataset for $p = 1.5, 2$ and $P_{\mathtt{out}} = 0.3$; however, even for these two cases, `MBOSADM` and `MBOSGD` obtain better $F_{\mathtt{NOUTL}}$ and $F_{\mathtt{TEST}}$ values. In terms of overall running time T, `SGD` is generally faster than `MBOSADM` and `MBOSGD`; this is expected, as each iteration of `SGD` only computes the gradient of a mini-batch of terms in the objective, while the other methods need to solve an inner-problem. Nonetheless, by comparing T^*, we see that the MBO variants obtain the same or better objective as `SGD` in a comparable time. In particular, T^* is less than T for `SGD` in 33 and 15 cases (out of 45) for `MBOSADM` and `MBOSGD`, respectively.

Comparing the performance between `MBOSADM` and `MBOSGD`, we first note that `MBOSADM` has a superior performance w.r.t. all three objective metrics for 25 out of 45 cases. In some cases, `MBOSADM` obtains considerably smaller objective values; for example, for `MNIST` and $P_{\mathtt{out}} = 0.0, p = 1$, $F_{\mathtt{NOUT}}$ is 0.03 of the value obtained by `MBOSGD` (also see Figures 6.2d and 6.3d). However, it seems that in the high-outlier setting $P_{\mathtt{out}} = 0.3$ the performance of `MBOSADM` deteriorates; this is mostly due to the fact that the high number of outliers adversely affects the convergence of SADM



(a) Non-outliers Loss, MSE

(b) Non-outliers Loss, $p = 2$

(c) Non-outliers Loss, $p = 1.5$

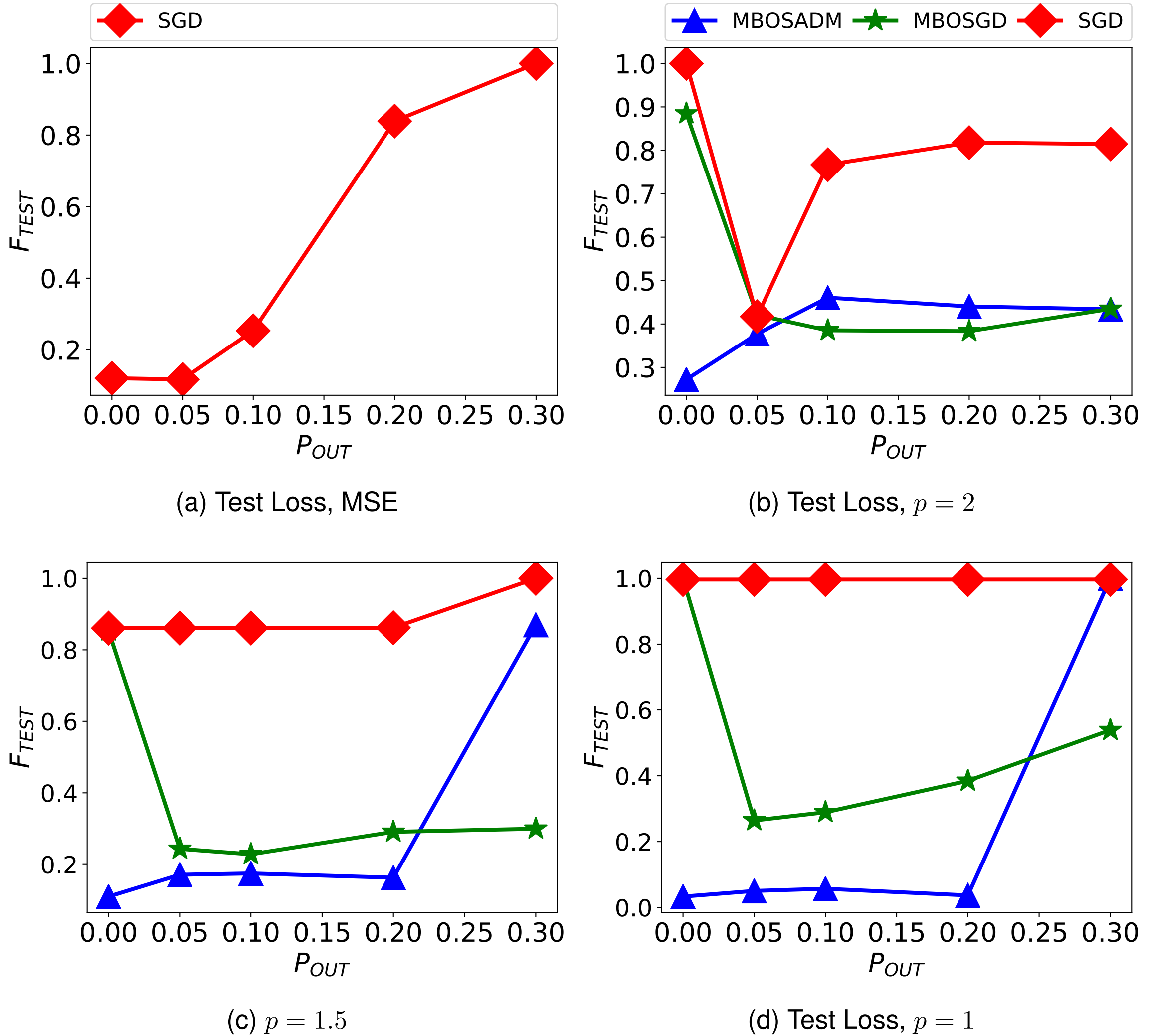
(d) Non-outliers Loss, $p = 1$

Figure 6.2: A comparison of scalability of the non-outliers loss $F_{\texttt{NOUT}}$ for different p-norms, w.r.t., outliers fraction $P_{\texttt{out}}$. We normalize values in each figure by the largest observed value, to make comparisons between different objectives possible. We see that MSE in Figures 6.2a and 6.3a are drastically affected by outliers and scale with outliers fraction $P_{\texttt{out}}$. Other ℓ_p norms for different methods in Figures 6.2b, 6.2c, and 6.2d generally stay unchanged w.r.t. $P_{\texttt{out}}$. However, MBOSADM in the high outlier regime and $p = 1, 1.5$ performs poorly.

and it takes more iterations to satisfy the desired accuracy.

6.4.2 Robustness Analysis

We further study the robustness of different p-norms and MSE to the presence of outliers. For brevity, we only report results for MNIST. We show the scaling of $F_{\texttt{NOUT}}$ and $F_{\texttt{TEST}}$ w.r.t. the



(a) Test Loss, MSE

(b) Test Loss, $p = 2$

(c) $p = 1.5$

(d) Test Loss, $p = 1$

Figure 6.3: A comparison of scalability of the test loss $F_{\texttt{TEST}}$ for different p-norms, w.r.t., outliers fraction $P_{\texttt{out}}$. We normalize values in each figure by the largest observed value, to make comparisons between different objectives possible. We see that MSE in Figure 6.3a is drastically affected by outliers and scale with outliers fraction $P_{\texttt{out}}$. Other ℓ_p norms for different methods in Figures 6.3b, 6.3c, and 6.3d generally stay unchanged w.r.t. $P_{\texttt{out}}$. However, `MBOSADM` in the high outlier regime and $p = 1, 1.5$ performs poorly.

fraction $P_{\texttt{out}}$ in Figures 6.3 and 6.2, respectively, for different norms. To make comparisons between different objectives interpretable, we normalize all values in each figure by the largest value in that figure.

