Frameworks for High Dimensional Convex Optimization

Thesis by
Palma Alise den Nijs London

In Partial Fulfillment of the Requirements for the
Degree of
Doctor of Philosophy

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2020
Defended July 8, 2020

© 2020

Palma Alise den Nijs London
ORCID: 0000-0001-6472-8293

All rights reserved except where otherwise noted

ACKNOWLEDGEMENTS

I would like to thank my advisor, Adam Wierman, for his guidance and support throughout my graduate studies. I also thank my committee members, Steven Low, Yisong Yue, and Anima Anandkumar for their advice on various topics over the years.

I would also like to thank Shai Vardi for his guidance throughout the years of my Ph.D., which was instrumental in setting the course for various projects on which we worked.

I would like to thank Reza Eghbali, with whom I worked closely on several topics, and had numerous discussions which significantly aided me in developing my background in convex optimization.

I would like to express my sincere gratitude for Maryam Fazel, with whom I worked as an undergraduate at the University of Washington. This put me on the path to working on topics in convex optimization, and Maryam continued to help me develop my research background and offer research guidance during my graduate studies.

I am also grateful for guidance by Petros Drineas and Haim Avron in the later stages of my graduate studies.

Finally I would like to thank my parents for their unending encouragement.

ABSTRACT

We present novel, efficient algorithms for solving extremely large optimization problems. A significant bottleneck today is that as the size of datasets grow, researchers across disciplines desire to solve prohibitively massive optimization problems. In this thesis, we present methods to *compress* optimization problems. The general goal is to represent a huge problem as a smaller problem or set of smaller problems, while still retaining enough information to ensure provable guarantees on solution quality and run time. We apply this approach to the following three settings.

First, we propose a framework for accelerating both linear program solvers and convex solvers for problems with linear constraints. Our focus is on a class of problems for which data is either very costly, or hard to obtain. In these situations, the number of data points m available is much smaller than the number of variables, n. In a machine learning setting, this regime is increasingly prevalent since it is often advantageous to consider larger and larger feature spaces, while not necessarily obtaining proportionally more data. Analytically, we provide worst-case guarantees on both the runtime and the quality of the solution produced. Empirically, we show that our framework speeds up state-of-the-art commercial solvers by two orders of magnitude, while maintaining a near-optimal solution.

Second, we propose a novel approach for distributed optimization which uses far fewer messages than existing methods. We consider a setting in which the problem data are distributed over the nodes. We provide worst-case guarantees on the performance with respect to the amount of communication it requires and the quality of the solution. The algorithm uses $O(\log(n + m))$ messages with high probability. We note that this is an exponential reduction compared to the $O(n)$ communication required during each round of traditional consensus based approaches. In terms of solution quality, our algorithm produces a feasible, near optimal solution. Numeric results demonstrate that the approximation error matches that of ADMM in many cases, while using orders-of-magnitude less communication.

Lastly, we propose and analyze a provably accurate long-step infeasible Interior Point Algorithm (IPM) for linear programming. The core computational bottleneck in IPMs is the need to solve a linear system of equations at each iteration. We employ *sketching* techniques to make the linear system computation lighter, by handling well-known ill-conditioning problems that occur when using iterative solvers in IPMs for LPs. In particular, we propose a preconditioned Conjugate Gradient iterative solver for the linear system. Our sketching strategy makes the condition number of the preconditioned system provably small. In practice we demonstrate that our approach significantly reduces the condition number of the linear system, and thus allows for more efficient solving on a range of benchmark datasets.

PUBLISHED CONTENT AND CONTRIBUTIONS

[1] Palma London et al. “A Parallelizable Acceleration Framework for Packing Linear Programs”. In: *Proc. of Association for the Advancement of Artificial Intelligence*. 2018, pp. 3706–3713. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/viewFile/17118/16589.
Palma London is the lead author and contributed to all aspects of the paper.

[2] Palma London, Shai Vardi, and Adam Wierman. “Logarithmic Communication for Distributed Optimization in Multi-Agent Systems”. In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 3.3 (2019), pp. 1–29. DOI: 10.1145/3366696.
Palma London is the lead author and contributed to all aspects of the paper.

[3] Agniva Chowdhury et al. “Speeding up Linear Programming using Randomized Linear Algebra”. In: *arXiv:2003.08072* (2020). URL: https://arxiv.org/abs/2003.08072.
Palma London contributed to all aspects of the paper and led the numerical experiments.

[4] Palma London et al. “Black-box Acceleration of Convex Program Solvers”. In: *Submitted* (2020).
Palma London is the lead author and contributed to all aspects of the paper.

TABLE OF CONTENTS

Acknowledgements iii
Abstract iv
Published Content and Contributions vi
Table of Contents vii
List of Illustrations viii
List of Tables ix
Chapter I: Introduction: Compression of Optimization Problems 1
1.1 Settings of Interest 3
1.2 State-of-the-art Optimization Solvers and Current Bottlenecks 4
1.2.1 Optimization Solvers 4
1.2.2 Distributed Optimization 5
1.2.3 Interior Point Methods for Linear Programming 7
1.3 Contributions 7
1.3.1 Acceleration Method for Packing Optimization Problems 8
1.3.2 Distributed Algorithm for Multi-Agent Systems 10
1.3.3 Speeding up IPMs for Linear Programming using Randomization 11
Chapter II: Acceleration Algorithm for Packing Optimization with Linear Constraints 13
2.1 Problem Formulation 14
2.2 Related literature 15
2.3 A Black-box Acceleration Framework 18
2.3.1 Set Up 19
2.3.2 Acceleration Algorithm 20
2.4 Results: Feasibility and Optimality Guarantees 23
2.5 Experiments 25
2.5.1 Linear Case: Accelerating Gurobi 26
2.5.2 Convex Case: Accelerating SCS 28
2.5.3 Setting Parameters in Practice 31
2.5.4 The Benefits of Cloning 31
2.5.5 Case Study: California Road Network Dataset 33
2.6 Proofs 34
2.6.1 Preliminary Results 34
2.6.2 Quantizing the Solution Space 37
2.6.3 Feasibility 38
2.6.4 Optimality 40
2.7 Discussion 45
Chapter III: Distributed Algorithm with Logarithmic Communication Complexity 46

3.1 Motivation 46
3.1.1 Problem Overview and Distributed Setting 49
3.1.2 Approach 52
3.2 Related literature 52
3.3 Problem Formulation 55
3.3.1 Network Utility Maximization (NUM) 57
3.3.2 Support vector machines (SVMs) 58
3.4 A Local Optimization Framework 59
3.5 Theoretical Results 63
3.5.1 Results on communication complexity and solution quality 64
3.5.2 Additional Results and Proofs 66
3.6 Case Studies 68
3.6.1 Experimental Setup 68
3.6.2 Benchmark & Performance Metrics 70
3.6.3 Experimental Results 72
3.6.3.1 Linear Programming 72
3.6.3.2 Network Utility Maximization (NUM) 73
3.6.3.3 Support Vector Machines (SVMs) 73
3.6.3.4 Sparsity 76
3.6.3.5 Stragglers & Failures 76
3.7 Discussion 77
3.8 Proofs of Technical Results 78
3.8.1 Proof of Lemma 18 78
3.8.2 Proof of Lemma 22 79
3.8.3 Supplementary information 82
3.8.4 Pseudocode for General Online Fractional Packing . . . 82
3.8.5 ADMM 83
Chapter IV: Speeding up IPMs for Linear Programming using Randomization 84
4.1 Background on Interior Point Methods 84
4.1.1 Computational Bottleneck for IPMs 85
4.2 Setting of interest: under/over constrained LPs 86
4.2.1 Our contributions 87
4.2.2 Comparison with Related Work 90
4.3 Notation and Background 92
4.4 Conjugate Gradient Solver 92
4.5 The Infeasible IPM algorithm 94
4.6 Discussion and Extensions 98
4.7 Experiments 100
4.8 Proofs of Technical Results 105
4.8.1 Proof of Lemma 28 105
4.8.2 Satisfying eqn. (4.9) using CG Solver 105
4.8.3 Proof of Lemma 29 106
4.8.4 Proof of Lemma 30 107
4.9 Convergence Analysis of Infeasible IPM Algorithm 6 108

4.9.1 Number of Iterations for the CG Solver 108
4.9.2 Determining Step-size and Final Convergence 114
4.9.3 Proof of Theorem 27 124
Bibliography 125

LIST OF ILLUSTRATIONS

Number *Page*

21 Illustration of the relative error and speedup across sample sizes, ϵ_s. The shaded area depicts a reasonable setting range for ϵ_s; the speedup is significant, while the relative error is low. Two levels of sparsity, p, are shown. 27

22 Illustration of the relative error and runtime as the problem size, m, grows. Throughout, $n/m = 10^3$. In (b), Gurobi ran into memory errors for problems with $m > 300$. However, our accelerator algorithm was able to solve the problems in a faction of the time, with low relative error. 28

23 The relative error and runtime as the problem size, m, grows, where we fix the ratio $n/m = 10^3$. In (a), (b), $f_j(x_j) = c_j \log(x_j)$, and in (c), (d) $f_j(x_j) = c_j x_j^{1/2}$. 30

24 The relative error and runtime ratios as the sample size, ϵ_s, varies, where we fix $n = 5 \times 10^5$, $m = 100$. In (a), (b), $f_j(x_j) = c_j \log(x_j)$, and in (c), (d) $f_j(x_j) = c_j x_j^{1/2}$. 30

25 Illustration of how the optimal solution x^* and our approximate $x(\hat{\phi})$ solution differ. In particular, increasing ϵ_f affects the feasibility of our solution: comparing (a) and (b), as ϵ_f is increased, the approximate solution $x(\hat{\phi})$ shifts downwards; in (a) the approximate solution is infeasible, but setting $\epsilon_f = 0.1$ produces a feasible $x(\hat{\phi})$. Here $m = 100$ and $n = 600$ and $f_j(x_j) = c_j \log(x_j)$. 32

26 Illustration of the impact of cloning on solution quality as the number of clones grows. 32

27 Illustration of the relative error and runtime across sample sizes, ϵ_s, for the real data experiment on the California road network dataset. 34

31 The constraint matrix is depicted in (a), and the *hypergraph* $\mathcal{H}$ in (b). Red shaded nodes represent the primal variables and blue hyperedges represent the constraints, or dual variables. Hyperedges encircle primal variables which appear together in a constraint. 56

32 An illustration of LOCO. The constraint matrix A is depicted in (a), where shaded entries represent non-zeros. The rankings of the constraints are indicated next to their corresponding rows in A. The *hypergraph* $\mathcal{H}$ is depicted in (b). Figures (c)-(h) illustrate the construction of sets X_1 and Y_1 for the local problem associated with variable x_1. The darkest shaded matrix elements in (c), (e), and (g) indicate constraints as they are received by agent 1. Blue emboldened hyperedges in (d), (f), and (h) represent constraints added to Y_1. Red shaded nodes represent variables added to set X_1. The process stops in (g) because rankings $0.1 < 0.3$. The local problem associated with variable x_1 is defined on variables $X_1 = \{1, 2, 3, 4\}$ and constraints, or dual variables, $Y_1 = \{1, 2, 3\}$. 60

33 Messages required by LOCO and ADMM for random linear programming instances. Plots (a) and (b) vary n while fixing sparsity $p = 10^{-4}$, showing the results in linear-scale and log-scale respectively. Plots (c) and (d) fix $n = 10^3$ and vary the sparsity p. 70

34 Comparison of the relative error and messages required by LOCO and ADMM for random linear programming instances. Plots (a) and (b) show the Pareto optimal curve for ADMM for two different settings of the relative tolerance parameter: $\epsilon^{rel} = 10^{-4}$ and $\epsilon^{rel} = 10^{-1}$ respectively. 71

35 Illustration of the number of messages required by LOCO and ADMM for NUM using an Autonomous System (AS) graph. . . 72

36 Plot (a) illustrates of the number of messages required by LOCO and the relative error between LOCO and ADMM, in the case of synthetic SVM data, when $n = 10,000$ and m is varied, and $p = 0.03\%$. In all instances, the number of messages required by ADMM was over $100K$, an order of magnitude larger than LOCO, and are not plotted due to being out of range for the plot. Plot (b) illustrates the speedup provided by LOCO compared to ADMM. 'Max' and 'Med' refer to the largest and median sized subproblem respectively. 74

37 Comparison of the local problem dimension ($|X_k| \times |Y_k|$) to the original problem size ($m \times n$), averaged over all the local problems, in the case of synthetic SVM data. In (a) $n = m = 10,000$, in (b) $n = 5,000$ and $m = 10,000$. 74

38 Illustration of the impact of stragglers. The plots illustrate how the speedup of LOCO relative to ADMM varies with (a) the problem dimension n, when the Pareto shape parameter is set to 5, and (b) the shape parameter of the straggler distribution. Synthetic SVM data is used with $n = m = 10,000$ and $p = 0.03\%$. 77

41 *ARCENE data set:* Our Algorithm 6 (Sk. IPM) requires an order of magnitude fewer (a) inner iterations than the Standard IPM with CG, at each outer iteration, due to the improved (b) conditioning of $Q^{-1/2}AD^2A^{\mathsf{T}}Q^{-1/2}$ compared to AD^2A^T. 99

42 *ARCENE data set*: for various (w, *tolCG*) settings, (a) the maximum number of inner iterations used by our algorithm and (b) the maximum condition number of $Q^{-1/2}AD^2A^{\mathsf{T}}Q^{-1/2}$, across outer iterations. The standard IPM, across all settings, needed on the order of 1,000 iterations and $\kappa(AD^2A^T)$ was on the order of 10^8. The relative error was fixed to 0.04%. 99

43 *DEXTER data set*: Our algorithm (Sk. IPM) requires an order of magnitude fewer inner iterations than the Standard IPM with CG, at each outer iteration, as demonstrated in (a). This is possible due to the improved conditioning of $Q^{-1/2}AD^2A^{\mathsf{T}}Q^{-1/2}$ compared to AD^2A^T, demonstrated in (b). For all, *tolCG* $= 10^{-5}$, $\tau = 10^{-9}$. 103

44 *ARCENE data set*: As w increases, (a) the number of inner iterations decreases, and is relatively robust to *tolCG*, and, (b) the condition number decreases as well. 103

LIST OF TABLES

Number *Page*

31 SVM on CCAT and C11 from Reuters RCV1 75

41 Comparison of (our) sketched IPM with CG, standard IPM with CG, and Standard IPM with a direct solver, for the ℓ_1-SVM problem on UCI Machine Learning Repository [67] data sets. Across all, $\tau = 10^{-9}$ and a relative error of 10^{-3} or less was achieved. We define $\kappa_{\text{Sk}} = \kappa(Q^{-1/2}AD^2A^\mathsf{T}Q^{-1/2})$ and $\kappa_{\text{Stan}} = \kappa(AD^2A^T)$. 104

Chapter 1

INTRODUCTION: COMPRESSION OF OPTIMIZATION PROBLEMS

As the size of datasets grow, researchers across disciplines desire to solve extremely large optimization problems. In these settings, practical applications require optimization algorithms to work at extreme scales, and despite decades of work, existing solvers do not scale as desired in many cases.

In this thesis, we present novel, efficient algorithms for extremely large optimization problems. A significant bottleneck today is that massive amounts of data are being gathered, leading to prohibitively massive optimization problems that are often impossible to solve in real time. In this thesis, we seek to understand if such seemingly huge problems are always inherently huge; perhaps a problem has a related, smaller approximate representation, which can be used to arrive at the optimal solution.

We present methods to **compress optimization problems**, and introduce novel algorithms which make use of this compression. The general goal is to represent a given optimization problem as a smaller problem, making it more amenable to efficient solving, while still retaining enough information to ensure a highly accurate solution. Our algorithms have provable guarantees on running time and solution quality. To make this goal possible, we make use of two strategies simultaneously: (1) develop **randomization** techniques and (2) leverage **characteristics of real datasets** to design novel algorithms. These strategies are outlined below:

First, given a huge optimization problem, can the large problem be rewritten as a smaller, representative problem? These smaller problems can be designed by using statistical properties of the data, in order to identify truly influential variables and constraints in the underlying optimization problem. Identifying such a useful random subset is nontrivial; the subset must be informative, however, the work required to make the selection must be computationally light. To tackle these challenges, we additionally develop the following strategy.

Second, to tackle the prohibitive computational challenge presented by extremely large datasets, it is increasingly necessary to develop algorithms that are specifically designed for various types of data, rather than apply a generic algorithm. We develop **structure specific algorithms** that leverage **commonly occurring characteristics of real datasets** to motivate the design of faster, memory efficient, provably optimal algorithms. For example, the constraint matrix in a Linear Program might be very short-and-fat, or it might be sparse, having few non-zeros. These characteristics occur overwhelmingly often in practice and are easy to identify. Thus, when confronted with a particular problem, a practitioner can quickly decide which algorithm is most appropriate for the problem.

We bring together the above strategies by determining how the structure of the problem data can motivate the identification of an influential random subset of the data or variables, used to design a "compressed" problem.

In this thesis, we focus on algorithms for both Linear Programs (LPs) and problems with a convex objective with linear constraints. We address various computational settings. In particular, the above strategies motivate algorithms that are amenable to **distributed and parallel computation**.

In what follows in this chapter, we provide background on the type of optimization problems we address here, give background on the current state of optimization solvers in both general and distributed settings, and then describe our contributions.

1.1 Settings of Interest

Throughout this thesis, we focus on problems of the following form: convex problems with *linear constraints*. Specifically, our algorithms apply to problems of the following form:

$$\begin{aligned} &\text{maximize} && \textstyle\sum_{j=1}^{n} f_j(x_j) && && \text{(1.1a)} \\ &\text{subject to} && \textstyle\sum_{j=1}^{n} a_{ij}x_j \le b_i && i \in [m] && \text{(1.1b)} \end{aligned}$$

where a_{ij} is an element of matrix $A \in \mathbb{R}^{(m \times n)}$, $b \in \mathbb{R}^m$, and $f_j : \mathbb{R} \to \mathbb{R}$ are continuous concave functions differentiable. An important special case of the above are Linear Programs (LPs), where $c \in \mathbb{R}^n$:

$$\begin{aligned} &\text{maximize} && \textstyle\sum_{j=1}^{n} c_j x_j && && \text{(1.2a)} \\ &\text{subject to} && \textstyle\sum_{j=1}^{n} a_{ij}x_j \le b_i && i \in [m] && \text{(1.2b)} \end{aligned}$$

Solving LPs is broadly applicable in mathematics, science, and engineering. Many problems in machine learning and inference are LPs problems at their core. For example, LP problems include ℓ_1 regularized support vector machines (SVM) [212], ℓ_1 regression or least absolute deviations (LAD) [198], basis pursuit [204], nonnegative matrix factorization [168, 80], sparse inverse covariance matrix estimation [209], Markov Decision Processes [21], applications of graphical models [166], and relaxations of maximum a posteriori (MAP) estimation problems [178, 133]. Providing fast solvers for these problems is crucial in practice. The design of very fast solvers is thus well-motivated, both from a theoretical and a practical perspective.

The more general problem of form (1.1) appears often in practice as well. In this work, we in particular focus on this problem in a distributed setting. For example, consider Network Utility Maximization (NUM) [206], which is a general class of optimization problems that has seen widespread applications in multi-agent systems, from the design of TCP congestion control [99, 119, 120, 186] to understanding of protocol layering as optimization decomposition [44, 156] and power system demand response [177, 114].

Linear support vector machines (SVMs) [53, 95, 93] are also of form (1.1). Of recent interest is the framework of federated machine learning, in which one seeks to train machine learning models in settings where data is distributed among multiple agents due to privacy concerns. This approach has received significant attention from researchers in recent years, e.g., [102], and appears in industry as well, e.g., [130].

Additional examples include distributed inference in sensor networks (which has broad applications to the Internet of Things [87, 158, 152, 85]), inference in graphical models [166, 5], relaxations of maximum a posteriori estimation problems [178], management of content distribution networks and data centers [28, 154], and control of power systems [71, 160].

In some results we focus on *covering or packing problems*. These are problems for which the input data is non-negative: $A \in \mathbb{R}_{\geq 0}^{m \times n}$, $b \in \mathbb{R}_{\geq 0}^{m}$, and $c \in \mathbb{R}_{\geq 0}^{n}$. Many problems in machine learning, inference, and resource allocation are packing problems at their core. Of the problems listed above, NUM [206] and maximum a posteriori estimation problems [178] are often packing problems, as well as channel transmission [96], inference problems in biology [128], scheduling and graph embedding [163], and associative Markov networks [190], the maximum cut problem [194], zero-sum matrix games [146], flow controls [19], auction mechanisms [213], wireless sensor networks [37], and many other areas.

Given the prevalence of LPs and problems of form (1.1) in practice, we focus on developing algorithms for these problems.

1.2 State-of-the-art Optimization Solvers and Current Bottlenecks

In this section we describe the state of current linear and convex solvers. We highlight several core computational challenges that are associated with these methods. In the following chapters, we present novel approaches to tackle each of these challenges.

1.2.1 Optimization Solvers

There has been considerable work over the past decades in developing linear and more generally, convex program solvers. These solvers are generally either interior-point methods (IPMs) or first order (gradient-based) methods. For linear programing, commercial software like Cplex and Gurobi internally use both the simplex method and interior-point methods. For convex solvers, interior-point methods are used in most off-the-shelf public software packages like SDPT3 [193] and SeDuMi [188], as well as the commercial software package MOSEK [9] and the more recent embedded conic solver ECOS [62]. However, the computational cost of first- and second-order methods used in some of the above packages is often prohibitive for high dimensional problems.

Prohibitively large datasets. As large amounts of data are gathered and stored, we desire to solve extremely large problems. On one hand, there is continual improvement in computing power. However, rather than solving large problems using brute force computational power, we can try to determine if seemingly huge problems can perhaps have smaller representations.

Black-box methods to accelerate existing solvers. Given the wealth of algorithms that exist, the pertinent goal today may not be to *replace* these algorithms, but rather to make sure we are using them in a smart way. In other words, an already existing algorithm can be used as a *black-box within a more complex framework.* The key challenge here is to identify how to design that framework. As motivated in the beginning of this chapter, we use easy-to-identify characteristics of a given huge problem in order to design smaller, related problems, that can be feed into already existing algorithms in a black-box fashion. In Chapter 2 we describe a novel method to accelerate a family of linear and convex solvers.

1.2.2 Distributed Optimization

Distributed optimization is a computing paradigm of major importance across fields. Beginning in the 1960s approaches emerged for solving large scale linear programs via decomposition into pieces that could be solved in a distributed manner. For example, two early approaches are Bender's decomposition [22] and the Dantzig-Wolfe decomposition [56, 57], which can both be generalized to nonlinear objectives via the subgradient method [23, 143, 182]. Today, there are a wide variety of approaches for distributed optimization, e.g., primal decomposition [105, 23], dual decomposition [74, 31, 143, 99, 119, 186, 191], subgradient methods [140, 142], and proximal gradient descent methods [181], to name a few.

Distributed Settings and Communication Complexity Challenges. Despite the extensive literature on distributed optimization algorithms, computational and communication challenges remain. As the size of datasets grow, it is essential to develop distributed methods that are computationally light at each node, require minimal communication between nodes, and are robust to link failures.

The distributed methods described above often employ *consensus* schemes as a mechanism for distributing the computation among the processing units, forming the basis for many first order and second order distributed optimization algorithms, e.g., [27, 141]. For example, ADMM is a popular dual decomposition method, introduced by [77] that can be implemented in a consensus setting [31].

In the consensus setting, distributed agents pass current estimates of the global solution between agents or a master node, gradually improving those estimates at each step with the goal of reaching global consensus or convergence to an optimal solution. The end goal is for each distributed agent to have a complete solution, $x \in \mathbb{R}^n$. Thus, in such approaches, the distributed agents at each step are required to store, update, and broadcast a vector of dimension that matches that of the full system-wide solution to the problem. However, in some problem settings, this approach can be overkill. For example, in *multi-agent systems*, typically an agent's final goal is only to compute its local variable, or piece of the solution in order to determine a local action. Thus, each agent computing the entire full global solution, $x \in \mathbb{R}^n$ is unnecessary. Further, no individual agent can determine its own action or estimate without global convergence of all agents in the network. In our work described in Chapter 3, we take advantage of this change of perspective, and design a distributed approach that uses far less communication that existing approaches.

Often, the more traditional distributed paradigms have been applied to multi-agent system settings. For example, distributed optimization has been used in the design of multi-agent systems in the following settings: the management of content distribution networks and data centers [28, 154], communication network protocol design [99, 120, 186], trajectory optimization [88, 104], formation control of vehicles [189, 165], sensor networks [152, 115], control of power systems [71, 160], and management of electric vehicles and distributed storage devices [79]. Additionally, recently such approaches have become prominent in the emerging field of federated machine learning [102, 130], where data is distributed across a set of agents and the goal of the agents is to train a model using the full data set without sharing data between them.

In Chapter 3, we propose a novel distributed algorithm that uses far less communication than existing approaches.

1.2.3 Interior Point Methods for Linear Programming

IPMs work by searching the interior of the feasible region in order to get close to an optimal vertex on the boundary. At each iteration, a new search direction and step length is computed. For a detailed discussion on IPMs, see Section 4.1. To guide the choice of new search direction, the algorithm strives to satisfy the optimality conditions of the problem. This computation involves solving a linear system.

The core computational bottleneck in IPMs is the need to solve the linear system of equations at each iteration. This leads to two key challenges: first, for high-dimensional matrices A, solving the linear system is computationally prohibitive. Most implementations of IPMs use a *direct solver*; see Chapter 6 of [149]. However, if A large and dense, direct solvers are computationally impractical. Thus, a natural alternative is to use iterative solvers rather than direct solvers. However, a second challenge arrises: the matrix involved in the normal equations, AD^2A^T, is typically ill-conditioned. Here, $D = X^{\frac{1}{2}}S^{-\frac{1}{2}}$, and $X, S \in \mathbb{R}^{n\times n}$ are diagonal matrices having i-th diagonal elements equal to i-th components of x and s respectively for $i = 1, 2, \ldots, n$, and $p \in \mathbb{R}^m$. In particular, due to the nature of the complementary slackness conditions of the LP, the diagonal matrix D grows increasingly ill-conditioned as IPM algorithms approach the optimal primal-dual solution.

Thus, a key challenge for large scale IPMs for LPs is to develop techniques that can handle very large datasets, and are robust to the ill-conditioning issue of the linear system.

1.3 Contributions

Here we summarize the work presented in this thesis. First, we include a brief outline of the contributions, and in the remainder of the section we present a detailed summary of each contribution.

(1) We propose a framework for accelerating exact and approximate convex programming solvers for packing problems of form (1.1), of which an important special case is linear programing (1.2). Analytically, we provide worst-case guarantees on both the runtime and the quality of the solution produced. Empirically, we show that our framework speeds up Gurobi and SCS by two orders of magnitude, while maintaining a near-optimal solutions. This work is described in Chapter 2 and [117].

(2) We propose a novel approach for distributed optimization which uses far fewer messages than existing methods. Problems of form (3.1) are considered, in which the problem data, A and b, are distributed over the nodes. We provide worst-case guarantees on the performance with respect to the amount of communication it requires and the quality of the solution. This work is described in Chapter 3 and [116].

(3) We propose and analyze a provably accurate long-step infeasible IPM algorithm for LPs. We employ *sketching* [201] techniques to make the computation lighter, and to handle well-known conditioning problems that occur when using iterative solvers in IPMs for LPs. This work is described in Chapter 4 and [46].

Below we include an in-depth summary of each of the three contributions listed above.

1.3.1 Acceleration Method for Packing Optimization Problems

We propose a framework for accelerating exact and approximate convex programming solvers for packing problems of form (1.1). The approach is not to design a *new* algorithm, but to design a black-box acceleration framework that can speed up *existing* algorithms. Given a convex program solver $\mathcal{A}$ and a problem of dimension $(m \times n)$, we select a subset of the variables of size $\epsilon_s n$ uniformly at random. We then construct a *sample problem* of dimension $(m \times \epsilon_s n)$, defined only on those selected variables. Thus, the number of variables in the sample problem is greatly reduced compared to the original formulation. We solve the dual of the sample problem using solver $\mathcal{A}$, treating it as a black box. Finally, we set the values of the original primal variables x_j for all $j \in [n]$ based on the approximate dual solution.

As described above, this algorithm is designed for problems for which $m \ll n$. Intuitively, the larger n is compared to m, the more amenable the problem is to subsampling the columns of the matrix A. We formalize this idea by quantifying two fundamental tradeoffs in the framework. The first is captured by the sample size, ϵ_s. Setting ϵ_s to be small yields a dramatic speedup of the algorithm $\mathcal{A}$; however, if ϵ_s is set too small the quality of the solution suffers. A second tradeoff involves feasibility. In order to ensure that the output of the framework is feasible with high probability, the constraints of the sample problem are scaled down by a factor denoted by ϵ_f. Feasibility is guaranteed if ϵ_f is large enough; however, if it is too large, the quality of the solution (as measured by the approximation ratio) suffers.

Our main technical result is a worst-case characterization of the impact of ϵ_s and ϵ_f on the speedup provided by the framework and the quality of the solution. Assuming that algorithm $\mathcal{A}$ gives an α–approximation to the optimal solution of the dual, we prove that the acceleration framework guarantees a $(1 - \epsilon_f)/\alpha^2$–approximation to the optimal solution of the original problem, under some assumptions about the input and ϵ_f. We formally state the result in Theorem 1. We note here that the solution quality does not depend on ϵ_s, which highlights that the framework maintains a high-quality approximation even when sample size is small.

The technical requirements for ϵ_f in Theorem 1 impose some restrictions on both the family of problems that can be provably solved using our framework and the algorithms that can be accelerated. In particular, Theorem 1 requires $\min_i b_i$ to be large and the algorithm $\mathcal{A}$ to satisfy approximate complementary slackness conditions (see Section 2.3). While the condition on b_i is restrictive, the condition on the algorithms is satisfied by most common solvers, e.g., exact solvers and many primal dual approximation algorithms. Further, our experimental results demonstrate that these technical requirements are conservative; the framework produces solutions of comparable quality to the original solver in settings that are far from satisfying the theoretical requirements. In addition, the accelerator works in practice for algorithms that do not satisfy approximate complementary slackness conditions, e.g., for gradient algorithms as in [185]. In particular, our experimental results show that the accelerator obtains solutions that are close in quality to those obtained by the algorithms being accelerated on the complete problem, *and* that the solutions are obtained considerably faster (by up to two orders of magnitude). The results reported in this thesis demonstrate this by accelerating the state-of-the-art commercial solver Gurobi, and the SCS solver, on a wide array of randomly generated packing LPs and CPs and obtaining solutions with $< 4\%$ relative error and a more than $150\times$ speedup.

1.3.2 Distributed Algorithm for Multi-Agent Systems

We propose a new approach for distributed optimization in multi-agent systems that reduces the communication overhead of traditional approaches, while also guaranteeing robustness to communication delay and failures in the system. We consider problems of form (1.1) (defined more specifically as problem (3.1) in Section 3), where the problem is defined over a network. Each node j is associated with a local variable, $x_j \in \mathbb{R}^{q_i}$, and a local function, f_j. The problem data, A and b, are distributed over the nodes. Nodes may communicate with neighbors, but no central communication is performed.

As described in Section 1.2.2, work on distributed algorithms has often focused on a distributed setting in which each node is required to compute the full solution $x \in \mathbb{R}^n$. However, in many applications it is not necessary to acquire the entire solution at each node. For example, an agent at a node may be interested only in computing its local action. In a distributed model fitting setting, training data may be acquired and stored at different nodes, but each node may desire to learn about only a subset of the features.

Thus we consider a muti-agent distributed setting in which the jth node is only concerned about solving for its local variable, x_j. Our results highlight that this change of focus enables an exponential reduction in communication. In particular, our approach requires sending only (1) minimal data between nodes; each node does not have to see the entire A matrix, and (2) the number of messages is low and thus link failures in the system only affect a small number of node. More concretely, our approach is as follows.