By comparing Figures 6.2a and 6.3a, corresponding to MSE, with other plots in Fig. 6.2 and Fig. 6.3, we see that the loss values considerably increase by adding outliers. For other p-norms,

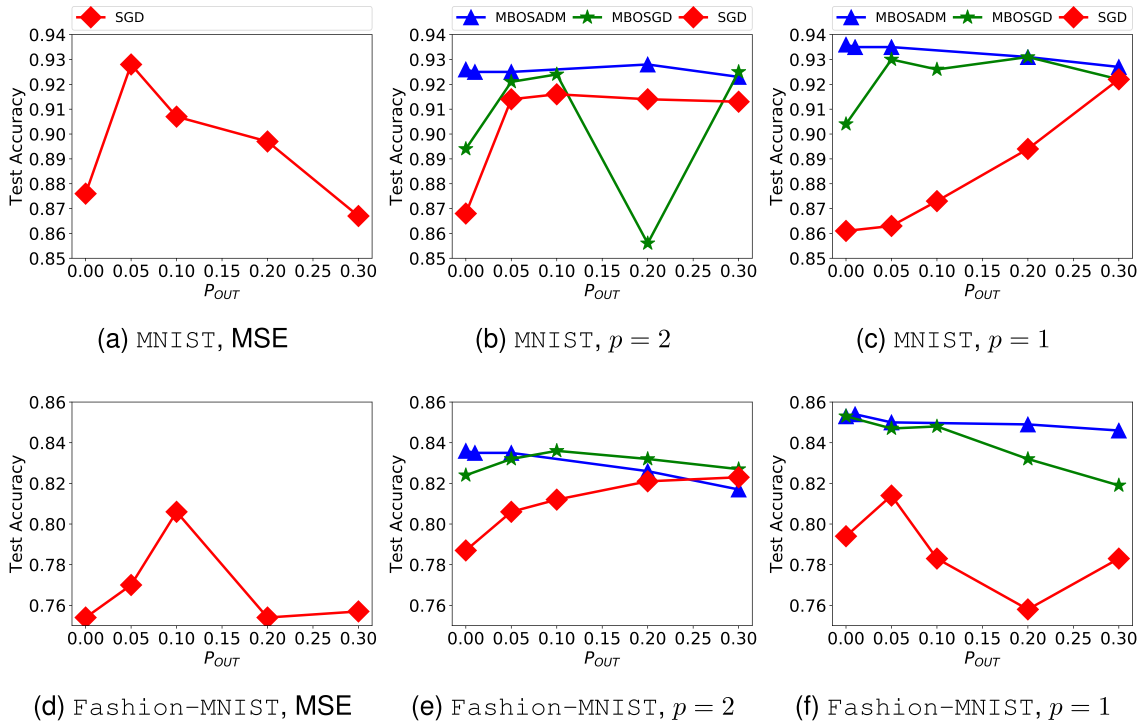


Figure 6.4: Classification performance for different methods and datasets. We use the embeddings obtained by auto-encoders trained via different algorithms to train a logistic regression model for classification. We generally observe that MBOSADM results in higher accuracy on the test sets. Moreover, we see that MSE is evidently sensitive to outliers, see Figures 6.4a and 6.4d for $P_{\text{out}} \geq 0.2$.

we see that SGD generally stays unchanged, w.r.t. outliers. However, the loss for SGD is higher than MBO variants. Loss values for MBOSGD also do not increase significantly by adding outliers. Moreover, we see that, when no outliers are present $P_{\text{out}} = 0.0$, MBOSGD obtains higher loss values. MBOSADM generally achieves the lowest loss values and these values again do not increase with increasing P_{out}; however, for the highest outliers ($P_{\text{out}} = 0.3$), the performance of MBOSADM is considerably worse for $p = 1$. As we emphasize in Sec. 6.4.1, high number of outliers adversely affects the convergence of SADM, and hence the poor performance of MBOSADM for $P_{\text{out}} = 0.3$.

6.4.3 Classification Performance

For autoencoder tasks, we train a logistic regression model, where the input features are the outputs of the first two convolutional layers used for encoding data. We set the parameters of the encoder to be the solutions obtained by the respective autoencoder training algorithm. Using the

P_{out}	p	Acc.		
		MBOSADM	MBOSGD	SGD
0.0	ℓ_2^2	-	-	0.876
0.0	2.0	0.926	0.894	0.868
0.0	1.5	0.930	0.882	0.861
0.0	1.0	**0.936**	0.904	0.861
0.05	ℓ_2^2	-	-	0.928
0.05	2.0	0.925	0.921	0.914
0.05	1.5	0.928	0.929	0.867
0.05	1.0	**0.935**	0.930	0.863
0.1	ℓ_2^2	-	-	0.907
0.1	2.0	0.925	0.924	0.916
0.1	1.5	0.927	0.930	0.877
0.1	1.0	**0.935**	0.926	0.873
0.2	ℓ_2^2	-	-	0.897
0.2	2.0	0.928	0.856	0.914
0.2	1.5	0.928	0.928	0.892
0.2	1.0	**0.931**	**0.931**	0.894
0.3	ℓ_2^2	-	-	0.867
0.3	2.0	0.923	0.925	0.913
0.3	1.5	0.927	**0.929**	0.914
0.3	1.0	0.927	0.922	0.918

Table 6.2: Accuracy of the classifiers for MNIST dataset.

encoder output for training sets and the corresponding target labels, we train the logistic regression, using the ℓ_2 regularizer. We train the latter with different regularizer coefficients (0.01, 0.1, and 1.0) and report the best observed accuracy on the test sets in Fig. 6.4. Full experimental results for MNIST and Fashion-MNIST are reported in Tables 6.2 and 6.3, respectively.

Fig. 6.4 shows the quality of the latent embeddings obtained by different trained autoencoders on the downstream classification over MNIST and FashionMNIST. Additional results are shown in Tables 6.2 and 6.3. We see that MBO variants again outperform SGD. For MNIST, (reported in Figures 6.4a to 6.4c), we see that MBOSADM for $p = 1$ obtains the highest accuracy. Moreover, for Fashion-MNIST (reported in Fig. 6.4d to 6.4f), we observe that again MBOSADM for $p = 1$ outperforms other methods. We also observe that MSE (reported in Figures 6.4a and 6.4d) are sensitive to outliers; the accuracy drastically drops for $P_{\text{out}} \geq 0.1$. An interesting observations is that adding outliers causes improvements in the performance of SGD; however, we see that SGD always results in lower accuracy, except in two cases ($P_{\text{out}} = 0.2$ in Fig. 6.4b and $P_{\text{out}} = 0.3$ in Fig. 6.4e).