We introduce a fundamentally new approach to distributed constrained optimization: LOCO (LOcal Convex Optimization). For each node in the network, we define a *local problem* associated with variable x_j, which is defined on a subset X_j of the primal variables and a subset Y_j of the data, or constraints. Each node j solves its local problem using a given algorithm $\mathcal{A}$, producing an approximation of its associated variable, x_j. Sets X_j and Y_j are much smaller in size than the dimensions of the original problem, n and m respectively, resulting in a dramatic dimension reduction and thus light computation when solving each local problem at each node. When the data matrix A is sparse, X_j and Y_j both have sizes on the order of $O(\log m)$, where the sparsity of the matrix appears in a constant.

We provide worst-case guarantees on the performance of LOCO with respect to the amount of communication it requires and the quality of the solution. Regarding communication, the process of determining sets X_j and Y_j requires $O(\log(n+m))$ messages with high probability. After this step, solving the local problems at each node requires no communication. Note that this is an exponential reduction compared to the $O(n)$ communication required during *each round* of traditional approaches such as consensus and dual descent, even when A is sparse.

We also provide worst-case guarantees on the performance of LOCO with respect to the quality of the solution. Since the nodes do not have access to the entire problem under LOCO, it is unreasonable to expect an exact solution. Instead, LOCO produces a feasible, α-approximation of the optimal solution, where α depends on the given algorithm $\mathcal{A}$ used to solve the local problem of an agent (Theorem 19). Our numeric results in Section 3.6 highlight that the approximation error of LOCO matches that of ADMM in many cases, while using orders-of-magnitude less communication.

This approach is based on an area of theoretical computer science: *local computation algorithms* [174]. A key result from this literature is that online algorithms can be converted into local algorithms with the same performance guarantee in *graph problems* with bounded degree [126, 169]. For the first time, we connect these techniques to *optimization problems*, where the bounded degree property can be interpreted as the sparsity of the constraint matrix.

Our framework is well suited for solving problems over a network where link failures are common. If a link fails, or if a node goes offline while a dual decomposition algorithm is executing, the process is brought to a halt. With LOCO, failures only affect a small number of nodes, making it robust to network failures. Similarly, our framework is robust to dynamic changes in the network; an arrival of a new node in the network only requires a few messages and the recomputation of the variables associated with nearby nodes. In contrast, a dual decomposition algorithm typically requires recomputing the entire solution.

1.3.3 Speeding up IPMs for Linear Programming using Randomization

As briefly described in Section 1.2.3, the core computational bottleneck in IPMs is the need to solve a linear system of equations at each iteration. This leads to two key challenges: for high-dimensional matrices A, solving the linear system is computationally prohibitive, and requires an iterative solver rather than a direct solver. However, a second challenge arrises: the matrix involved in the normal equations, AD^2A^T, is typically ill-conditioned.

We address both challenges by developing a preconditioner for AD^2A^T using matrix sketching, which allows us to prove strong convergence guarantees for the *residual error* of Conjugate Gradient (CG) solvers. First, this can be presented as an independent result; we propose and analyze a preconditioned CG iterative solver for the normal equations (see eqn. (4.4a)). Our sketching strategy will make the condition number of the pre-conditioned system *provably* small. This is key for developing an algorithm that works well in practice.

Second, we propose and analyze a provably accurate long-step *infeasible* IPM algorithm. The proposed IPM solves the normal equations using iterative solvers. In this thesis, for brevity and clarity, we primarily focus our description and analysis on the CG iterative solver. We note that a non-trivial concern is that the use of iterative solvers and matrix sketching tools implies that the normal equations at each iteration will be solved only approximately. In our proposed IPM, we develop a novel way to *correct* for the error induced by the approximate solution in order to guarantee convergence. Importantly, this correction step is relatively computationally light, unlike a similar step proposed in [137].

We empirically show that our algorithm performs well in practice. We consider solving LPs that arise from ℓ_1-regularized SVMs and test them on a variety of synthetic and real datasets. Several extensions of our work are discussed in Section 4.6.

Our approach is designed for the special case where $m \ll n$, i.e., the number of constraints is much smaller than the number of variables; see Section 4.6 for a generalization. This is a common setting in ML applications of LP solvers, since ℓ_1-SVMs and basis pursuit problems often exhibit such structure when the number of available features (n) is larger than the number of objects (m). We can also handle the case when $m \gg n$, by simply taking the dual of the original problem.

Chapter 2

ACCELERATION ALGORITHM FOR PACKING OPTIMIZATION WITH LINEAR CONSTRAINTS

In this chapter we propose a black-box framework that can be used to accelerate both exact and approximate convex programming (CP) solvers for packing problems. We exploit characteristics of the problem structure to enable these solvers to run in a fraction of the original time [117].

As summarized in Section 1.3.1, our framework can speed up *existing* algorithms. Given a convex program solver $\mathcal{A}$ and a problem of dimension $(m \times n)$, we select a subset of the variables of size $\epsilon_s n$ uniformly at random. We then construct a *sample problem* of dimension $(m \times \epsilon_s n)$, defined only on those selected variables. Thus, the number of variables in the sample problem is greatly reduced compared to the original formulation. We then solve the dual of the sample problem using solver $\mathcal{A}$, treating it as a black box. Finally, we set the values of the original primal variables x_j for all $j \in [n]$ based on the approximate dual solution.

In particular, our approach is designed for class of problems for which data is either very costly, or hard to obtain. In these situations $m \ll n$; i.e., the number of data points m available is much smaller than the number of variables, n. In a machine learning setting, this regime is increasingly prevalent since it is often advantageous to consider larger and larger feature spaces, while not necessarily obtaining proportionally more data. Such instances are also common in areas such as genetics, astronomy, and chemistry. There has been considerable research focusing on this class of problems in recent years, in the context of LPs [64, 26] and also more generally in convex optimization and compressed sensing [39, 63], low rank matrix recovery [167, 38], and graphical models [208, 134].

2.1 Problem Formulation

Our focus is on convex problems with *packing constraints*. Specifically, our framework applies to problems of the following form:

$$\begin{aligned} \text{maximize} \quad & \textstyle\sum_{j=1}^{n} f_j(x_j) && && \text{(2.1a)} \\ \text{subject to} \quad & \textstyle\sum_{j=1}^{n} a_{ij} x_j \le b_i && i \in [m] && \text{(2.1b)} \\ & 0 \le x_j \le 1 && j \in [n] && \text{(2.1c)} \end{aligned}$$

where $a_{ij} \in [0,1]$ is an element of matrix A of size $(m \times n)$, $b \in \mathbb{R}^m_{\ge 0}$, and $f_j : [0,1] \to \mathbb{R}$ are continuous concave nondecreasing functions differentiable over $(0,1)$. We further assume that $f_j(0) = 0$. Note that if this is not the case, the functions can be appropriately shifted. An important special case of the above are packing linear programs (LP):

$$\begin{aligned} \text{maximize} \quad & \textstyle\sum_{j=1}^{n} c_j x_j && && \text{(2.2a)} \\ \text{subject to} \quad & \textstyle\sum_{j=1}^{n} a_{ij} x_j \le b_i && i \in [m] && \text{(2.2b)} \\ & 0 \le x_j \le 1 && j \in [n] && \text{(2.2c)} \end{aligned}$$

where $c \in \mathbb{R}^n_{\ge 0}$. In this chapter, we develop a general algorithm for problems of form (2.1), and an algorithm specific to packing linear programs (2.2).

Many problems in machine learning, inference, and resource allocation are packing problems at their core. Providing fast solvers for these problems is crucial, e.g., the network utility maximization problem and large scale wireless networks [181], channel transmission [96], inference problems in biology [128], scheduling and graph embedding [163], and associative Markov networks [190]. Packing linear programs arise in a wide variety of settings, including the maximum cut problem [194], zero-sum matrix games [146], flow controls [19], auction mechanisms [213], wireless sensor networks [37], and many other areas. In machine learning, they show up in an array of problems, e.g., in applications of graphical models [166], and in relaxations of maximum a posteriori (MAP) estimation problems [178], among others.

Below we discuss related approaches.

2.2 Related literature

The approach underlying our framework is motivated by recent work that uses ideas from online algorithms to make offline algorithms more scalable, e.g., [127, 118, 116], and builds on our preliminary work [117]. A specific inspiration for this work is an online algorithm [4] that uses a two step procedure: it solves an LP based on a subsampled constraint matrix, acquired during the first stages of the online algorithm. It then uses the LP solution as the basis of a rounding scheme in later stages. The algorithm only works when the arrival order is random, which is analogous to sampling in the offline setting. However, [4] rely on exactly solving the sample LP acquired in the first stages; considering approximate solutions of the sampled problem, as we do, adds complexity to the algorithm and analysis. Also, unlike their approach, we leverage the offline setting to fine-tune ϵ_f in order to optimize our solution while ensuring feasibility, which is not possible in the online setting. Additionally, we extend these ideas about LPs to convex problems of form (3.1).

In general, our work is related to the literature on online algorithms for packing LPs and convex problems under the random permutation model. As discussed above, [4] present an online algorithm for packing LPs. Subsequently, algorithms for online packing LPs and various generalizations to the convex case have been studied by [100, 89, 90, 3], and [69]. However, these algorithms either require an update of the dual variable at each online step, or solve the primal problem to optimality at each step. Neither of those approaches can be adapted to design an acceleration framework for the convex case in the way that we adapt the work of [4] to design an algorithm for LPs as discussed above. Hence, the design presented for the convex case deviates significantly from the online algorithms literature.

The sampling phase of our framework is reminiscent of the method of *sketching*; see [201] and references therein. To construct a sketch of a matrix, the matrix is pre-multiplied by a random matrix, essentially selecting a subset of the measurements, or rows of the matrix. We however have a different goal: to select a subset of the variables, or columns of the matrix. While sketching is designed for solving overdetermined regression problems for which $m \gg n$, we consider the $m \ll n$ setting, where there are relatively few measurements, and we would like to consider a subset of the variables.

Additionally, the second step of our algorithm is a departure from the sketching approach. In sketching, the smaller problem associated with the sketched matrix is solved, producing an approximate primal solution. The guarantees produced are often in the form of bounds on norms of the sketched matrix, which translate directly to results about the optimally of the solution. This is possible because the objective function depends only on norms involving the data to be sketched: A and b. For example in linear regression, the objective function is $\|Ax - b\|_2^2$. Sketching results typically produce bounds on $\|S(Ax - b)\|_2^2$, where S is the sketching matrix. These bounds translate directly to the quality of the approximation of the solution. However in more general problems, the objective function does not conveniently depend on a norm of the sketched matrix. For example, in linear programs the objective function is $c^T x$; new analysis is needed to account for how the c vector affects the quality of the solution, independently from the subsampled data matrix. In our approach we develop a second step, in which we assign the primal variables of the original problem based on the dual solution to the sampled problem. This assignment is dependent on the objective function. No such analogy is present in sketching methods. Thus despite the similarity to sketching in our first step, sketching results do not apply here.

The sampling phase of the framework is also reminiscent of the *experiment design* problem, in which the goal is to solve the least squares problem using only a subset of available data while minimizing the error covariance of the estimated parameters, see e.g. [32]. Recent work by [172] applies these ideas to online algorithms, when collecting data for regression modeling. Like sketching, experiment design is applied in the overdetermined setting, whereas we consider the underdetermined scenario. Additionally, instead of sampling constraints, we sample variables.

The second step of our algorithm is a thresholding step, which is related to the rich literature of LP rounding, see [25] for a survey. Typically, rounding is used to arrive at a solution to an ILP; however we use thresholding to “extend” the solution of a sampled problem to the original problem. The scheme we use is a deterministic threshold based on complementary slackness conditions. It is inspired by [4] in the LP case, but adapted here for more general problems, and extended to handle approximate solvers rather than exact solvers. Within the rounding LP literature, the most related recent work is that of [185], which proposes a scheme for rounding an approximate LP solution. However, [185] use all of the problem data during the approximation step, whereas we show that it is enough to use a (small) sample of the data.

A key feature of our framework is that it can be parallelized easily when used to accelerate a distributed or parallel algorithm. There is a rich literature on distributed and parallel LP solvers, e.g., [205, 150, 36, 171]. More specifically, there is significant interest in distributed strategies for approximately solving covering and packing linear problems, such as the problems we consider here, e.g., [121, 207, 19, 12, 6].

Our algorithm is related to *column generation*, a heuristic technique developed for LPs with an extremely large number of variables; see [145, 58] for a survey. In column generation, a series of smaller problems, all defined on a subset of the variables, are solved cyclically. More specifically, in column generation a subset of the variables is used to define a smaller *master* problem. The dual solution of the master problem is used to define a new *subproblem*, the solution to which will help identify a new variable to be added to the master problem. The master problem is then re-solved and this process is repeated. In contrast, in our approach we solve a single smaller problem based on one subset of the variables. To make a direct connection with our approach, it is as if we solve the master problem only once; a sort of one-shot column generation.

Column generation has made it possible to find nearly optimal solutions to huge LPs, which would otherwise be considered intractable to solve, and has made a significant impact in, e.g., airline crew scheduling problems [18, 86], transportation routing problems, and scheduling problems. In general, column generation is a heuristic approach; see work by [161, 175]. Theoretical bounds on the solution quality exist only for problem specific applications regarding ILPs; for example the classical cutting stock problem has been studied in this context [81]. For solving ILPs in general, column generation along with *branch-and-bound* rounding techniques have been developed to produce the so called *branch-and-price* algorithm [17]. In the context of column generation, the results in this work can be thought of as theoretical groundwork for a one-shot column generation for packing problems where, in order to ensure feasibility, we require an assumption on the maximum value of the b vector (recall that we scale $a_{ij} \in [0, 1]$ w.l.o.g.).

Another related algorithmic idea that has been applied to LPs is *constraint sampling*, in which a subproblem is formed from a subset of the constraints;

see [147] for recent work. We note that subsampling the constraints differs from subsampling variables (as in column generation), in that in constraint sampling a complete $x \in \mathbb{R}^n$ solution is acquired after solving the subproblem, while in column generation additional work is required to recover the primal solution.

In the general context of convex solvers, there has been considerable work over the past decade. Convex solvers are generally either interior-point methods or first order methods. Interior-point methods are used in most off-the-shelf public software packages like SDPT3 [193] and SeDuMi [188], the commercial software package MOSEK [9], and the more recent embedded conic solver, ECOS [62]. However, the computational cost of the second-order methods used in interior-point methods is often prohibitive for high dimensional problems. In comparison, first order methods tend to scale well for very large problems. For a survey on large scale optimization methods and first-order methods see [73] and [40]. SCS [151] is a widely used first order method. It employs an operator splitting approach, or specifically *alternating directions method of multipliers* [78, 33]. Given the popularity of SCS, we use it in this thesis to illustrate the acceleration provided by our framework in the convex case.

2.3 A Black-box Acceleration Framework

In this section we introduce our acceleration framework. At a high level, the framework works by using an existing solver in a black-box fashion to solve a small problem, defined on a subset of the variables. This approximate solution is then incorporated in a deterministic thresholding scheme to assign the variables in the original problem.

The framework applies to solvers that are either exact or approximate. In the approximate case, the solution satisfies the approximate complementary slackness if the following holds. Let $x_1, \ldots, x_n$ be a feasible solution to the primal and $y_1, \ldots, y_m$ be a feasible solution to the dual. We refer to column j of matrix A as a_j.

- *Primal Approximate Complementary Slackness:* For $\alpha_p \geq 1$ and $j \in [n]$, if $x_j > 0$ then $c_j \leq \sum_{i=1}^{m} a_{ij} y_i \leq \alpha_p \cdot c_j$.
- *Dual Approximate Complementary Slackness:* For $\alpha_d \geq 1$ and $i \in [m]$, if $y_i > 0$ then $b_i/\alpha_d \leq \sum_{j=1}^{n} a_{ij} x_j \leq b_i$.