P_{out}	p	Acc.		
		MBOSADM	MBOSGD	SGD
0.0	ℓ_2^2	-	-	0.754
0.0	2.0	0.836	0.824	0.787
0.0	1.5	0.848	0.843	0.793
0.0	1.0	**0.853**	**0.853**	0.794
0.05	ℓ_2^2	-	-	0.770
0.05	2.0	0.835	0.832	0.806
0.05	1.5	0.841	0.847	0.803
0.05	1.0	**0.850**	0.847	0.814
0.1	ℓ_2^2	-	-	0.806
0.1	2.0	0.835	0.836	0.812
0.1	1.5	0.837	0.846	0.811
0.1	1.0	**0.854**	0.848	0.783
0.2	ℓ_2^2	-	-	0.754
0.2	2.0	0.826	0.832	0.821
0.2	1.5	0.815	0.837	0.815
0.2	1.0	**0.849**	0.832	0.758
0.3	ℓ_2^2	-	-	0.757
0.3	2.0	0.817	0.827	0.823
0.3	1.5	0.838	0.837	0.826
0.3	1.0	**0.846**	0.819	0.783

Table 6.3: Accuracy of the classifiers for `Fashion-MNIST` dataset.

6.5 Conclusion

We present a generic class of robust formulations that includes many applications, i.e., auto-encoders, multi-target regression, and matrix factorization. We show that SADM, in combination with MBO, provides efficient solutions for our class of robust problems. Studying other proximal measures described by Ochs et al. [66] is an open area. Moreover, characterizing the sample complexity of our proposed method for obtaining a stationary point, as in MBO variants that use gradient methods [65, 64], is an interesting future direction.

Bibliography

[1] J. Bento and S. Ioannidis, "A family of tractable graph distances," in *SDM*, 2018.

[2] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*. Springer series in statistics New York, 2001, vol. 1, no. 10.

[3] C. M. Bishop, *Pattern recognition and machine learning*. Berlin, Heidelberg: Springer-Verlag, 2006.

[4] Y. Guo, P. Tian, J. Kalpathy-Cramer, S. Ostmo, J. P. Campbell, M. F. Chiang, D. Erdogmus, J. G. Dy, and S. Ioannidis, "Experimental design under the bradley-terry model." in *IJCAI*, 2018.

[5] T. Zhang and F. Oles, "The value of unlabeled data for classification problems," in *ICML*, 2000.

[6] M. Mahdian, A. Moharrer, S. Ioannidis, and E. Yeh, "Kelly cache networks," in *INFOCOM*, 2019.

[7] S. Ioannidis and E. Yeh, "Adaptive caching networks with optimality guarantees," in *SIGMETRICS*, 2016.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NeurIPS*, 2012.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[10] J. Tsitsiklis, D. Bertsekas, and M. Athans, "Distributed asynchronous deterministic and stochastic gradient optimization algorithms," *IEEE transactions on automatic control*, vol. 31, no. 9, pp. 803–812, 1986.

[11] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, “Scaling distributed machine learning with the parameter server.” in *OSDI*, vol. 14, 2014, pp. 583–598.

[12] D. Jakovetić, J. Xavier, and J. M. Moura, “Fast distributed gradient methods,” *IEEE Transactions on Automatic Control*, vol. 59, no. 5, pp. 1131–1146, 2014.

[13] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, “Distributed optimization and statistical learning via the alternating direction method of multipliers,” *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[14] R. Zhang and J. Kwok, “Asynchronous distributed ADMM for consensus optimization,” in *ICML*, 2014.

[15] A. Makhdoumi and A. Ozdaglar, “Convergence rate of distributed admm over networks,” *IEEE Transactions on Automatic Control*, vol. 62, no. 10, pp. 5082–5095, 2017.

[16] A. Bellet, Y. Liang, A. B. Garakani, M.-F. Balcan, and F. Sha, “A Distributed Frank-Wolfe Algorithm for Communication-Efficient Sparse Learning,” in *SDM*, 2015.

[17] A. Moharrer and S. Ioannidis, “Distributing Frank–Wolfe via map-reduce,” *Knowledge and Information Systems*, vol. 60, no. 2, pp. 665–690, 2019.

[18] K. L. Clarkson, “Coresets, sparse greedy approximation, and the Frank-Wolfe algorithm,” *ACM Transactions on Algorithms*, vol. 6, no. 4, pp. 63:1–63:30, 2010.

[19] S. Shalev-Shwartz, N. Srebro, and T. Zhang, “Trading accuracy for sparsity in optimization problems with sparsity constraints,” *SIAM Journal on Optimization*, vol. 20, no. 6, pp. 2807–2832, 2010.

[20] A. Moharrer, J. Gao, S. Wang, J. Bento, and S. Ioannidis, “Massively distributed graph distances,” *IEEE Transactions on Signal and Information Processing over Networks*, vol. 6, pp. 667–683, 2020.

[21] K. Kanellopoulos, N. Vijaykumar, C. Giannoula, R. Azizi, S. Koppula, N. M. Ghiasi, T. Shahroodi, J. G. Luna, and O. Mutlu, “Smash: Co-designing software compression and hardware-accelerated indexing for efficient sparse matrix operations,” in *MICRO*, 2019.

[22] J. Sherman and W. J. Morrison, "Adjustment of an inverse matrix corresponding to a change in one element of a given matrix," *The Annals of Mathematical Statistics*, vol. 21, no. 1, pp. 124–127, 1950.

[23] M. A. Woodbury, "The stability of out-input matrices," *Chicago, IL*, vol. 9, p. 1862, 1949.

[24] ——, *Inverting modified matrices*. Statistical Research Group, 1950.

[25] X. Liu, M. Smelyanskiy, E. Chow, and P. Dubey, "Efficient sparse matrix-vector multiplication on x86-based many-core processors," in *ICS*, 2013.

[26] N. Bell and M. Garland, "Efficient sparse matrix-vector multiplication on CUDA," Citeseer, Tech. Rep., 2008.

[27] J. Lafond, H.-T. Wai, and E. Moulines, "D-fw: Communication efficient distributed algorithms for high-dimensional sparse optimization," in *ICASSP*, 2016.

[28] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," in *EMNLP*, 2017.

[29] M. Frank and P. Wolfe, "An Algorithm for Quadratic Programming," *Naval Research Logistics Quarterly*, vol. 3, no. 1-2, pp. 95–110, 1956.

[30] M. Jaggi, "Revisiting Frank-Wolfe: Projection-free sparse convex optimization." in *ICML*, 2013.

[31] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge university press, 2004.

[32] T. Horel, S. Ioannidis, and S. Muthukrishnan, "Budget feasible mechanisms for experimental design," in *Latin American Symposium on Theoretical Informatics*. Springer, 2014, pp. 719–730.

[33] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets." in *HotCloud*, 2010.

[34] U. Feige, "A threshold of ln n for approximating set cover," *Journal of the ACM (JACM)*, vol. 45, no. 4, pp. 634–652, 1998.

[35] A. M. Frieze, "A cost function property for plant location problems," *Mathematical Programming*, vol. 7, no. 1, pp. 245–248, 1974.

[36] D. Kempe, J. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003, pp. 137–146.

[37] H. Lin and J. 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*, 2011, pp. 510–520.

[38] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions—I," *Mathematical Programming*, vol. 14, no. 1, pp. 265–294, Dec 1978.

[39] G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák, "Maximizing a submodular set function subject to a matroid constraint," in *International Conference on Integer Programming and Combinatorial Optimization*. Springer, 2007, pp. 182–196.

[40] ——, "Maximizing a monotone submodular function subject to a matroid constraint," *SIAM Journal on Computing*, vol. 40, no. 6, pp. 1740–1766, 2011.

[41] E. J. Rosensweig, J. Kurose, and D. Towsley, "Approximate models for general cache networks," in *INFOCOM, 2010 Proceedings IEEE*. IEEE, 2010, pp. 1–9.