We call an algorithm $\mathcal{A}$ whose solution is guaranteed to satisfy the above conditions an (α_p, α_d)-*approximation algorithm* for $\mathcal{F}$. This terminology is non-standard, but is instructive when describing our results. It stems from a foundational result which states that an algorithm $\mathcal{A}$ that satisfies the above conditions is an α-approximation algorithm for any LP in $\mathcal{F}$ for $\alpha = \alpha_p \alpha_d$ e.g., [35].

The framework we present can be used to accelerate any $(1, \alpha_d)$-approximation algorithm. While this is a stronger condition than simply requiring that $\mathcal{A}$ is an α-approximation algorithm, many common dual ascent algorithms satisfy this condition, e.g., [2, 15, 16, 70, 82]. For example, the vertex cover and Steiner tree approximation algorithms of [2] and [16] respectively are both $(1, 2)$-approximation algorithms.

2.3.1 Set Up

We define the dual problem to (2.1). Let $\phi \in \mathbb{R}^m$ be the dual variable corresponding to the primal constraints (2.1a), and let $\psi \in \mathbb{R}^n$ correspond to constraints (2.1b). The dual problem is

$$\underset{\phi \in \mathbb{R}^m_+, \psi \in \mathbb{R}^n_+}{\text{minimize}} \quad b^T \phi - \textstyle\sum_{j=1}^n f_j^\star(a_j^T \phi + \psi_j) + \mathbf{1}^T \psi,$$

where $\mathbf{1}$ is the vector of ones, and $f^\star$ is the concave conjugate function defined as

$$f_j^\star(v) = \inf_{x \in \mathbb{R}_+} v x_j - f_j(x_j). \tag{2.3}$$

The Lagrangian for problem (2.1) is

$$L(x, \phi, \psi) := \sum_{t=1}^n f_j(x_j) - \phi^T(Ax - b) - \psi^T(\mathbf{1} - x), \tag{2.4}$$

where $x \geq 0$, from which we derive the following optimality condition:

$$x_j \in \underset{x_j \geq 0}{\text{argmax}} \quad f_j(x_j) - (a_j^T \phi + \psi_j) x_j \quad \forall j \in [n]. \tag{2.5}$$

Thus note that

$$f_j'(x_j) = a_j^T \phi + \psi_j \quad \forall j \in [n]. \tag{2.6}$$

The complimentary slackness conditions $\psi_j(1 - x_j) = 0$ for all $j \in [n]$ imply that if $\psi_j > 0$, then $x_j^* = 1$ where * denotes optimality. However, if $\psi_j = 0$, then $f_j'(x_j^*) = a_j^T \phi^*$. We use these facts to motivate our algorithm.

2.3.2 Acceleration Algorithm

Given a $(1, \alpha_d)$-approximation algorithm $\mathcal{A}$, the acceleration framework is comprised of the following two steps. The approach is summarized in Algorithm 1.

Step 1.

Uniformly at random select a subset of the variables, $S \subset [n]$, $|S| = s = \lceil \epsilon_s n \rceil$. Relabel the sampled variables as $1, \ldots, s$ for clarity, and define the *sample problem* on s variables as follows:

$$\begin{aligned}
&\underset{x \in \mathbb{R}^s}{\text{maximize}} && \textstyle\sum_{j=1}^{s} f_j(x_j) && && \text{(2.7a)}\\
&\text{subject to} && \textstyle\sum_{j=1}^{s} a_{ij} x_j \leq \frac{(1-\epsilon_f)\epsilon_s}{\alpha_d} b_i && i \in [m] && \text{(2.7b)}\\
& && 0 \leq x_j \leq 1 && j \in [s]. && \text{(2.7c)}
\end{aligned}$$

Here, α_d is the parameter of the dual approximate complementary slackness guarantee of $\mathcal{A}$, $\epsilon_f > 0$ is a parameter set to ensure feasibility during the thresholding step, and $\epsilon_s > 0$ is a parameter that determines the fraction of the primal variables that are be sampled. Our analytic results give insight for setting ϵ_f and ϵ_s; see Section 2.5.3.

Next, use solver $\mathcal{A}$ to solve the dual of the sample problem (2.7). Let $\phi^S \in \mathbb{R}^m$ be the dual variable corresponding to the primal constraints (2.7b), and let $\psi^S \in \mathbb{R}^s$ correspond to constraints (2.7c). Then, the sample dual problem is

$$\underset{\phi \in \mathbb{R}^m_+, \psi \in \mathbb{R}^s_+}{\text{minimize}} \quad \frac{(1-\epsilon_f)\epsilon_s}{\alpha_d} b^T \phi - \textstyle\sum_{j=1}^{s} f_j^\star(a_j^T \phi + \psi_j) + \mathbf{1}^T \psi, \qquad (2.8)$$

where $\mathbf{1}$ is the vector of ones, and $f^\star$ is the concave conjugate function as defined in (2.3).

Finally, retrieve the sample dual solution ϕ^S, which is an approximation of the dual solution of the original problem (2.1).

Step 2.

The second step in our acceleration framework uses the dual solution from the sample problem to define a deterministic thresholding procedure, which is used to construct the solution of problem (2.1). This procedure is motivated by the optimality condition (2.5).

Evaluating (2.5) when f' is non-invertible (equivalently f^* is non-differentiable; $f'^{-1} = f^{\star\prime}$), we find, for $j \in [s]$,

$$x_j^S \in \partial f^*(a_j^T \phi^S + \psi_j^S) = \underset{0 \le x_j^S \le 1}{\operatorname{argmax}} \, (a_j^T \phi^S + \psi_j^S) x_j^S - f_j(x_j^S). \tag{2.9}$$

The set of subgradients $\partial f^*(a_j^T \phi^S + \psi_j^S)$ is not necessarily a singleton set. We assign our approximation of the primal solution, which we denote as $x_j(\phi^S)$, to be the smallest element in $\partial f^*(a_j^T \phi^S + \psi_j^S)$. Such an element exists because the set is a closed convex subset of $[0,1]$. We define the *allocation rule* $x_j(\phi^S)$, in which we set the primal variables for all $j \in [n]$ as a function of the dual solution ϕ^S:

$$x_j(\phi^S) := \begin{cases} 1 & \text{if } \; a_j^T \phi^S < f_j'(1) \\ \arg\min \; \partial f_j^*(a_j^T \phi^S + \psi^S) & \text{otherwise.} \end{cases} \tag{2.10}$$

If f is strictly convex, (2.10) reduces to the following:

$$x_j(\phi^S) := \begin{cases} 1 & \text{if } \; a_j^T \phi^S < f_j'(1) \\ f_j'^{-1}(a_j^T \phi^S) & \text{otherwise.} \end{cases} \tag{2.11}$$

For an example of (2.11), consider the following: if $f_j(x_j) = c_j \log(x_j)$, then $x_j(\phi^S) = 1$ if $a_j^T \phi^S < c_j$, and $x_j(\phi^S) = \frac{c_j}{a_j^T \phi^S}$ otherwise. If f is linear, (2.10) reduces to the following binary thresholding procedure:

$$x_j(\phi^S) = \begin{cases} 1 & \text{if } \; a_j^T \phi^S < c_j \\ 0 & \text{otherwise.} \end{cases} \tag{2.12}$$

The allocation rule (2.10) reduces to (2.12) in the linear case because the derivative of f_j is c_j for all x. In (2.10) we take the smallest value in the set of subgradients $\partial f^*(a_j^T \phi^S + \psi_j^S)$, which in this case is $x = 0$.

Setting ϵ_s and ϵ_f

The parameters ϵ_s and ϵ_f must be chosen when using the framework. In general, parameter ϵ_s is chosen based on the user's requirement for accuracy versus speed; it directly controls the size of the sample problem. The parameter ϵ_f is related to the feasibility and nearness to optimally of the solution. Concretely, we suggest choosing ϵ_s and ϵ_f using the following approach. First, choose ϵ_s

Algorithm 1: Acceleration Framework

Input: Convex Program $\mathcal{C}$, solver $\mathcal{A}$, $\epsilon_s > 0$, $\epsilon_f > 0$

Output: $\hat{x} \in \mathbb{R}^n$

```
(1) Select $s = \lceil \epsilon_s n \rceil$ primal variables uniformly at random. Solve the
    sample dual problem (2.8) using $\mathcal{A}$ to find an (approximate) dual
    solution $y^S = [\phi^S, \psi^S] \in [\mathbb{R}^m, \mathbb{R}^s]$.
(2) Set $\hat{x}_j = x_j(\phi^S)$ as defined in (2.10), for all $j \in [n]$.
```

based on the user's requirement for speed vs. accuracy. This tradeoff is illustrated in the experiments section in Figures 21 and 24. Given a fixed ϵ_s, a concrete approach for choosing ϵ_f is given in Algorithm 2. Here, we set $\epsilon_f = 0$ and increase ϵ_f iteratively, solving the sample problem with different values of the parameter. Solving multiple problems in order to identify the best ϵ_f is generally not time prohibitive, because the sample problems are very small compared to the original. One can also simply set ϵ_f in accordance with the theoretical bounds. Our analytic results in the next section provide guarantees on the largest ϵ_f that guarantees feasibility. However, we find in practice that we can often use a smaller ϵ_f and get a closer to optimal solution without violating feasibility. Thus, it is useful to search for the minimal ϵ_f that provides feasibility for a given problem. A practical discussion of setting both ϵ_s and ϵ_f is found in Section 2.5.3.

Algorithm 2: An approach for setting ϵ_f

Input: Convex Program $\mathcal{L}$, solver $\mathcal{A}$, $\epsilon_s > 0$, $\epsilon_f > 0$

Output: $\hat{x} \in \mathbb{R}^n$

```
Set $\epsilon_f = 0$.
while $\epsilon_f < 1$ do
    $\hat{x}$= Algorithm 1($\mathcal{L}$,$\mathcal{A}$,$\epsilon_s$,$\epsilon_f$).
    if $\hat{x}$ is a feasible solution to $\mathcal{L}$ then
        Return $\hat{x}$.
    else
        Increase $\epsilon_f$.
```

Discussion

Before moving to the analysis, we provide some context about why the allocation rule (2.10) works. Recall that (2.6) states that $f_j'(x_j) = a_j^T \phi + \psi_j$

for all $j \in [n]$. Hence, if $\psi_j > 0$, the complimentary slackness conditions $\psi_j(1 - x_j) = 0$ imply that $x_j^* = 1$. However, if $\psi_j = 0$, then $x_j^* = f_j'^{-1}(a_j^T \phi^*)$. Thus, in the allocation rule we set $x_j(\phi)$ based on these optimality conditions. Notice that we do not have access to the optimal dual solution ϕ^* to problem (2.1), but rather to an approximation ϕ^S, the solutions to the sample dual problem (2.8). Thus we cannot satisfy the complementary slackness conditions exactly. The key argument we need to make is that, despite having only an approximate ϕ^S, the solution $x_j(\phi^S)$ is nearly optimal. To account for the fact that an approximate dual solution is produced in the first step, we preemptively scale the b vector by the factor $\frac{(1-\epsilon_f)\epsilon_s}{\alpha_d}$ in constraints (2.1a). Given this rescaling, the solution $x_j(\phi^S)$ is feasible and nearly optimal with high probability.

2.4 Results: Feasibility and Optimality Guarantees

In this section we present our main technical results, which provide worst-case guarantees on the feasibility and optimality of the solution provided by our acceleration framework.

Let $\mathcal{C}$ be a packing problem with n variables and m constraints, as in (2.1), and define $B := \min_{i\in[m]}\{b_i\}$. The following theorem bounds the quality of the solution provided by the acceleration framework.

Theorem 1. *Let $\mathcal{C}$ be a packing problem of form* (2.1) *or* (2.2)*, with n variables and m constraints, and define $B := \min_{i\in[m]}\{b_i\}$. Let $\mathcal{A}$ be a $(1, \alpha_d)$-approximation algorithm for $\mathcal{C}$, with runtime $f(n, m)$. For any $\epsilon_s, \epsilon_f > 0$, in the concave case* (2.1)*, if,*

$$B \geq \frac{10(m\log(n + \frac{4\epsilon_f\epsilon_s n^2}{m\log n}) + \log(\frac{m}{\epsilon_s}))}{\epsilon_f^2\epsilon_s}, \tag{2.13}$$

or in the linear case (2.2)*, if,*

$$B \geq \frac{10(m\log(n) + \log(\frac{m}{\epsilon_s}))}{\epsilon_f^2\epsilon_s}, \tag{2.14}$$

then Algorithm 1 runs in time $f(\epsilon_s n, m) + O(n)$, and obtains a feasible $\left(\frac{1-\epsilon_f}{\alpha_d^2}\right)$-approximation to the optimal solution for $\mathcal{C}$ with probability at least $1 - 2\epsilon_s$.

The key trade-off in the acceleration framework is between the size of the sample problem, determined by ϵ_s, and the resulting quality of the solution, determined by the feasibility parameter, ϵ_f. The accelerator provides a large speedup if ϵ_s can be made small without causing ϵ_f to be too large. Theorem 1 quantifies the trade-off between ϵ_f and ϵ_s, along with the relation to B. The bound on B in Theorem 1 defines the class of problems for which the accelerator is guaranteed to perform well: problems for which $m \ll n$ and B is not too small. Nevertheless, our experimental results successfully apply the framework well outside of these parameters; the theoretical analysis provides a very conservative view on the applicability of the framework.

We also compare the results for the concave versus linear case in Theorem 1. First, note that the concave result does not depend on the form of the concave function f. Second, note that the bounds on B in the concave (2.13) and linear (2.14) cases are of the same order; $\Omega\left(\frac{m}{\epsilon_f^2 \epsilon_s}\log(n)\right)$. In a recent paper on linear programs, [117] achieve the same order bound on B. Thus, despite applying to general convex programs in this work, we achieve the same order bound. We also note that the difference in (2.13) and (2.14) occurs in the second term in the argument of the first logarithmic term, which is present due to the analysis of the convex case. The reason for this difference is made clear in the proof of Lemma 12.

As discussed in the introduction, our work is related to the literature on online algorithms for packing linear programs with optimal competitive ratio under random permutation model. In this context B is often referred to as the bid-to-budget ratio. In the case of [4], the one-time learning algorithm achieves a competitive ratio of $1 - \epsilon$ if the bid-to-budget ratio is $\Omega\left(\frac{m}{\epsilon^3}\log(\frac{n}{\epsilon})\right)$, which is similar to the assumption on B in Theorem 1. However, [135] present a different analysis for the algorithm by [4], in which they require a bid-to-budget ratio of $\Omega\left(\frac{m^2}{\epsilon^3}\log(\frac{m}{\epsilon})\right)$. Removing the dependency on n, which is the time horizon of the online problem, is of theoretical importance in the online setting. However, in our setting, even though $m \ll n$, typically m is comfortably larger than $\log(n)$; hence a result similar to that of [4] is more desirable.

In addition to exact and approximate LP solvers, our framework can be used solve integer linear programs (ILP). These are linear programs with binary decision variables:

$$\begin{aligned} \text{maximize} \quad & \textstyle\sum_{j=1}^{n} c_j x_j & & \qquad (2.15a) \\ \text{subject to} \quad & \textstyle\sum_{j=1}^{n} a_{ij} x_j \leq b_i & i \in [m] & \qquad (2.15b) \\ & x_j \in \{0,1\} & j \in [n]. & \qquad (2.15c) \end{aligned}$$

The allocation rule (2.12) produces binary solutions, and so naturally the solutions satisfy ILP integrality constraints.

Our framework can be parallelized; Step 2 of Algorithm 1 can be done in parallel. Additionally, if the solver $\mathcal{A}$ that is being accelerated is parallelizable, further parallelization can be achieved. We formally state these points below.

Corollary 2. *Let $\mathcal{A}$ be an $(1, \alpha_d)$-approximation algorithm for problems of form* (2.1)*, with runtime $f(n, m)$.*

- *Consider an ILP. Algorithm 1 obtains a feasible $\frac{(1-\epsilon_f)}{\alpha_d^2}$-approximation to the optimal solution for the ILP with probability at least $1 - \frac{1}{n}$ with runtime $f(\epsilon_s n, m) + O(n)$.*
- *If $\mathcal{A}$ is a parallel algorithm, then executing Algorithm 1 on p processors in parallel obtains a feasible $\frac{(1-\epsilon_f)}{\alpha_d^2}$-approximation to the optimal solution with probability at least $1 - \frac{1}{n}$ and runtime $f_p(\epsilon_s n, m) + O(n/p)$, where $f_p(\epsilon_s n, m)$ denotes $\mathcal{A}$'s runtime for the sample program on p processors.*

2.5 Experiments

We illustrate the speedup provided by our acceleration framework by using it to accelerate state-of-the-art commercial solvers. We first run our acceleration framework on synthetic randomly generated problems, and then provide a real data example. Additionally, we demonstrate the benefits of cloning.

We accelerate Gurobi and SCS, in the case of linear and convex programs respectively. Due to limited space, we do not present results for more specialized solvers; however, the improvements shown here provide a conservative estimate of the improvements possible with more specialized solvers. Similarly, the speedup provided by an exact solver (such as Gurobi) provides a conservative estimate of the improvements when applied to approximate solvers or when applied to solve ILPs.

Note that our experiments consider situations where the assumptions in Theorem 1 on B, m, and n do not hold. Thus, they highlight that the assumptions of the theorem are conservative and the accelerator can perform well outside of the settings prescribed by the analysis. This is also true with respect to the assumptions on the algorithm being accelerated. While our proof requires the algorithm to be a $(1, \alpha_d)$-approximation, the accelerator works well for other types of algorithms too. For example, we have applied our framework to the gradient algorithm such as [185] in the linear case, with results that parallel those presented for Gurobi below. All experiments are run on a server with Intel E5-2623V3@3.0GHz 8 cores and 64GB RAM.

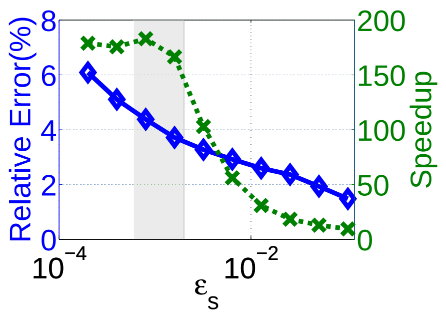
2.5.1 Linear Case: Accelerating Gurobi

In this section we describe experiments done on synthetic randomly generated linear programs. We describe the speedup provided for Gurobi, a state-of-the-art commercial LP solver.

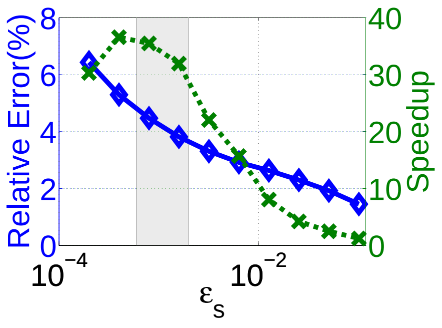
Experimental Setup. To illustrate the performance of our accelerator, we run Algorithm 1 on randomly generated LPs. Unless otherwise specified, the experiments use a matrix $A \in \mathbb{R}^{m \times n}$ of size $m = 10^2, n = 10^6$. Each element of A, denoted as a_{ij}, is first generated from $[0, 1]$ uniformly at random and then set to zero with probability $1 - p$. Hence, p controls the sparsity of matrix A, and we vary p in the experiments. The vector $c \in \mathbb{R}^n_{\geq 0}$ is drawn i.i.d. from $[1, 100]$ uniformly. Each element of the vector $b \in \mathbb{R}^m_{\geq 0}$ is fixed as $0.1n$. (Note that the results are qualitatively the same for other choices of b.) By default, the parameters of the accelerator are set as $\epsilon_s = 0.01$ and $\epsilon_f = 0$, though these are varied in some experiments. Each point in the presented figures is the average of over 100 executions under different realizations of A and c.

In order to measure the quality of the solution, we define the *relative error* as $(1 - \hat{p}/p^*)$, where p^* is the optimal objective value and $\hat{p}$ is the objective value produced by our algorithm. To measure the acceleration, we define the *speedup* of the accelerated algorithm as the ratio of the runtime of the original solver to the runtime of our algorithm.

We implement the accelerator in Matlab and use it to accelerate Gurobi. We intentionally perform the experiments with a small degree of parallelism in order to obtain a conservative estimate of the acceleration provided by our framework. As the degree of parallelism increases, the speedup of the accel-



(a) $p = 0.8$



(b) $p = 0.4$

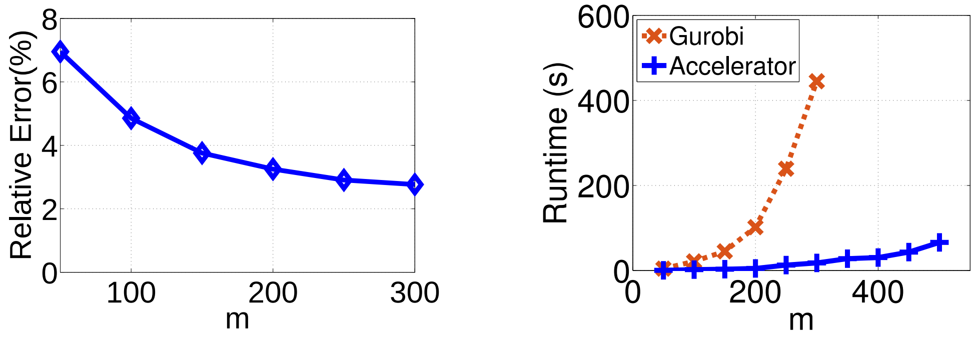
Figure 21: Illustration of the relative error and speedup across sample sizes, ϵ_s. The shaded area depicts a reasonable setting range for ϵ_s; the speedup is significant, while the relative error is low. Two levels of sparsity, p, are shown.

erator increases and the quality of the solution remains unchanged (unless cloning is used, in which case it improves).

Experimental Results. Our experimental results highlight that our acceleration framework provides speedups of two orders of magnitude (over 150×), while maintaining high-quality solutions (relative errors of $< 4\%$).

The trade-off between relative error and speed. The fundamental trade-off in the design of the accelerator is between the sample size, ϵ_s, and the quality of the solution. The speedup of the framework comes from choosing ϵ_s small, but if it is chosen too small then the quality of the solution suffers. For the algorithm to provide improvements in practice, it is important for there to be a sweet spot where ϵ_s is small and the quality of the solution is still good, as indicated in the shaded region of Figure 21.

Scalability. In addition to speeding up LP solvers, our acceleration framework provides significantly improved scalability. Because the LP solver only needs to be run on a (small) sample LP, rather than the full LP, the accelerator provides order of magnitude increase in the size of problems that can be solved. This is illustrated in Figure 23. The figure shows the runtime and relative error of the accelerator. In these experiments we have fixed $p = 0.8$ and $n/m = 10^3$ as we scale m. We have set $\epsilon_s = 0.01$ throughout. As (a) shows, one can choose ϵ_s more aggressively in large problems since leaving ϵ_s fixed leads to improved accuracy for large scale problems. Doing this would lead to larger speedups; thus by keeping ϵ_s fixed we provide a conservative estimate of the improved scalability provided by the accelerator. The results in (b) illustrate



(a) Relative Error (b) Runtime

Figure 22: Illustration of the relative error and runtime as the problem size, m, grows. Throughout, $n/m = 10^3$. In (b), Gurobi ran into memory errors for problems with $m > 300$. However, our accelerator algorithm was able to solve the problems in a faction of the time, with low relative error.

the improvements in scalability provided by the accelerator. Gurobi's run time grows quickly until finally, it runs into memory errors and cannot arrive at a solution. In contrast, the runtime of the accelerator grows slowly and can (approximately) solve problems of much larger size. To emphasize the improvement in scalability, we run an experiment on a laptop with Intel Core i5 CPU and 8 GB RAM. For a problem with size $m = 10^2, n = 10^7$, Gurobi fails due to memory limits. In contrast, the accelerator produces a solution in 10 minutes with relative error less than 4%.

2.5.2 Convex Case: Accelerating SCS

In this section we empirically demonstrate the speed up provided by our acceleration method for convex. problems of form (2.1). We accelerate SCS, a state-of-the-art convex program solver. As in the LP case, the experiments show that our method provides order-of-magnitude speedups, while producing near-optimal solutions. We consider several monotonically increasing objective functions, as described below.

Experimental Setup. We implement the accelerator in python and use CVXPY [60] to call SCS. We use randomly generated problems for our experiments and each point in the figures is averaged over 50 executions under different realizations of A, b and f_j for all $j \in [n]$.

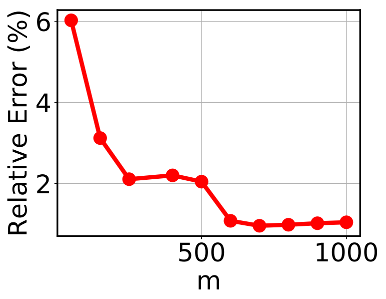
To generate random instances, we generate matrices $A \in \mathbb{R}^{m \times n}$ as follows. Each element of A is drawn i.i.d. from the uniform distribution $U[0, 1]$, and then set to zero with probability $1 - p$. We vary p in the experiments, which controls the sparsity of matrix. We set $b = Ay + 0.1z$, where y and z are both random vectors drawn i.i.d. from $U[0, 1]$. We consider concave objective functions $f_j(x_j) = c_j \log(x_j)$ and $f_j(x_j) = c_j {x_j}^{1/2}$, where c_j is drawn i.i.d. from $U[0, 1]$. We note that the derivative does not exist at $x_j = 0$ for the function $\log(x_j)$. Instead, to satisfy our assumptions, we can solve an equivalent problem with objective function $\log(x_j + \delta) - \log(\delta)$.

Unless otherwise specified, the problem dimensions are $m = 10^2, n = 10^6$, the parameters are set as $\epsilon_s = 0.01$ and $\epsilon_f = 0$ and ϵ_f is increased as described in Algorithm 2, found in Section 2.3, until feasibility is reached. The parameters are varied in experiments.

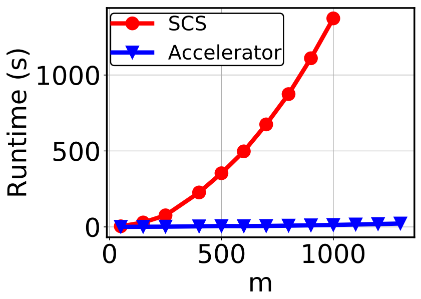
Note that these experiments consider settings in which the assumptions of Theorem 1 on B do not hold. When we set $b = Ay + 0.1z$ as described above, these values are smaller than the bound on B. However our algorithm performs well even in this setting, demonstrating that the assumptions of the theorem are conservative.

Experimental Results. We see a remarkable speedup in our experiments, while maintaining high-quality solutions. Figure 23 demonstrates the relative error and speedup as the size of the problems are varied. The ratio $n/m = 10^3$ is fixed as we vary both n and m. We set $p = 0.8$ and $\epsilon_s = 0.01$. Here we see a speedup of multiple orders of magnitude compared to SCS. Additionally, the line corresponding to SCS stops at about $n = 1M$ variables. At this point, SCS runs into memory errors and is unable to proceed. However, our algorithm is able to solve the same problem in a factor of 100 less time, while achieving a relative error of 1%. Our accelerator is additionally able to proceed with larger problems, ones that SCS can not handle. Thus, our accelerator provides remarkable scalability: we can quickly solve problems with orders of magnitude more variables than SCS can handle.

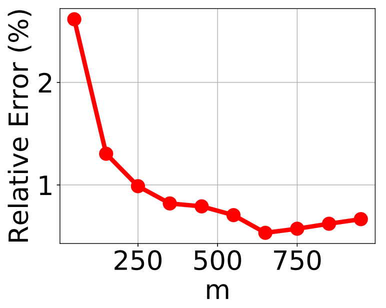
Figure 24 demonstrates the relative error and speedup as the sample size ϵ_s is varied. We see that, as ϵ_s decreases, the speedup is larger, but problems become harder and thus error increases. For larger ϵ_s, the speedup is less pronounced but the accuracy is very high.



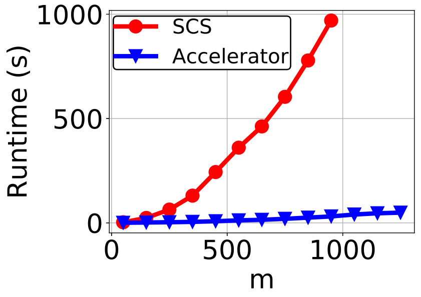
(a) Relative Error



(b) Runtime

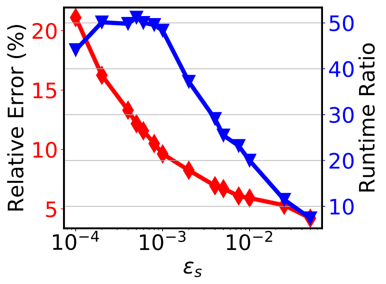


(c) Relative Error

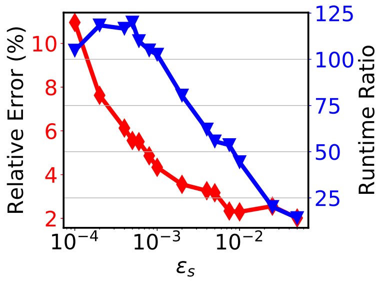


(d) Runtime

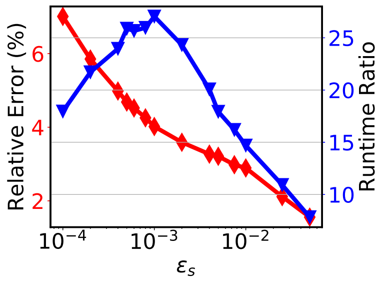
Figure 23: The relative error and runtime as the problem size, m, grows, where we fix the ratio $n/m = 10^3$. In (a), (b), $f_j(x_j) = c_j \log(x_j)$, and in (c), (d) $f_j(x_j) = c_j x_j^{1/2}$.



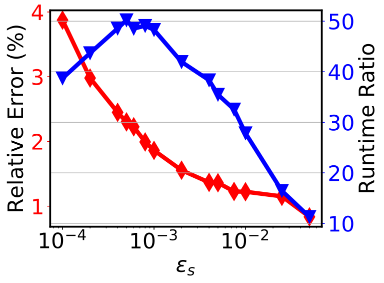
(a) Relative Error



(b) Runtime



(c) Relative Error



(d) Runtime

Figure 24: The relative error and runtime ratios as the sample size, ϵ_s, varies, where we fix $n = 5 \times 10^5$, $m = 100$. In (a), (b), $f_j(x_j) = c_j \log(x_j)$, and in (c), (d) $f_j(x_j) = c_j x_j^{1/2}$.

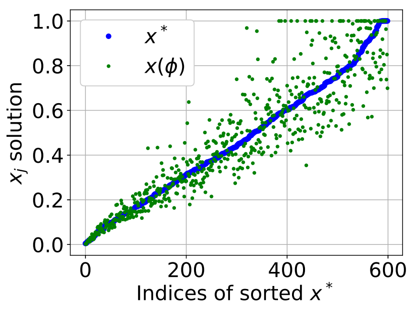
2.5.3 Setting Parameters in Practice

While Theorem 1 gives guidance on how to set ϵ_f and ϵ_s to ensure the worst-case guarantees, in practice it is possible to be more aggressive.

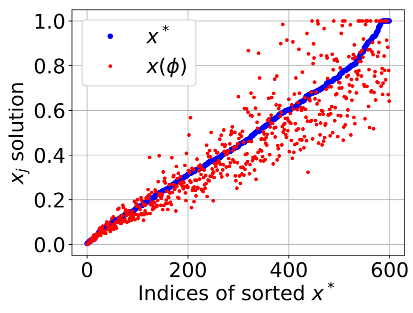
Figure 25 demonstrates the nature of the solution as ϵ_s and ϵ_f vary. We plot the optimal solution sorted by magnitude, as well as our approximate solution sorted in the same order as the optimal, i.e., x_j^* and $x(\phi)_j$ appear on the same vertical line for the same j. First, we observe the effects of varying ϵ_f. In (a) and (c) we experiment with setting $\epsilon_f = 0$, and find that the solution produced is infeasible. However in (b) and (d) we increase ϵ_f and find a feasible solution. Visually, when comparing (a) and (b), we can see our solution shifts downwards in (b), which makes sense; a solution with smaller values is more likely to be feasible in a packing maximization problem. Secondly, we observe the effects of varying ϵ_s. When comparing (b) and (d), we demonstrate that increasing ϵ_s causes the approximate solution to tighten around the optimal; a larger sample size allows for a closer to optimal solution.