[42] N. C. Fofack, P. Nain, G. Neglia, and D. Towsley, "Analysis of ttl-based cache networks," in *6th International ICST Conference on Performance Evaluation Methodologies and Tools*. IEEE, 2012, pp. 1–10.

[43] L. Dagum and R. Menon, "Openmp: An industry-standard api for shared-memory programming," *Computing in Science & Engineering*, 1998.

[44] W. Jiang, H. Gao, F.-l. Chung, and H. Huang, "The l2, 1-norm stacked robust autoencoders for domain adaptation," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[45] J. Mehta, K. Gupta, A. Gogna, A. Majumdar, and S. Anand, "Stacked robust autoencoder for classification," in *NeurIPS*, 2016.

[46] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, 2009.

[47] W. Waegeman, K. Dembczyński, and E. Hüllermeier, "Multi-target prediction: a unifying view on problems and methods," *Data Mining and Knowledge Discovery*, vol. 33, no. 2, pp. 293–324, 2019.

[48] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of machine learning research*, vol. 11, no. Dec, pp. 3371–3408, 2010.

[49] C. Ding, D. Zhou, X. He, and H. Zha, "R 1-PCA: rotational invariant l 1-norm principal component analysis for robust subspace factorization," in *ICML*, 2006.

[50] F. Nie, H. Huang, X. Cai, and C. H. Ding, "Efficient and robust feature selection via joint l2, 1-norms minimization," in *NIPS*, 2010.

[51] D. Kong, C. Ding, and H. Huang, "Robust nonnegative matrix factorization using l21-norm," in *Proceedings of the 20th ACM international conference on Information and knowledge management*. ACM, 2011.

[52] Q. Ke and T. Kanade, "Robust l/sub 1/norm factorization in the presence of outliers and missing data by alternative convex programming," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1. IEEE, 2005, pp. 739–746.

[53] M. Qian and C. Zhai, "Robust unsupervised feature selection," in *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

[54] L. Du, P. Zhou, L. Shi, H. Wang, M. Fan, W. Wang, and Y.-D. Shen, "Robust multiple kernel k-means using l21-norm," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[55] A. Eriksson and A. Van Den Hengel, "Efficient computation of robust low-rank matrix approximations in the presence of missing data using the l 1 norm," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 2010, pp. 771–778.

[56] C. Croux and P. Filzmoser, "Robust factorization of a data matrix," in *COMPSTAT*. Springer, 1998, pp. 245–250.

[57] N. Kwak, "Principal component analysis based on l1-norm maximization," *IEEE transactions on pattern analysis and machine intelligence*, vol. 30, no. 9, pp. 1672–1680, 2008.

[58] X. Li, Y. Pang, and Y. Yuan, "L1-norm-based 2dpca," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 40, no. 4, pp. 1170–1175, 2010.

[59] A. Baccini, P. Besse, and A. Falguerolles, "A l1-norm pca and a heuristic approach," *Ordinal and symbolic data analysis*, vol. 1, no. 1, pp. 359–368, 1996.

[60] S. Pesme and N. Flammarion, "Online robust regression via sgd on the l1 loss," in *NeurIPS*, 2020.

[61] A. S. Lewis and S. J. Wright, "A proximal method for composite minimization," *Mathematical Programming*, vol. 158, no. 1, pp. 501–546, 2016.

[62] D. Davis and B. Grimmer, "Proximally guided stochastic subgradient method for nonsmooth, nonconvex problems," *SIAM Journal on Optimization*, vol. 29, no. 3, pp. 1908–1930, 2019.

[63] D. Drusvyatskiy and A. S. Lewis, "Error bounds, quadratic growth, and linear convergence of proximal methods," *Mathematics of Operations Research*, vol. 43, no. 3, pp. 919–948, 2018.

[64] D. Davis and D. Drusvyatskiy, "Stochastic model-based minimization of weakly convex functions," *SIAM Journal on Optimization*, vol. 29, no. 1, pp. 207–239, 2019.

[65] D. Drusvyatskiy and C. Paquette, "Efficiency of minimizing compositions of convex functions and smooth maps," *Mathematical Programming*, vol. 178, no. 1, pp. 503–558, 2019.

[66] P. Ochs, J. Fadili, and T. Brox, "Non-smooth non-convex bregman minimization: Unification and new algorithms," *Journal of Optimization Theory and Applications*, vol. 181, no. 1, pp. 244–278, 2019.

[67] H. Wang and A. Banerjee, "Online alternating direction method," in *ICML*, 2012.

[68] J. Liu and J. Ye, "Efficient l1/lq NormRregularization," *arXiv preprint arXiv:1009.4766*, 2010.

[69] V. Mai and M. Johansson, "Convergence of a stochastic gradient method with momentum for non-smooth non-convex optimization," in *ICML*, 2020.

[70] D. P. Bertsekas, *Nonlinear programming*. Belmont, MA, USA: Athena scientific, 1999.

[71] J. L. W. V. Jensen *et al.*, "Sur les fonctions convexes et les inégalités entre les valeurs moyennes," *Acta mathematica*, vol. 30, pp. 175–193, 1906.

[72] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004.

[73] A. J. Kurdila and M. Zabarankin, *Convex functional analysis*. Springer Science & Business Media, 2006.

[74] J. Kiefer and J. Wolfowitz, "Stochastic estimation of the maximum of a regression function," *The Annals of Mathematical Statistics*, pp. 462–466, 1952.

[75] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, "Parallelized stochastic gradient descent," in *NIPS*, 2010.

[76] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *NIPS*, 2011.

[77] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *OSDI*, 2016.

[78] M. Li, L. Zhou, Z. Yang, A. Li, F. Xia, D. G. Andersen, and A. Smola, "Parameter server for distributed machine learning," in *NIPS Workshop*, 2013.

[79] M. Dudik, Z. Harchaoui, and J. Malick, "Lifted coordinate descent for learning with trace-norm regularization," in *AISTATS*, 2012.

[80] E. Hazan and S. Kale, "Projection-free online learning," in *ICML*, 2012.

[81] Y. Ying and P. Li, "Distance metric learning with eigenvalue optimization," *Journal of Machine Learning Research*, vol. 13, no. Jan, pp. 1–26, 2012.

[82] A. Joulin, K. Tang, and L. Fei-Fei, "Efficient image and video co-localization with Frank-Wolfe algorithm," in *ECVV*, 2014.

[83] D. Gabay and B. Mercier, *A dual algorithm for the solution of non linear variational problems via finite element approximation*. Institut de recherche d'informatique et d'automatique, 1975.

[84] ——, "A dual algorithm for the solution of nonlinear variational problems via finite element approximation," *Computers & mathematics with applications*, vol. 2, no. 1, pp. 17–40, 1976.

[85] R. Glowinski and A. Marroco, "Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de dirichlet non linéaires," *ESAIM: Mathematical Modelling and Numerical Analysis-Modélisation Mathématique et Analyse Numérique*, vol. 9, no. R2, pp. 41–76, 1975.

[86] T. Yang, "Trading computation for communication: Distributed stochastic dual coordinate ascent," in *NIPS*, 2013.

[87] K.-W. Chang, C. Hsieh, C. Lin, S. Keerthi, and S. Sundararajan, "A dual coordinate descent method for large-scale linear svm," in *Proc. Intl. Conf. on Machine Learning (ICML)*, 2008.