2.5.4 The Benefits of Cloning

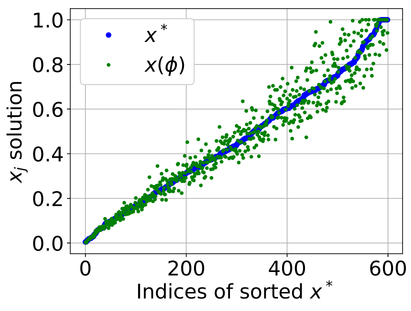
Speculative execution is an important tool that parallel analytics frameworks use to combat the impact of stragglers. Our acceleration framework can implement speculative execution seamlessly by running multiple clones (samples) in parallel and choosing the ones that finish the quickest. We illustrate the benefits associated with cloning in Figure 26. This figure shows the percentage gain in relative error and speedup associated with using different numbers of clones. In these experiments, we consider LPs generated as described in Section 2.5.1. We fix $\epsilon = 0.002$ and $p = 0.8$. We vary the number of clones run and the accelerator outputs a solution after the fastest four clones have finished. Note that the first four clones do not impact the speedup as long as they can be run in parallel. However, for larger numbers of clones our experiments provide a conservative estimate of the value of cloning since our server only has 8 cores. The improvements would be larger than shown in Figure 26 in a system with more parallelism. Despite this conservative comparison, the improvements illustrated in Figure 26 are dramatic. Cloning reduces the relative error of the solution by 12% and triples the speedup. Note that these improvements are significant even though the solver we are accelerating is not a parallel solver.



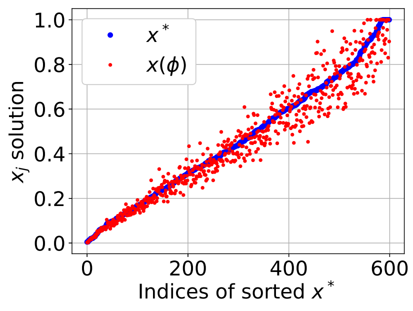
(a) $\epsilon_f = 0$, $\epsilon_s = 0.5$



(b) $\epsilon_f = 0.1$, $\epsilon_s = 0.5$

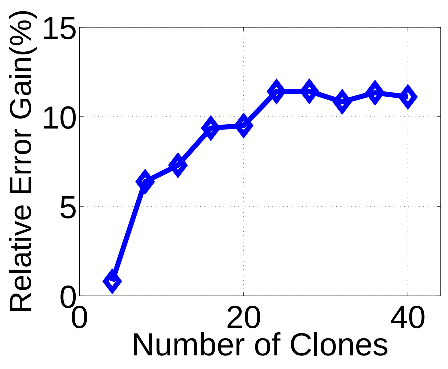


(c) $\epsilon_f = 0$, $\epsilon_s = 0.8$

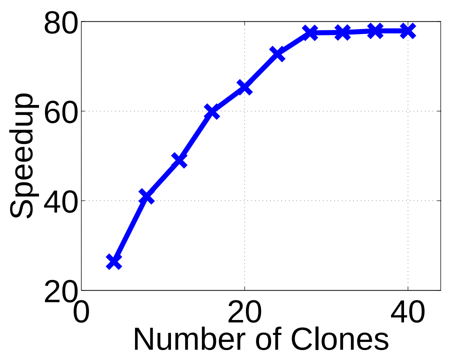


(d) $\epsilon_f = 0.05$, $\epsilon_s = 0.8$

Figure 25: Illustration of how the optimal solution x^* and our approximate $x(\hat{\phi})$ solution differ. In particular, increasing ϵ_f affects the feasibility of our solution: comparing (a) and (b), as ϵ_f is increased, the approximate solution $x(\hat{\phi})$ shifts downwards; in (a) the approximate solution is infeasible, but setting $\epsilon_f = 0.1$ produces a feasible $x(\hat{\phi})$. Here $m = 100$ and $n = 600$ and $f_j(x_j) = c_j \log(x_j)$.



(a) Relative Error



(b) Speedup

Figure 26: Illustration of the impact of cloning on solution quality as the number of clones grows.

2.5.5 Case Study: California Road Network Dataset

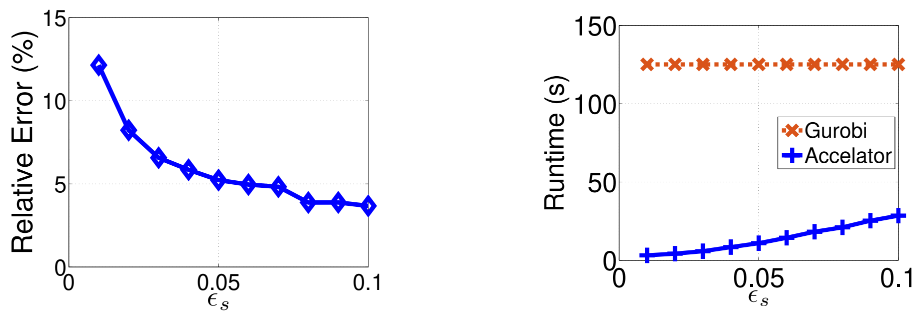
To illustrate the performance in a specific practical setting, we consider an example focused on optimal resource allocation in a network. We consider an LP that represents a multi-constraint knapsack problem associated with placing resources at intersections in a city transportation network. For example, we can place medical testing facilities, advertisements, or emergency supplies at intersections in order to maximize social welfare, but such that there never is a particularly high concentration of resources in any area.

Specifically, we consider a subset of the California road network dataset [111], consisting of $100,000$ connected traffic intersections. We consider only a subset of a total of $1,965,206$ intersections because Gurobi is unable to handle such a large dataset when run on a laptop with Intel Core i5 CPU and 8 GB RAM. We choose 1000 of the $100,000$ intersections uniformly at random and defined for each of them a local vicinity of $20,000$ neighboring intersections, allowing overlap between the vicinities. The goal is to place resources strategically at intersections, such that the allocation is not too dense within each local vicinity. Each intersection is associated with a binary variable which represents a yes or no decision to place resources there. Resources are constrained such that the sum of the number of resource units placed in each local vicinity does not exceed $10,000$.

Thus, the dataset is as follows. Each element A_{ij} in the data matrix is a binary value representing whether or not the i-th intersection is part of the j-th local vicinity. There are 1000 local vicinities and $100,000$ intersections, hence A is a $(1000 \times 100,000)$ matrix. Within each local vicinity, there are no more than $b_j = 10,000$ resource units.

The placement of resources at particular locations has an associated utility, which is a quantifier of how beneficial it is to place resources at various locations. For example, the benefit of placing medical test supplies, advertisements, or emergency supplies at certain locations may be proportional to the population of the surrounding area. In this problem, we randomly draw the utilities from Unif$[1, 10]$. The objective value is the sum of the utilities at whose associated nodes resources are placed.

Figure 27 demonstrates the relative error and runtime of the accelerator compared to Gurobi, as we vary the sample size ϵ_s. There is a speed up by a factor of more than 30 when the approximation ratio is 0.9, or a speed up by a factor



(a) Relative Error (b) Runtime

Figure 27: Illustration of the relative error and runtime across sample sizes, ϵ_s, for the real data experiment on the California road network dataset.

of about 9 when the approximation ratio is 0.95.

2.6 Proofs

In this section we present a proof of Theorem 1. The proof approach has two main steps: (1) show that the solution provided by Algorithm 1 is feasible with high probability (Lemma 12); and (2) show that the value of the solution is sufficiently close to optimal with high probability (Lemma 16). First, we present preliminary results.

2.6.1 Preliminary Results

We make use of the following concentration bound, for example as found in [196].

Theorem 3 (Hoeffding-Bernstein Inequality). *Let $u_1, u_2 \ldots, u_s$ be random samples without replacement from the real numbers $r_1, \ldots, r_n$, where $r_j \in [0,1]$. For $t > 0$, $\Pr\left[|\sum_{j=1}^{s} u_j - \frac{s}{n}\sum_{j=1}^{n} r_j| \geq t\right] \leq 2\exp\left(\frac{-t^2}{2s\sigma_n^2 + t}\right)$, where $\sigma_n^2 = \frac{1}{n}\sum_{j=1}^{n}(r_j - \sum_{j=1}^{n} r_j/n)^2$.*

Throughout this section, we use the following definition.

Definition 4. Denote the solution produced by Algorithm 1 as $x(\phi^S) \in \mathbb{R}^n$, the solution to the sample problem (2.7) as $x^S \in \mathbb{R}^s$, and the dual sample solutions as $\phi^S \in \mathbb{R}^m$ and $\psi^S \in \mathbb{R}^s$. Let the algorithm $\mathcal{A}$ being accelerated be a $(1, \alpha_d)$–approximation algorithm. Define a sample $S \subset [n]$, $|S| = \epsilon_s n$. Let $\mathcal{S}$ be the set of all samples, and $N = [n]$. Define events, associated with each constraint $i \in [m]$,

$$\begin{aligned} D_i &= \left\{S \in \mathcal{S} : \textstyle\sum_{j \in S} a_{ij} x_j^S \le \frac{(1-\epsilon_f)\epsilon_s}{\alpha_d} b_i \right\} \\ E_i &= \left\{S \in \mathcal{S} : \textstyle\sum_{j \in S} a_{ij} x_j(\phi^S) \le (1-\epsilon_f)\epsilon_s b_i \right\} \\ F_i &= \left\{S \in \mathcal{S} : \textstyle\sum_{j \in N} a_{ij} x_j(\phi^S) \le b_i \right\}. \end{aligned}$$

For brevity, we drop $\mathcal{S}$ from our notation hereafter. The following claim describes the relation between the sample solution x^S to problem (2.7) and the solution $x(\phi^S)$ produced by Algorithm 1.

Claim 5. *Let x^S be the sample solution to* (2.7) *and $x(\phi^S)$ be the approximate solution to* (2.1)*. If Algorithm $\mathcal{A}$ is a $(1, \alpha_d)$-approximate solver, then $\alpha_d x_j^S \ge x_j(\phi^S)$ for all $j \in [s]$.*

Proof. Recall that in the allocation rule (2.10) we set $x_j(\phi^S)$ based on the optimality conditions of the sample problem. Specifically, we consider (2.6), reproduced below,

$$f_j'(x_j^S) = a_j^T \phi^S + \psi_j^S \ \forall j \in [s] \tag{2.16}$$

and the value of ψ_j^S, to motivate the two branches of the general allocation rule (2.10), and its special cases, (2.11) and (2.12). Consider the following cases.

- $\psi_j^S > 0$: Evaluating (2.16), we see that $f_j'(x_j^S) > a_j^T \phi^S$. This is the condition in the first branch of each of the allocation rules, in which we set $x_j(\phi^S) = 1$. The approximate complimentary slackness conditions imply that $1 \le x_j^S \alpha_d$. Recalling that $\alpha_d \ge 1$, we see that $\alpha_d x_j^S \ge x(\phi^S)$. Note also that in the case of an exact solver ($\alpha_d = 1$), the exact complimentary slackness conditions $\psi_j^S(1 - x_j^S) = 0$ imply that $x_j^S = 1$, and so $x_j^S = x_j(\phi^S) = 1$.
- $\psi_j^S = 0$: Evaluating (2.16), we see that $f_j'(x_j^S) = a_j^T \phi^S$, which corresponds to the second branch of the allocation rules (2.10), (2.11) and (2.12), which differ. Consider the following cases.
  - Case (1) Concave f_j' non-invertible: As described in the general allocation rule (2.10), we set $x_j(\phi^S)$ equal to the smallest value that satisfies $f_j'(x_j^S) = a_j^T \phi^S$. Thus, $x_j^S \ge x_j(\phi^S)$.

- Case (2) Strictly Concave: The sample solution and the result of the strictly concave allocation rule (2.11) are equal: $x_j^S = x_j(\phi^S) = f_j'^{-1}(a_j^T \phi^S)$.
- Case (3) Linear: The linear allocation rule (2.12) sets $x_j(\phi^S) = 0$, and so $x_j^S \geq x_j(\phi^S)$.

□

The following lemma relates the sample solution x^S to the solution $x(\phi^S)$ produced by Algorithm 1.

Lemma 6. *Define events D_i and E_i as in Definition 4. For all $i \in [m]$, $D_i \subset E_i$.*

Proof. Due to Claim 5, $\alpha_d x_j^S \geq x_j(\phi^S)$ for all $j \in S$. Thus,

$$\sum_{j\in S} a_{ij} x_j(\phi^S) \leq \sum_{j\in S} \alpha_d a_{ij} x_j^S \leq \alpha_d \frac{(1-\epsilon_f)\epsilon_s}{\alpha_d} b_i = (1-\epsilon_f)\epsilon_s b_i,$$

where the second inequality is due to the definition of D_i. □

Finally, the following lemma will be used in the feasibility argument.

Lemma 7. *For all constraints $i \in [m]$,*

$\Pr[D_i \cap \bar{F}_i] \leq \Pr[E_i \cap \bar{F}_i] \leq \Pr\left[|\sum_{j\in S} a_{ij} x_j(\phi^S) - \epsilon_s \sum_{j\in N} a_{ij} x_j(\phi^S)| > \epsilon_f \epsilon_s b_i\right]$.

Proof. For each constraint $i \in [m]$, we evaluate $\Pr[D_i \cap \bar{F}_i]$:

$$\begin{aligned}
\Pr[D_i \cap \bar{F}_i] &= \Pr\left[\sum_{j\in S} a_{ij} x_j^S \leq \frac{(1-\epsilon_f)\epsilon_s}{\alpha_d} b_i \ \cap \ \sum_{j\in N} a_{ij} x_j(\phi^S) > b_i\right] \\
&\leq \Pr\left[\sum_{j\in S} a_{ij} x_j(\phi^S) \leq (1-\epsilon_f)\epsilon_s b_i \ \cap \ \sum_{j\in N} a_{ij} x_j(\phi^S) > b_i\right] \qquad (2.17) \\
&\leq \Pr\left[\sum_{j\in S} a_{ij} x_j(\phi^S) \leq (1-\epsilon_f)\epsilon_s b_i \ \middle| \ \sum_{j\in N} a_{ij} x_j(\phi^S) > b_i\right] \\
&\leq \Pr\left[|\sum_{j\in S} a_{ij} x_j(\phi^S) - \epsilon_s \sum_{j\in N} a_{ij} x_j(\phi^S)| > \epsilon_f \epsilon_s b_i\right],
\end{aligned}$$

where (2.17) is due to Lemma 6. □

2.6.2 Quantizing the Solution Space

We bound the number of possible solutions $x_j(\phi^S)$. Naively it may seem that there are an infinite number of possible solutions, as $x_j \in [0,1]$. However, we can bound the number of possible solutions by considering classes of dual variables that result in approximately equivalent primal realizations $x_j(\phi^S)$. We discretize the primal solution space, $[0,1]$, with a grid size of q, and define the following:

Definition 8. Consider a discretization of the primal solution space $[0,1]$, with grid size $q \leq 1$. Denote the value of f_j' evaluated at discrete values as $T_j = \{f_j'(0^+), f_j'(q), \ldots, f_j'(q\lfloor 1/q \rfloor\} \cup \{f_j'(1^-)\}$.

Definition 9. Consider two dual solutions $\phi^{S_1}, \phi^{S_2} \in \mathbb{R}_+^m$ to problem (2.7). Discretize the primal solution space as in Definition 8. Let $G_j = \{(a_j, \xi) | \xi \in T_j\}$ and $G = \cup_j G_j$. We say ϕ^{S_1} *and* ϕ^{S_2} are in the same equivalence class and write $\phi^{S_1} \sim \phi^{S_2}$ when

$$\forall j, \forall (a_j, \xi) \in G_j : a_j^T \phi^{S_1} \geq \xi \iff a_j^T \phi^{S_2} \geq \xi. \tag{2.18}$$

We show that two dual variables in the same equivalence class, when applied to the allocation rule (2.10), result in two primal solutions which are element-wise within a distance of q from each other.