[88] S. Shalev-Shwartz and T. Zhang, "Stochastic dual coordinate ascent methods for regularized loss minimization," *Journal of Machine Learning Research*, vol. 14, no. Feb, pp. 567–599, 2013.

[89] Z.-Q. Luo and P. Tseng, "On the convergence of the coordinate descent method for convex differentiable minimization," *Journal of Optimization Theory and Applications*, vol. 72, no. 1, pp. 7–35, 1992.

[90] A. Schrijver, *Combinatorial optimization: polyhedra and efficiency*. Springer Science & Business Media, 2003, vol. 24.

[91] A. Krause and D. Golovin, "Submodular function maximization." *Tractability*, vol. 3, pp. 71–104, 2014.

[92] A. A. Bian, B. Mirzasoleiman, J. Buhmann, and A. Krause, "Guaranteed Non-convex Optimization: Submodular Maximization over Continuous Domains," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, A. Singh and J. Zhu, Eds., vol. 54. Fort Lauderdale, FL, USA: PMLR, 20–22 Apr 2017, pp. 111–120.

[93] A. Bian, K. Levy, A. Krause, and J. M. Buhmann, "Continuous dr-submodular maximization: Structure and algorithms," in *Advances in Neural Information Processing Systems*, 2017, pp. 486–496.

[94] Y. Bian, J. Buhmann, and A. Krause, "Optimal continuous DR-submodular maximization and applications to provable mean field inference," in *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[95] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[96] A. Bialecki, M. Cafarella, D. Cutting, and O. O?malley, "Hadoop: a framework for running applications on large clusters built of commodity hardware," 2005.

[97] H.-c. Yang, A. Dasdan, R.-L. Hsiao, and D. S. Parker, "Map-reduce-merge: simplified relational data processing on large clusters," in *SIGMOD*, 2007.

[98] R. Kumar, B. Moseley, S. Vassilvitskii, and A. Vattani, "Fast greedy algorithms in mapreduce and streaming," *ACM Transactions on Parallel Computing*, vol. 2, no. 3, p. 14, 2015.

[99] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii, "Scalable k-means++," *VLDB*, 2012.

[100] S. Suri and S. Vassilvitskii, "Counting triangles and the curse of the last reducer," in *WWW*, 2011.

[101] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun, "Map-reduce for machine learning on multicore," in *NIPS*, 2006.

[102] F. T. Leighton, *Introduction to parallel algorithms and architectures: Trees Hypercubes*. Elsevier, 2014.

[103] S. Sur, M. J. Koop, and D. K. Panda, "High-performance and scalable mpi over infiniband with reduced memory usage: an in-depth performance analysis," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. ACM, 2006, p. 105.

[104] N. L. Tran, T. Peel, and S. Skhiri, "Distributed frank-wolfe under pipelined stale synchronous parallelism," in *2015 IEEE International Conference on Big Data (Big Data)*, 2015.

[105] A. Tewari, P. K. Ravikumar, and I. S. Dhillon, "Greedy algorithms for structurally constrained high dimensional problems," in *NIPS*, 2011.

[106] I. W. Tsang, J. T. Kwok, and P.-M. Cheung, "Core vector machines: Fast svm training on very large data sets," *Journal of Machine Learning Research*, vol. 6, no. Apr, pp. 363–392, 2005.

[107] P. Wolfe, "Convergence theory in nonlinear programming," *Integer and nonlinear programming*, pp. 1–36, 1970.

[108] A. Beck and S. Shtern, "Linearly convergent away-step conditional gradient for non-strongly convex functions," *Mathematical Programming*, pp. 1–27, 2015.

[109] D. Garber and E. Hazan, "A linearly convergent variant of the conditional gradient algorithm under strong convexity, with applications to online and stochastic optimization," *SIAM Journal on Optimization*, vol. 26, no. 3, pp. 1493–1528, 2016.

[110] S. Lacoste-Julien and M. Jaggi, "On the global linear convergence of Frank-Wolfe optimization variants," in *NIPS*, 2015.

[111] Z. Harchaoui, M. Douze, M. Paulin, M. Dudik, and J. Malick, "Large-scale image classification with trace-norm regularization," in *CVPR*, 2012.

[112] D. Garber and O. Meshi, "Linear-memory and decomposition-invariant linearly convergent conditional gradient algorithm for structured polytopes," in *Advances in Neural Information Processing Systems*, 2016, pp. 1001–1009.

[113] G. Lan and Y. Zhou, "Conditional gradient sliding for convex optimization," *SIAM Journal on Optimization*, vol. 26, no. 2, pp. 1379–1409, 2016.

[114] E. Hazan and H. Luo, "Variance-reduced and projection-free stochastic optimization," in *ICML*, 2016.

[115] S. J. Reddi, S. Sra, B. Póczós, and A. Smola, "Stochastic Frank-Wolfe methods for non-convex optimization," in *Allerton*, 2016.

[116] J. C. Dunn, "Rates of convergence for conditional gradient algorithms near singular and nonsingular extremals," *SIAM Journal on Control and Optimization*, vol. 17, no. 2, pp. 187–211, 1979.

[117] M. Canon and C. Cullum, "A tight upper bound on the rate of convergence of frank-wolfe algorithm," *SIAM Journal on Control*, vol. 6, no. 4, pp. 509–516, 1968.

[118] J. Guélat and P. Marcotte, "Some comments on Wolfe's away step," *Mathematical Programming*, vol. 35, no. 1, pp. 110–119, 1986.

[119] P. Kumar and E. A. Yıldırım, "A linearly convergent linear-time first-order algorithm for support vector classification with a core set result," *INFORMS Journal on Computing*, vol. 23, no. 3, pp. 377–391, 2011.

[120] R. Ñanculef, E. Frandi, C. Sartori, and H. Allende, "A novel frank–wolfe algorithm. analysis and applications to large-scale svm training," *Information Sciences*, vol. 285, pp. 66–99, 2014.

[121] S. Damla Ahipasaoglu, P. Sun, and M. J. Todd, "Linear convergence of a modified frank–wolfe algorithm for computing minimum-volume enclosing ellipsoids," *Optimisation Methods and Software*, vol. 23, no. 1, pp. 5–19, 2008.

[122] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Advances in neural information processing systems*, 2013, pp. 315–323.

[123] M. Mahdavi, L. Zhang, and R. Jin, "Mixed optimization for smooth functions," in *Advances in Neural Information Processing Systems*, 2013, pp. 674–682.

[124] S. Lacoste-Julien, M. Jaggi, M. Schmidt, and P. Pletscher, "Block-coordinate frank-wolfe optimization for structural svms," in *Proceedings of ICML*, 2013.

[125] A. Osokin, J.-B. Alayrac, I. Lukasewitz, P. K. Dokania, and S. Lacoste-Julien, "Minding the Gaps for Block Frank-Wolfe Optimization of Structured SVMs," in *ICML*, 2016.

[126] A. Y. Ng, "Feature selection, l 1 vs. l 2 regularization, and rotational invariance," in *ICML*, 2004.

[127] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

[128] P. Shah, B. N. Bhaskar, G. Tang, and B. Recht, "Linear system identification via atomic norm regularization," in *CDC*, 2012.

[129] S. Chen and A. Banerjee, "Structured estimation with atomic norms: General bounds and applications," in *NIPS*, 2015.

[130] V. Chandrasekaran, B. Recht, P. A. Parrilo, and A. S. Willsky, "The convex geometry of linear inverse problems," *Foundations of Computational Mathematics*, vol. 12, no. 6, pp. 805–849, 2012.

[131] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, pp. 19:1–19:19, Dec. 2015.

[132] M. Lichman, "UCI machine learning repository," 2013.

[133] P. R. Goundan and A. S. Schulz, "Revisiting the greedy approach to submodular set function maximization," *Optimization online*, pp. 1–25, 2007.

[134] A. Krause and C. E. Guestrin, "Near-optimal nonmyopic value of information in graphical models," *arXiv preprint arXiv:1207.1394*, 2012.

[135] A. Krause and V. Cevher, "Submodular dictionary selection for sparse representation," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 567–574.

[136] A. Das and D. Kempe, "Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection," *arXiv preprint arXiv:1102.3975*, 2011.

[137] H. Lin and J. 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.

[138] B. Mirzasoleiman, A. Karbasi, R. Sarkar, and A. Krause, "Distributed submodular maximization: Identifying representative elements in massive data," in *Advances in Neural Information Processing Systems*, 2013, pp. 2049–2057.

[139] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions?i," *Mathematical Programming*, vol. 14, no. 1, pp. 265–294, 1978.

[140] Y. Bian, B. Mirzasoleiman, J. M. Buhmann, and A. Krause, "Guaranteed non-convex optimization: Submodular maximization over continuous domains," in *AISTATS*, 2017.

[141] J. Vondrák, "Optimal approximation for the submodular welfare problem in the value oracle model," in *Proceedings of the fortieth annual ACM symposium on Theory of computing*. ACM, 2008, pp. 67–74.

[142] G. L. Nemhauser and L. A. Wolsey, "Best algorithms for approximating the maximum of a submodular set function," *Mathematics of operations research*, vol. 3, no. 3, pp. 177–188, 1978.

[143] F. Bach, "Submodular functions: from discrete to continous domains," *arXiv preprint arXiv:1511.00394*, 2015.

[144] D. M. Topkis, "Minimizing a submodular function on a lattice," *Operations research*, vol. 26, no. 2, pp. 305–321, 1978.

[145] L. Lovász, "Submodular functions and convexity," in *Mathematical Programming The State of the Art*. Springer, 1983, pp. 235–257.

[146] F. P. Kelly, *Reversibility and stochastic networks*. Cambridge University Press, 2011.

[147] R. G. Gallager, *Stochastic processes: theory for applications*. Cambridge University Press, 2013.

[148] R. Nelson, *Probability, Stochastic Processes, and Queueing Theory: The Mathematics of Computer Performance Modeling*, 1st ed. Springer Publishing Company, Incorporated, 2010.

[149] H. Chen and D. D. Yao, *Fundamentals of queueing networks: Performance, asymptotics, and optimization*. Springer Science & Business Media, 2013, vol. 46.

[150] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *CoNEXT*, 2009.

[151] E. Yeh, T. Ho, Y. Cui, M. Burd, R. Liu, and D. Leong, "VIP: A framework for joint dynamic forwarding and caching in named data networks," in *ICN*, 2014.

[152] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *INFOCOM*, 2010.

[153] M. Dehghan, A. Seetharam, B. Jiang, T. He, T. Salonidis, J. Kurose, D. Towsley, and R. Sitaraman, "On the complexity of optimal routing and content caching in heterogeneous networks," in *INFOCOM*, 2014.

[154] N. Laoutaris, S. Syntila, and I. Stavrakakis, "Meta algorithms for hierarchical web caches," in *ICPCC*, 2004.

[155] H. Che, Y. Tung, and Z. Wang, "Hierarchical web caching systems: Modeling, design and experimental results," *Selected Areas in Communications*, vol. 20, no. 7, pp. 1305–1314, 2002.

[156] Y. Zhou, Z. Chen, and K. Li, "Second-level buffer cache management," *Parallel and Distributed Systems*, vol. 15, no. 6, pp. 505–519, 2004.

[157] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "Femtocaching: Wireless content delivery through distributed caching helpers," *IEEE Transactions on Information Theory*, vol. 59, no. 12, pp. 8402–8413, 2013.

[158] K. Naveen, L. Massoulié, E. Baccelli, A. Carneiro Viana, and D. Towsley, "On the interaction between content caching and request assignment in cellular cache networks," in *ATC*, 2015.

[159] K. Poularakis, G. Iosifidis, and L. Tassiulas, "Approximation caching and routing algorithms for massive mobile data delivery," in *GLOBECOM*, 2013.

[160] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," in *ICS*, 2002.

[161] E. Cohen and S. Shenker, "Replication strategies in unstructured peer-to-peer networks," in *SIGCOMM*, 2002.

[162] I. Baev, R. Rajaraman, and C. Swamy, "Approximation algorithms for data placement problems," *SIAM Journal on Computing*, vol. 38, no. 4, pp. 1411–1429, 2008.

[163] Y. Bartal, A. Fiat, and Y. Rabani, "Competitive algorithms for distributed data management," *Journal of Computer and System Sciences*, vol. 51, no. 3, pp. 341–358, 1995.

[164] L. Fleischer, M. X. Goemans, V. S. Mirrokni, and M. Sviridenko, "Tight approximation algorithms for maximum general assignment problems," in *SODA*, 2006.

[165] D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, and K. K. Ramakrishnan, "Optimal content placement for a large-scale VoD system," in *CoNEXT*, 2010.

[166] A. A. Ageev and M. I. Sviridenko, "Pipage rounding: A new method of constructing algorithms with proven performance guarantee," *Journal of Combinatorial Optimization*, vol. 8, no. 3, pp. 307–328, 2004.

[167] S. Ioannidis and E. Yeh, "Jointly optimal routing and caching for arbitrary network topologies," in *ICN*, 2017.

[168] D. P. Bertsekas, R. G. Gallager, and P. Humblet, *Data networks*. Prentice-Hall International New Jersey, 1992, vol. 2.

[169] C. Chekuri, J. Vondrak, and R. Zenklusen, "Dependent randomized rounding via exchange properties of combinatorial structures," in *FOCS*, 2010.

[170] H. L. Lee and M. A. Cohen, "A note on the convexity of performance measures of M/M/c queueing systems," *Journal of Applied Probability*, vol. 20, no. 4, p. 920?923, 1983.

[171] W. Grassmann, "The convexity of the mean queue size of the M/M/c queue with respect to the traffic intensity," *Journal of Applied Probability*, vol. 20, no. 4, p. 916?919, 1983.

[172] K. Kamran, A. Moharrer, S. Ioannidis, and E. Yeh, "Rate allocation and content placement in cache networks," in *IEEE International Conference on Computer Communications*, 2021.

[173] G. Özcan, A. Moharrer, and S. Ioannidis, "Submodular maximization via taylor series approximation," in *Proceedings of the 2021 SIAM International Conference on Data Mining*, 2021.

[174] J. A. Hartigan, *Clustering algorithms*. New York: Wiley, 1975.

[175] D. Conte, P. Foggia, C. Sansone, and M. Vento, "Thirty years of graph matching in pattern recognition," *International Journal of Pattern Recognition and Artificial Intelligence*, 2004.