Lemma 10. *Consider two dual variables in the same equivalence class. The corresponding primal solutions satisfy* $|x_j(\phi^{S_1}) - x_j(\phi^{S_2})| \leq q$ *for all* $j \in [n]$.

Proof. Suppose that the interval $[0,1]$ is discretized with a grid size of q, as in Definition 8. Consider two dual variables in the same equivalence class as specified in Definition 9. Equation (2.18), along with the allocation rule (2.10), imply that if $\phi^{S_1} \sim \phi^{S_2}$, then the following cases occur:

- when $a_j^T \phi^{S_1} < f_j'(1^-)$ and $a_j^T \phi^{S_2} < f_j'(1^-)$, then $x_j(\phi^{S_1}) = x_j(\phi^{S_2}) = 1$
- when $a_j^T \phi^{S_1} \geq f_j'(0^+)$ and $a_j^T \phi^{S_2} \geq f_j'(0^+)$, then $x_j(\phi^{S_1}) = x_j(\phi^{S_2}) = 0$
- when $f_j'(1^-) \leq a_j^T \phi^{S_1} < f_j'(0^+)$ and $f_j'(1^-) \leq a_j^T \phi^{S_2} < f_j'(0^+)$, then $|x_j(\phi^{S_1}) - x_j(\phi^{S_2})| \leq q$.

Thus in general, $|x_j(\phi^{S_1}) - x_j(\phi^{S_2})| \leq q$ for all j.

□

We employ a classical result of combinatorial geometry [153] to bound the number of possible primal solutions.

Lemma 11. *There are at most $(n(1+\frac{1}{q}))^m$ possible primal solutions $x(\phi^S)$.*

Proof. We employ a classical result of combinatorial geometry [153]. This result says that given k points in m-dimensional space, the number of possible separations, or regions, created by an m-dimensional plane is k^m.

We characterize each primal solution by a separation of k points in an m-dimensional plane by a hyperplane. In this context, each point corresponds to a value that the primal solution can take on. The number of values the primal solution $x_j(\phi^S)$ can take on is described by the size of the set T_j as defined in Definition 8. Thus, $k = \sum_{j=1}^{n} |T_j|$.

In the linear case, note that $|T_j| = 1$, $\forall j$, thus $k = n$. However in general, the size of set T_j is determined by the number of quantized values on the $[0, 1]$ interval, which is dependent on the grid size q. Recalling Definition 8, $|T_j| \leq \left(1 + \frac{1}{q}\right)$, $\forall j$. There are n such sets, and so $k \leq n(1 + \frac{1}{q})$.

Applying [153], we find that the number of possible primal solutions is equal to the maximum number of regions created by the hyperplane, which is at most $(n(1 + \frac{1}{q}))^m$. □

2.6.3 Feasibility

Now we turn to the feasibility portion of the proof; we show that the solution provided by Algorithm 1 is feasible with high probability.

Lemma 12. *Let $\mathcal{A}$ be a $(1, \alpha_d)$-approximation algorithm for packing problems, $\alpha_d \geq 1$. For any $\epsilon_s, \epsilon_f > 0$, if the conditions (2.13) and (2.14) on B hold, for the concave (2.1) and linear (2.2) cases respectively, then the solution Algorithm 1 produces is feasible with probability at least $1 - 2\epsilon_s$.*

Proof. We bound the probability that for a given sample S, the sample solution x^S is feasible for the sample problem (2.7), while there is some constraint i for which the complete solution $x(\phi^S)$ is infeasible in the original problem (2.1). Recall the events defined in Definition 4. Our goal is to bound $\Pr[D_i \cap \bar{F}_i]$.

First, we relate the sample solution x^S to $x(\phi^S)$. Due to Claim 5, we find that $\alpha_d x_j^S \geq x_j(\phi^S)$ for all $j \in [s]$. Applying Lemma 6, we find that $D_i \subset E_i$, and by Lemma 7 $\Pr[D_i \cap \bar{F}_i] \leq \Pr[E_i \cap \bar{F}_i]$. We now proceed to bound $\Pr[E_i \cap \bar{F}_i]$.

Suppose that the primal solution space $[0,1]$ is discretized with a grid size of q. Let $q = \frac{\epsilon_f \min_i b_i}{4n}$. Consider two equivalent dual variables ϕ^{S_1} and ϕ^{S_2} as defined in Definition 9. Applying Lemma 10, the inequality $q \geq \frac{\epsilon_f b_i}{4n}$, and recalling that $a_{ij} \in [0,1]$, we find:

$$\left|\sum_{j\in S} a_{ij}x_j(\phi^{S_1}) - \sum_{j\in S} a_{ij}x_j(\phi^{S_2})\right| \leq \sum_{j\in S} q \leq \epsilon_s nq \leq \epsilon_s\epsilon_f b_i/4 \tag{2.19}$$

$$\left|\sum_{j\in N} a_{ij}x_j(\phi^{S_1}) - \sum_{j\in N} a_{ij}x_j(\phi^{S_2})\right| < \sum_{j\in N} q < nq < \epsilon_f b_i/4. \tag{2.20}$$

Now for each equivalence class C of dual variables, and for each constraint i, we bound the probability of $\Pr[E_i \cap \bar{F}_i \cap \{\phi^S \in C\}]$.

In line (2.21) below, we consider any dual variable $\phi^S \in C$ in a fixed equivalence class. In line (2.22) we fix a particular dual variable Φ^C within the equivalence class C. Thus, Φ^C is not a random variable. This allows us to apply the Hoeffding-Bernstein Inequality (Theorem 3) in the next line (2.23), where the randomness present is due only to the choice of samples, indexed by j, which leads to a random selection of a_{ij}. We emphasize that $x_j(\Phi^C)$ is deterministic.

$$\Pr[E_i \cap \bar{F}_i \cap \{\phi^S \in C\}] \leq \Pr\left[\left|\sum_{j\in S} a_{ij}x_j(\phi^S) - \epsilon_s\sum_{j\in N} a_{ij}x_j(\phi^S)\right| > \epsilon_f\epsilon_s b_i \cap \{\phi^S \in C\}\right] \tag{2.21}$$

$$\leq \Pr\left[\left|\sum_{j\in S} a_{ij}x_j(\Phi^C) - \epsilon_s\sum_{j\in N} a_{ij}x_j(\Phi^C)\right| > \frac{\epsilon_f\epsilon_s b_i}{2}\right] \tag{2.22}$$

$$\leq \Pr\left[\left|\sum_{j\in S} a_{ij}x_j(\Phi^C) - \mathbb{E}\left[\sum_{j\in N} a_{ij}x_j(\Phi^C)\right]\right| > \frac{\epsilon_f\epsilon_s b_i}{2}\right] \tag{2.23}$$

$$\leq 2\exp\left(-\frac{\epsilon_f^2\epsilon_s^2 b_i^2}{2\epsilon_s b_i + \epsilon_f\epsilon_s b_i/2}\right) \tag{2.24}$$

$$= 2\exp\left(-\frac{\epsilon_f^2\epsilon_s b_i}{8+2\epsilon_f}\right) \leq 2\exp\left(-\frac{\epsilon_f^2\epsilon_s b_i}{8}\right), \tag{2.25}$$

where line (2.21) is due to Lemma 7. In line (2.22) we fix a particular dual variable Φ^C within the equivalence class C. The inequality holds due to the fact that if

$$\left|\sum_{j\in S} a_{ij}x_j(\phi^S) - \epsilon_s \sum_{j\in N} a_{ij}x_j(\phi^S)\right| > \epsilon_f\epsilon_s b_i \text{ and } \phi^S \in C, \tag{2.26}$$

then

$$\begin{aligned}\left|\sum_{j\in S} a_{ij}x_j(\Phi^C) - \epsilon_s \sum_{j\in N} a_{ij}x_j(\Phi^C)\right| &\geq \left|\sum_{j\in S} a_{ij}x_j(\phi^S) - \epsilon_s \sum_{j\in N} a_{ij}x_j(\Phi^C)\right| - \frac{\epsilon_f\epsilon_s b_i}{4} && \text{by (2.19)}\\ &\geq \left|\sum_{j\in S} a_{ij}x_j(\phi^S) - \epsilon_s \sum_{j\in N} a_{ij}x_j(\phi^S)\right| - \frac{\epsilon_f\epsilon_s b_i}{2} && \text{by (2.20)}\\ &> \frac{\epsilon_f\epsilon_s b_i}{2} && \text{by (2.26).}\end{aligned}$$

In line (2.24) we apply the Hoeffding-Bernstein Inequality (Theorem 3). To complete the proof, we take a union bound over all possible primal solutions, and the values i of the m constraints. The number of possible primal solutions is described in Lemma 11. Setting $q \geq \frac{\epsilon_f b_i}{4n}$ we find

$$2(n(1+\frac{4n}{\epsilon_f b_i}))^m \exp\left(-\frac{\epsilon_f^2\epsilon_s b_i}{8}\right) \tag{2.27}$$

$$\leq 2(n(1+\frac{4\epsilon_f\epsilon_s n}{m\log(n)}))^m \exp\left(-\frac{\epsilon_f^2\epsilon_s b_i}{8}\right) \leq \frac{2\epsilon_s}{m}. \tag{2.28}$$

In (2.27), b_i appears twice. If we make the assumption on B as in (2.13), then that also implies that $b_i > \frac{m\log(n)}{\epsilon_f^2\epsilon_s}$. Thus, (2.28) follows first because of the weaker assumption on b_i, and then because of the stronger assumption (2.13). Finally, we take a union bound over all constraints: $P(\cup_{i=1}^m (D_i \cap \bar{F}_i)) \leq m\frac{2\epsilon_s}{m} = 2\epsilon_s$.

□

2.6.4 Optimality

The following lemma describes the relation between the approximate solution $x(\phi^S)$ and the approximation of the primal objective function.

Lemma 13. *Given the dual solution (ϕ^S, ψ^S) to the sample problem (2.7), if the solution produced by Algorithm (1), $\hat{x} = x(\phi^S)$, satisfies the dual approxi-*

mate complementary slackness conditions for $r \leq 1$:

$$\forall i \quad \phi_i^S > 0 \Rightarrow rb_i \leq (A\hat{x})_i \leq b_i, \tag{2.29}$$

$$\forall j \quad \psi_j^S > 0 \Rightarrow r \leq \hat{x}_j \leq 1, \tag{2.30}$$

then $\hat{x}$ is an r-approximation of the optimal primal solution x^* original problem (2.1)*.*

Proof. Consider any approximate solution $(\hat{x}, \phi^S, \psi^S)$ that satisfies the approximate complementary slackness conditions, as stated in (2.29) and (2.30).

Recall that (2.5) motivates the allocation (2.10). Thus, (2.6) and (2.10) imply that $f_j'(\hat{x}_j) \geq a_j^T \phi^S$, $\forall j \in [n]$. Taking into account the concavity of the objective function and the assumption that $f_j(0) = 0$ for all j, we derive the following:

$$\forall j \quad f_j(\hat{x}_j) \geq \hat{x}_j f_j'(\hat{x}_j) \geq \hat{x}_j a_j^T \phi^S. \tag{2.31}$$

Recall that the dual to the sample problem (2.7) is

$$\underset{\phi \in \mathbb{R}_+^m, \psi \in \mathbb{R}_+^s}{\text{minimize}} \quad b^T \phi - \textstyle\sum_{j=1}^n f_j^\star(a_j^T \phi + \psi_j) + \mathbf{1}^T \psi,$$

where $f_j^\star(v) = \inf_{x \in \mathbb{R}_+} vx - f_j(x_j)$ is the concave conjugate function. Thus,

$$\textstyle\sum_{j=1}^n f_j^\star(a_j^T \phi^S + \psi_j^S) = \phi^S A\hat{x} + {\psi^S}^T \hat{x} - \sum_{j=1}^n f_j(\hat{x}_j). \tag{2.32}$$

So, the dual objective is

$$\begin{aligned} \mathcal{D}(\phi^S) &:= \textstyle\sum_{i=1}^m b_i {\phi^S}_i - \sum_{j=1}^n f_j^\star(a_j^T \phi^S + \psi_j^S) + \mathbf{1}^T \psi^S \\ &\leq \frac{1}{r} \textstyle\sum_{i=1}^m (A\hat{x})_i {\phi^S}_i - \sum_{j=1}^n f_j^\star(a_j^T \phi^S + \psi_j^S) + {\psi^S}^T \hat{x} \\ &= \frac{1}{r} {\phi^S}^T A\hat{x} - {\phi^S}^T A\hat{x} - {\psi^S}^T \hat{x} + \textstyle\sum_{j=1}^n f_j(\hat{x}_j) + {\psi^S}^T \hat{x} \\ &= (\frac{1}{r} - 1) {\phi^S}^T A\hat{x} + \textstyle\sum_{j=1}^n f_j(\hat{x}_j). \end{aligned} \tag{2.33}$$

The second line above follows from (2.29) and (2.30), and the third from (2.32). Thus by (2.31), the primal objective is

$$\mathcal{P}(\hat{x}) := \textstyle\sum_{t=1}^n f_j(\hat{x}_j) \geq {\phi^S}^T A\hat{x}$$

$$\Rightarrow \quad \mathcal{D}(\phi^S) \leq \frac{1}{r} \mathcal{P}(\hat{x}) \quad \Rightarrow \quad \mathcal{P}(x^*) \leq \frac{1}{r} \mathcal{P}(\hat{x}).$$

□

Next, we make a mild technical assumption.

Assumption 14. *For any dual solution ϕ^S there are at most m columns a_j of A such that $a_j^T \phi^S = f_j'(x_j)$.*

Assumption 14 does not always hold; however it can be enforced by perturbing each f_j by a small amount at random, for example as described by [59] and [4].

Claim 15. *Let x^S and ϕ^S be solutions to the sample problem* (2.7)*. Then $\{x_j(\phi^S)\}_{j\in[s]}$ and $\{x_j^S\}_{j\in[s]}$ differ on at most m values of j.*

Proof. When $f_j(x)$ is strictly concave, or equivalently $f_j'(x)$ is invertible, by allocation rule (2.11) the solutions are trivially equivalent $x_j(\phi^S) = x_j^S$ for all $j \in [s]$.

For instances in which $f_j(x)$ has piece-wise linear components, then $f_j'(x)$ may be non-invertible. In such cases, as described in the general allocation rule (2.10), we set $x_j(\phi^S)$ to be the smallest element such that $f_j'(x) \geq a_j^T \phi^S$. Thus the resulting function, which we denote $f_j'^{-1}$, may have discontinuity points.

We are concerned with instances in which $a_j^T \phi^S$ falls on a discontinuity point of $f_j'^{-1}$. In such cases it is possible that $x_j^S \neq x_j(\phi^S)$.

Here, $f'^{-1}(x)$ is a non-increasing function and thus we can apply Froda's Theorem, which describes the set of discontinuities of a monotone real valued function. The set of discontinuities is countable and is thus of Lebesgue measure zero.

By Assumption 14, there are at most m values of j for which $a_j^T \phi^S = f_j'(x_j)$. If f_j' is non-invertible in these instances, then by the above reasoning, we choose m values from a countable set of discontinuity points. Therefore, there are at most m cases in which $a_j^T \phi^S$ falls on a discontinuity point, which implies there are at most m values of j for which $x_j^S \neq x_j(\phi^S)$.

□

Now we show that the solution is approximately optimal with high probability.

Lemma 16. *Let $\mathcal{A}$ be a $(1, \alpha_d)$-approximation algorithm for packing problems, $\alpha_d \geq 1$. For any $\epsilon_s, \epsilon_f > 0$, if, the conditions* (2.13) *and* (2.14) *on B hold,* *for the concave* (2.1) *and linear* (2.2) *cases respectively, then the solution Algorithm 1 produces is a* $(1 - 3\epsilon_f)/\alpha_d^2$*-approximation with probability at least* $1 - 2\epsilon_s$.