[176] F. H. Allen, "The Cambridge Structural Database: a quarter of a million crystal structures and rising," *Acta Crystallographica Section B: Structural Science*, 2002.

[177] V. Kvasnička, J. Pospíchal, and V. Baláž, "Reaction and chemical distances and reaction graphs," *Theoretical Chemistry Accounts: Theory, Computation, and Modeling (Theoretica Chimica Acta)*, 1991.

[178] O. Macindoe and W. Richards, "Graph comparison using fine structure analysis," in *SocialCom*, 2010.

[179] D. Koutra, J. T. Vogelstein, and C. Faloutsos, "Deltacon: A principled massive-graph similarity function," in *SDM*, 2013.

[180] M. Berlingerio, D. Koutra, T. Eliassi-Rad, and C. Faloutsos, "Network similarity via multiple social theories," in *ASONAM*, 2013.

[181] D. Koutra, N. Shah, J. T. Vogelstein, B. Gallagher, and C. Faloutsos, "Deltacon: Principled massive-graph similarity function with attribution," *TKDD*, 2016.

[182] C. Chen, X. Yan, F. Zhu, and J. Han, "gapprox: Mining frequent approximate patterns from a massive network," in *ICDM*, 2007.

[183] H. Tong, C. Faloutsos, B. Gallagher, and T. Eliassi-Rad, “Fast best-effort pattern matching in large attributed graphs,” in *KDD*, 2007.

[184] F. Falchi, C. Gennaro, and P. Zezula, “A content–addressable network for similarity search in metric spaces,” in *Databases, Information Systems, and Peer-to-Peer Computing*, 2006.

[185] S. B. Roy, T. Eliassi-Rad, and S. Papadimitriou, “Fast best-effort search on graphs with multiple attributes,” *IEEE Trans. Knowl. Data Eng.*, 2015.

[186] K. Henderson, B. Gallagher, T. Eliassi-Rad, H. Tong, S. Basu, L. Akoglu, D. Koutra, C. Faloutsos, and L. Li, “Rolx: structural role extraction & mining in large graphs,” in *KDD*, 2012.

[187] K. L. Clarkson, “Nearest-neighbor searching and metric space dimensions,” *Nearest-Neighbor Methods for Learning and Vision: Theory and Practice*, 2006.

[188] C. K. L, “Nearest neighbor queries in metric spaces,” *Discrete & Computational Geometry*, 1999.

[189] A. Beygelzimer, S. Kakade, and J. Langford, “Cover trees for nearest neighbor,” in *ICML*, 2006.

[190] F. Angiulli and C. Pizzuti, “Fast outlier detection in high dimensional spaces,” in *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, 2002.

[191] M. R. Ackermann, J. Blömer, and C. Sohler, “Clustering for metric and nonmetric distance measures,” *ACM Transactions on Algorithms (TALG)*, 2010.

[192] R. R. Mettu and C. G. Plaxton, “Optimal time bounds for approximate clustering,” *Machine Learning*, vol. 56, no. 1-3, pp. 35–60, 2004.

[193] Y. Bartal, M. Charikar, and D. Raz, “Approximating min-sum k-clustering in metric spaces,” in *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, 2001.

[194] P. Indyk, “Sublinear time algorithms for metric space problems,” in *Proceedings of the thirty-first annual ACM symposium on Theory of computing*. ACM, 1999.

[195] S. Gold and A. Rangarajan, “A graduated assignment algorithm for graph matching,” *TPAMI*, 1996.

[196] L. Wiskott, J.-M. Fellous, N. Krüger, and C. Von Der Malsburg, "Face recognition by elastic bunch graph matching," in *CAIP*. Springer, 1997.

[197] G. W. Klau, "A new graph-based method for pairwise global network alignment," *BMC bioinformatics*, 2009.

[198] T. Goldstein, G. Taylor, K. Barabin, and K. Sayre, "Unwrapping admm: efficient distributed computing via transpose reduction," in *Artificial Intelligence and Statistics*, 2016.

[199] M. El-Kebir, J. Heringa, and G. W. Klau, "Natalie 2.0: Sparse global network alignment as a special case of quadratic assignment," *Algorithms*, 2015.

[200] M. Bayati, M. Gerritsen, D. F. Gleich, A. Saberi, and Y. Wang, "Algorithms for large, sparse network alignment problems," in *ICDM*, 2009.

[201] R. Singh, J. Xu, and B. Berger, "Pairwise global alignment of protein interaction networks by matching neighborhood topology," in *RECOMB*, 2007.

[202] V. Lyzinski, D. E. Fishkind, M. Fiori, J. T. Vogelstein, C. E. Priebe, and G. Sapiro, "Graph matching: Relax at your own risk," *TPAMI*, 2016.

[203] C. Schellewald and C. Schnörr, "Probabilistic subgraph matching based on convex relaxation," in *CVPR*, 2005.

[204] G. Chartrand, G. Kubicki, and M. Schultz, "Graph similarity and distance in graphs," *Aequationes Mathematicae*, 1998.

[205] L. Babai, "Graph isomorphism in quasipolynomial time," in *STOC*, 2016.

[206] B. Weisfeiler and A. Lehman, "A reduction of a graph to a canonical form and an algebra arising during this reduction," *Nauchno-Technicheskaya Informatsia*, 1968.

[207] P. Papadimitriou, A. Dasdan, and H. Garcia-Molina, "Web graph similarity for anomaly detection," *Internet Services and Applications*, 2010.

[208] S. Soundarajan, T. Eliassi-Rad, and B. Gallagher, "A guide to selecting a network similarity method," in *SDM*, 2014.

[209] M. R. Garey and D. S. Johnson, *Computers and Intractability*. WH Freeman New York, 2002.

[210] A. Fischer, C. Y. Suen, V. Frinken, K. Riesen, and H. Bunke, "Approximation of graph edit distance based on hausdorff matching," *Pattern Recognition*, 2015.

[211] H. Bunke and K. Shearer, "A graph distance metric based on the maximal common subgraph," *Pattern Recognition Letters*, 1998.

[212] H. Bunke, "On a relation between graph edit distance and maximum common subgraph," *Pattern Recognition Letters*, 1997.

[213] J. Koca, M. Kratochvil, V. Kvasnicka, L. Matyska, and J. Pospichal, *Synthon model of organic chemistry and synthesis design*. Springer Science & Business Media, 2012.

[214] B. J. Jain, "On the geometry of graph spaces," *Discrete Applied Mathematics*, 2016.

[215] K. Riesen and H. Bunke, "Approximate graph edit distance computation by means of bipartite graph matching," *Image and Vision Computing*, 2009.

[216] S. Fankhauser, K. Riesen, and H. Bunke, "Speeding up graph edit distance computation through fast bipartite matching," in *GBR*, 2011.

[217] K. Riesen, M. Neuhaus, and H. Bunke, "Graph embedding in vector spaces by means of prototype selection," in *GBR*, 2007.

[218] K. Riesen and H. Bunke, *Graph classification and clustering based on vector space embedding*. World Scientific, 2010.

[219] M. Ferrer, E. Valveny, F. Serratosa, K. Riesen, and H. Bunke, "Generalized median graph computation by means of graph embedding in vector spaces," *Pattern Recognition*, 2010.

[220] P. Zhu and R. C. Wilson, "A study of graph spectra for comparing graphs." in *BMVC*, 2005.

[221] R. C. Wilson and P. Zhu, "A study of graph spectra for comparing graphs and trees," *Pattern Recognition*, 2008.

[222] H. Elghawalby and E. R. Hancock, "Measuring graph similarity using spectral geometry," in *ICIAR*, 2008.

[223] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. Russell, "Distance metric learning with application to clustering with side-information," in *NIPS*, 2002.

[224] L. Page, S. Brin, R. Motwani, and T. Winograd, “The pagerank citation ranking: Bringing order to the web.” Stanford InfoLab, Tech. Rep., 1999, previous number = SIDL-WP-1999-0120.

[225] A. Grover and J. Leskovec, “node2vec: Scalable Feature Learning for Networks,” in *KDD*, 2016.

[226] W. Hamilton, Z. Ying, and J. Leskovec, “Inductive Representation Learning on Large Graphs,” in *NIPS*, 2017.

[227] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, “The Emerging Field of Signal Processing on Graphs: Extending High-Dimensional Data Analysis to Networks and Other Irregular Domains,” *IEEE Signal Processing Magazine*, vol. 30, no. 3, pp. 83–98, 2013.

[228] S. Sra, “Fast Projections Onto l1, q-norm Balls for Grouped Feature Selection,” in *ECML PKDD*, 2011.

[229] ——, “Fast projections onto mixed-norm balls with applications,” *Data Mining and Knowledge Discovery*, vol. 25, no. 2, pp. 358–377, 2012.

[230] N. Parikh and S. Boyd, “Proximal algorithms,” *Foundations and Trends in Optimization*, 2014.

[231] J. J. Moreau, “Décomposition orthogonale d’un espace hilbertien selon deux cônes mutuellement polaires,” *Comptes rendus hebdomadaires des séances de l’Académie des sciences*, vol. 255, pp. 238–240, 1962.

[232] S. R. Becker, E. J. Candès, and M. C. Grant, “Templates for convex cone problems with applications to sparse signal recovery,” *Mathematical programming computation*, 2011.

[233] C. Song, S. Yoon, and V. Pavlovic, “Fast admm algorithm for distributed optimization with adaptive penalty,” in *AAAI*, 2016.

[234] T.-H. Chang, M. Hong, W.-C. Liao, and X. Wang, “Asynchronous distributed alternating direction method of multipliers: Algorithm and convergence analysis,” in *ICASSP*, 2016.

[235] E. Wei and A. Ozdaglar, “Distributed alternating direction method of multipliers,” in *CDC*, 2012.

[236] Z. Xu, G. Taylor, H. Li, M. A. Figueiredo, X. Yuan, and T. Goldstein, “Adaptive consensus admm for distributed optimization,” in *ICML*, 2017.

[237] L. Majzoobi and F. Lahouti, "Analysis of distributed admm algorithm for consensus optimization in presence of error," in *ICASSP*, 2016.

[238] G. França and J. Bento, "An explicit rate bound for over-relaxed admm," in *Information Theory (ISIT), 2016 IEEE International Symposium on*. IEEE, 2016.

[239] T. Chang, M. Hong, and X. Wang, "Multi-agent distributed optimization via inexact consensus admm," *IEEE Transactions on Signal Processing*, 2015.

[240] M. Hong and Z.-Q. Luo, "On the linear convergence of the alternating direction method of multipliers," *Mathematical Programming*, 2017.

[241] R. J. Tibshirani, *The solution path of the generalized lasso*. Stanford University, 2011.

[242] C. Michelot, "A finite algorithm for finding the projection of a point onto the canonical simplex of n," *Journal of Optimization Theory and Applications*, 1986.

[243] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU press, 2013, vol. 3.

[244] J. L. Gustafson, "Reevaluating Amdahl's Law," *Communications of the ACM*, vol. 31, no. 5, pp. 532–533, 1988.

[245] E. Spyromitros-Xioufis, G. Tsoumakas, W. Groves, and I. Vlahavas, "Multi-target regression via input space expansion: treating targets as inputs," *Machine Learning*, vol. 104, no. 1, pp. 55–98, 2016.

[246] P. Paatero and U. Tapper, "Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values," *Environmetrics*, vol. 5, no. 2, pp. 111–126, 1994.

[247] C. Févotte and J. Idier, "Algorithms for nonnegative matrix factorization with the β-divergence," *Neural computation*, vol. 23, no. 9, pp. 2421–2456, 2011.

[248] H. Attouch, J. Bolte, P. Redont, and A. Soubeyran, "Proximal alternating minimization and projection methods for nonconvex problems: An approach based on the kurdyka-łojasiewicz inequality," *Mathematics of operations research*, vol. 35, no. 2, pp. 438–457, 2010.

[249] B. K. Natarajan, "Sparse approximate solutions to linear systems," *SIAM journal on computing*, vol. 24, no. 2, pp. 227–234, 1995.

[250] T. Blumensath and M. E. Davies, “Iterative hard thresholding for compressed sensing,” *Applied and computational harmonic analysis*, vol. 27, no. 3, pp. 265–274, 2009.

[251] J.-P. Vial, “Strong and weak convexity of sets and functions,” *Mathematics of Operations Research*, vol. 8, no. 2, pp. 231–259, 1983.

[252] J. C. Duchi and F. Ruan, “Stochastic methods for composite and weakly convex optimization problems,” *SIAM Journal on Optimization*, vol. 28, no. 4, pp. 3229–3259, 2018.

[253] H. Le, N. Gillis, and P. Patrinos, “Inertial block proximal methods for non-convex non-smooth optimization,” in *ICML*, 2020.

[254] P. Ochs and Y. Malitsky, “Model function based conditional gradient method with armijo-like line search,” in *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[255] H. Wang and A. Banerjee, “Online alternating direction method (longer version),” *arXiv preprint arXiv:1306.3721*, 2013.

[256] Y. Peng, A. Ganesh, J. Wright, W. Xu, and Y. Ma, “Rasl: Robust alignment by sparse and low-rank decomposition for linearly correlated images,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, no. 11, pp. 2233–2246, 2012.

[257] M. Tao and X. Yuan, “Recovering low-rank and sparse components of matrices from incomplete and noisy observations,” *SIAM Journal on Optimization*, vol. 21, no. 1, pp. 57–81, 2011.

[258] S. Zheng and J. T. Kwok, “Fast-and-light stochastic admm,” in *IJCAI*, 2016, pp. 2407–2613.

[259] Y. Liu, F. Shang, and J. Cheng, “Accelerated variance reduced stochastic admm,” in *AAAI*, 2017.

[260] H. Ouyang, N. He, L. Tran, and A. Gray, “Stochastic alternating direction method of multipliers,” in *International Conference on Machine Learning*. PMLR, 2013, pp. 80–88.

[261] T. Suzuki, “Dual averaging and proximal gradient descent for online alternating direction multiplier method,” in *ICML*, 2013.

[262] S. Hosseini, A. Chapman, and M. Mesbahi, “Online distributed admm via dual averaging,” in *CDC*, 2014.

[263] S. Shalev-Shwartz *et al.*, “Online learning and online convex optimization,” *Foundations and trends in Machine Learning*, vol. 4, no. 2, pp. 107–194, 2011.

[264] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro, “Robust stochastic approximation approach to stochastic programming,” *SIAM Journal on optimization*, vol. 19, no. 4, pp. 1574–1609, 2009.