Proof. To show the solution is approximately optimal, we bound the probability that for a given sample, the sample solution x^S causes constraints i in the sample problem to be nearly tight, while the complete solution $x(\phi^S)$ does not cause those constraints to be nearly tight in the original problem. Define events,

$$M_i = \{S \in \mathcal{S} : \sum_{j \in S} a_{ij} x_j^S \geq \frac{(1-2\epsilon_f)\epsilon_s}{\alpha_d^2} b_i\}$$
$$N_i = \{S \in \mathcal{S} : \sum_{j \in N} a_{ij} x_j(\phi^S) < \frac{1-3\epsilon_f}{\alpha_d^2} b_i\}.$$

We want to bound $\Pr[M_i \cap \bar{N}_i]$. When $\phi_i^S > 0$, the approximate dual complementary slackness condition associated with the i-th primal constraint of problem (2.7) is

$$\sum_{j \in S} a_{ij} x_j^S \geq \frac{(1-\epsilon_f)\epsilon_s}{\alpha_d^2} b_i.$$

This allows us to bound $\sum_{j \in S} a_{ij} x_j(\phi^S)$ as follows:

$$\sum_{j \in S} a_{ij} x_j(\phi^S) \geq \sum_{j \in S} a_{ij} x_j^S - m \geq \frac{(1-2\epsilon_f)\epsilon_s}{\alpha_d^2} b_i,$$

where the first inequality follows from Claim 15 and the second follows from the fact that $B \geq \frac{m\alpha_d^2}{\epsilon_f \epsilon_s}$.

We discretize the values that the primal solution can take, as done in the feasibility argument in Lemma 12. However, we now let $q = \frac{\epsilon_f \min_i b_i}{4n\alpha_d}$. Consider two dual variables ϕ^{S_1} and ϕ^{S_2} in the same equivalence class, as defined in Definition 9. Applying Lemma 10,

$$\left| \sum_{j \in S} a_{ij} x_j(\phi^{S_1}) - \sum_{j \in S} a_{ij} x_j(\phi^{S_2}) \right| \leq \sum_{j \in S} q \leq \epsilon_s n q \leq \frac{\epsilon_s \epsilon_f b_i}{4\alpha_d} \tag{2.34}$$

$$\left| \sum_{j \in N} a_{ij} x_j(\phi^{S_1}) - \sum_{j \in N} a_{ij} x_j(\phi^{S_2}) \right| < \sum_{j \in N} q < nq < \frac{\epsilon_f b_i}{4\alpha_d}. \tag{2.35}$$

For each equivalence class C of dual variables, and for each constraint i, we derive the following bound on the probability of $\Pr[M_i \cap \bar{N}_i \cap \{\phi \in C\}]$. We employ the same approach used in the analogous argument in the feasibility proof of Lemma 12, equations (2.21) and (2.22), where we fix a particular dual variable Φ^C within the equivalence class C. We bound

$$\begin{aligned}
&\Pr\left[\sum_{j\in S} a_{ij}x_j^S \geq \frac{(1-2\epsilon_f)\epsilon_s}{\alpha_d^2}b_i \cap \sum_{j\in N} a_{ij}x_j(\phi^S) < \frac{1-3\epsilon_f}{\alpha_d^2}b_i \cap \{\phi\in C\}\right]\\
&\leq \Pr\left[\sum_{j\in S} a_{ij}x_j(\Phi^C) \geq \frac{(1-2\epsilon_f)\epsilon_s}{\alpha_d^2}b_i \cap \sum_{j\in N} a_{ij}x_j(\Phi^C) < \frac{1-3\epsilon_f}{\alpha_d^2}b_i\right]\\
&\leq \Pr\left[\left|\sum_{j\in S} a_{ij}x_j(\Phi^C) - \mathbb{E}\left[\sum_{j\in N} a_{ij}x_j(\Phi^C)\right]\right| > \frac{\epsilon_f\epsilon_s}{2\alpha_d^2}b_i\right] \leq 2\exp\left(-\frac{\epsilon_f^2\epsilon_s b_i}{8\alpha_d^2+2\alpha_d\epsilon_f}\right).
\end{aligned}$$

Now take α_d close to one, i.e., we assume $10 \geq 8\alpha_d^2 + 2\alpha_d\epsilon_f$. We apply Lemma 11 for $q = \frac{\epsilon_f \min_i b_i}{4n\alpha_d}$ and find

$$\begin{aligned}
&2(n(1+\frac{4n}{\epsilon_f b_i}\alpha_d))^m \exp\left(-\frac{\epsilon_f^2\epsilon_s b_i}{10}\right)\\
&\leq 2(n(1+\frac{4\epsilon_f\epsilon_s n}{m\log(n)}))^m \exp\left(-\frac{\epsilon_f^2\epsilon_s b_i}{10}\right) \leq \frac{2\epsilon_s}{m},
\end{aligned}$$

where the last line follows first because $b_i > \frac{m\log(n)}{\epsilon_f^2\epsilon_s\alpha_d}$, and then because of the assumption made on B (2.13). Taking the union bound over values of i we find that $P(\cup_{i=1}^m(M_i\cap\bar{N}_i)) \leq 2\epsilon_s$. Finally, consider Lemma 13, for $r = \frac{1-3\epsilon_f}{\alpha_d^2}$. It follows that if x^* is an optimal solution to $\mathcal{L}$, then with probability at least $1-2\epsilon_s$,

$$\mathcal{P}(x(\phi^S)) \geq \frac{1-3\epsilon_f}{\alpha_d^2}\mathcal{P}(x^*).$$

□

Finally, the proof of Theorem 1 follows from the above results.

Proof. Proof of Theorem 1. The solution generated by the allocation rule is feasible due to Lemma 12. The guarantee of a $(1-\epsilon_f)/\alpha_d^2$-approximation of the optimal solution follows from Lemma 16. For simplicity, we state the result of Lemma 16 with a rescaling of ϵ_f by 1/3. Concerning the runtime, $\mathcal{A}$ is executed on a problem with $\epsilon_s n$ variables, and so it takes that fraction of the original runtime. Then, the second step of the algorithm is n simple computations of $f_j'^{-1}(a_j^T\phi^S)$ for all $j\in[n]$. □

2.7 Discussion

We proposed a framework for accelerating exact and approximate convex programming solvers for packing linear programming problems and a family of convex programming problems with linear constraints. Analytically, we provide worst-case guarantees on the runtime and the quality of the solution produced. Numerically, we demonstrate that our framework speeds up Gurobi and SCS by two orders of magnitude, while maintaining a near-optimal solution.

Our framework works by subsampling columns of the data matrix, and then defining a smaller sample problem defined on that subsampled matrix. We solve the dual of the sample problem using any given convex program solver in a black-box fashion. Finally, we set the values of the original primal variables based on the approximate dual solution of the sample problem.

Possible future areas of research include the following. In numerical experiments we find that our algorithm can handle a larger family of problems than suggested by our theoretical bounds on B. Understanding this gap and improving the analysis is an area of interest. Additionally, our analysis relies partly on the fact that we are concerned with packing problems in this paper. It would be interesting to see what type of techniques are useful for more general problems.

Chapter 3

DISTRIBUTED ALGORITHM WITH LOGARITHMIC COMMUNICATION COMPLEXITY

This chapter introduces a novel approach for distributed optimization in multi-agent systems. We consider a setting in which distributed agents work together to solve a global optimization problem. In a multi-agent system, an agent typically participates in a global optimization problem in order to obtain a solution to a local variable, associated with a local action for an agent. A key characteristic of this setting is that each agent participates in the global problem, but *does not necessarily need to know the full global solution.* This setting contrasts with many more general distributed settings, in which all distributed units are required to participate in calculating the entire global solution. In the work presented here, we take advantage of this change in perspective in the multi-agent setting, allowing us to guarantee exponentially reduced communication and significantly improved robustness compared to more traditional distributed algorithms that are currently employed in multi-agent systems.

3.1 Motivation

Distributed optimization is an area of crucial importance to the design and control of multi-agent systems. Despite the wide variety of approaches to distributed optimization in multi-agent systems, the approaches that are studied and used today are similar at a high level. This similarity leads to fundamental limitations on their scalability and robustness. In particular, many of the algorithms discussed in Section 1.2.2 work by passing current estimates of the global solution between agents, or a central node, and gradually improve those estimates at each step with the goal of convergence to a (near) optimal solution, i.e., consensus. Classically, in such approaches, the distributed agents are required to store, update, and broadcast a vector of dimension that matches that of the full system-wide solution to the problem at each step, which for multi-agent systems in modern applications can be enormous. Further, no individual agent can determine its own action or estimate without global convergence of all agents in the network. This is a result of the fact that distributed optimization algorithms are designed to allow each distributed agent to compute the full global solution. But, this is overkill for multi-agent systems, where typically an agent needs only to compute its local piece of the solution in order to determine its action.

As a result, there are a number of serious and fundamental challenges when it comes to applying distributed optimization algorithms in the design of multi-agent distributed systems.

First, since the network size can be enormous, consisting of tens or hundreds of thousands of distributed agents (for example, in emerging internet of things (IoT) applications, the communication and storage demands for each iteration may be extreme. In fact, in most such approaches, e.g., consensus-style approaches, the communication within a single round requires $O(n)$ messages, typically containing a current estimate of the global solution. There has been considerable research that seeks to reduce the communication overhead of these approaches, e.g., [140, 142, 181, 94]. These approaches seek to partition the global solution into multiple blocks, each of which can be communicated less frequently, thus lowering the communication overhead. However, to this point, order-of-magnitude improvements have not been found for general classes of optimization problems.

Second, the iterative convergence of traditional distributed optimization algorithms means that the convergence of *all nodes* can be delayed if a single node or communication link is congested. For example, if there is communication lag in one part of the network, a consensus algorithm cannot reach consensus, and thus no agent in the network can determine its local action. Such "stragglers" are frequent in modern distributed systems and lead to significant delays in many distributed optimization designs. The importance of this issue has been recognized for decades, and there has been considerable work toward developing asynchronous approaches for dual descent and consensus algorithms, e.g., [195, 210, 41, 20]. However, even asynchronous algorithms require all nodes to communicate repeatedly in order for consensus to be achieved. Thus, if a set of agents is suffering from poor communication conditions, agents across the network must still wait for that part of the network to converge in order to determine their actions.

Third, classical approaches result in designs where any changes in network structure due to communication links failing or agents entering/leaving the network means that the algorithm is brought to a halt and needs to restart the convergence process. Again, this is a long-standing issue and the design of fault tolerant distributed optimization has received considerable attention. Robustness to failures and changes in the system are typically addressed through the design of fault-tolerant, Byzantine distributed optimization approaches, e.g., [43]; however, such approaches require significant adjustments to the classical algorithms and come at significant expense in terms of convergence rates and optimality guarantees.

Fourth, because classical distributed algorithms require *global* convergence or consensus before any individual agent can determine its local action, a single agent computing its individual action or estimate imposes communication and computation demands on every agent in the network. This introduces unnecessary overhead and delay since it means that an individual agent is impacted by stragglers, agents entering/exiting, etc., across the *whole system* even though it only seeks to compute its *local* action. Ideally, an agent would be able to compute its part of the solution without the need to compute the full global solution.

Goal. *In this work, we seek to develop a new approach for distributed optimization in multi-agent systems that can reduce the communication overhead of traditional approaches, while also guaranteeing robustness to communication delay and failures in the system. To accomplish this, we seek a design that allows an individual agent to compute its local optimal action without the need for global communication.*

Our approach toward achieving this goal is to develop a novel connection between distributed optimization and an emerging sub-field of theoretical computer called *local computation algorithms* (LCAs) [174] – applying local computation algorithms to optimization problems for the first time. The LCA framework was formally introduced by [174] in order to connect a variety of algorithms with similar goals that had recently appeared in distinct areas [176, 10, 98]. Until our work, the field has focused on the design of LCAs for graph problems such as matching, maximal independent set, and coloring [7, 112, 169, 75]. In this work we show that the approach is promising for distributed optimization as well.

The defining property of local computation algorithms is that they seek to compute a local "piece" of the solution to some algorithmic problem using only information that is “close” to that piece of the problem. For example, an LCA for matching allows each node in the graph to compute its own match locally by communicating only with a small neighborhood of other nodes, without computing the entire matching for the graph. Yet, if all nodes run the LCA, then the solution each node computes is part of the same global matching.

In the context of distributed optimization in multi-agent systems, this means that when running an LCA, a distributed agent computes its own action or estimate (its local piece of the solution to the global optimization problem) *without computing or communicating the global solution.* However, if every agent runs the LCA, then the agents together (approximately) solve the global optimization problem, i.e., compute pieces of the same global solution. So, if there exists an LCA for the optimization problems used in networked and distributed systems, it would allow an agent to compute its local action without waiting for global consensus to be achieved. Thus, it could provide a significant reduction in communication compared to traditional approaches while also improving robustness to stragglers and agents entering/exiting the system, since stragglers and agents entering/exiting would only impact an agent's computation of their action if they happen within the small, local neighborhood of the agent.

In this work we develop the first local computation algorithm for convex optimization, LOCO (LOcal Convex Optimization). This optimization framework represents a fundamentally new approach for distributed optimization in multi-agent systems that allows an individual agent to compute its action with *exponentially less communication* than traditional approaches, while maintaining robustness to both stragglers and the entrance/exit of agents into the system. Further, LOCO allows an individual agent to compute its action or estimate without the need for global convergence, and thus without the need for global communication and computation.

3.1.1 Problem Overview and Distributed Setting

We consider a multi-agent system with N distributed agents that wish to compute actions or estimates $x_j \in \mathbb{R}^{q_j}$, where q_j is the dimension of the actions for agent j, so that the combination of the actions forms a global solution $x \in \mathbb{R}^n$ to a constrained optimization problem of the following form. This form is of interest for a wide variety of problems in multi-agent networked systems, e.g., regression problems and support vector machines [53, 95, 93], distributed inference in sensor networks, which has broad applications to the Internet of Things [87, 158, 152, 85], inference in graphical models [166, 5], relaxations of maximum a posteriori (MAP) estimation problems [178], network utility maximization (NUM) problems, [99, 119], management of content distribution networks and data centers [28, 154], and control of power systems [71, 160]:

$$\begin{array}{rll} \text{minimize} & \sum_{j=1}^{N} f_j(x_j) & \\ \text{subject to} & \sum_{j=1}^{N} a_{ij} x_j \geq b_i & i \in [m] \\ & x \geq 0, & \end{array} \tag{3.1}$$

where $x_j \in \mathbb{R}^{q_j}$ and $f_j : \mathbb{R}^{q_j} \to \mathbb{R}$ are convex functions. We allow overlap or coupling between the functions f_j; i.e., a component of the entire solution $x \in \mathbb{R}^n$ may appear in multiple local functions f_j. When this happens, the agents' actions are coupled through the overlapping variables. Formally, this implies that $\sum_{j=1}^{N} q_j \geq n$, where if there is equality there are no variables that appear in multiple agents actions, x_j, and if the inequality is strict there are variables that appear in two or more agents' actions. Additionally, $a_{ij} \in \mathbb{R}^{\leq q_j}$ are submatrices of an $A \in \mathbb{R}^{m \times n}$ matrix, where $[a_{i1}, \ldots, a_{ij}, \ldots, a_{in}]$ is the ith row of A, and $b \in \mathbb{R}^m$.

The problem is defined over a network, where each agent j is associated with a node j, a variable x_j, and a function f_j. The problem data (m constraints), A and b, are distributed over the agents, and the N agents are completely distributed. In this work, we are concerned with settings in which n, m, and N are large, but each local function f_j depends on a relatively small number of components of x, i.e., the dimension of the agents' actions is small, and the matrix A is sparse (has a small number of non-zero entries in each row and column).

We would like to emphasize that the task for an individual, distributed agent is to compute its own local action or estimate, x_j. The agent does not need the full global solution x, only its local piece. Note that in traditional approaches for distributed optimization, e.g., consensus and dual descent, a byproduct of the algorithms used is that each agent computes the full global solution x, which may be of significant size and requires global convergence (and thus communication and computation by every agent in the network) to compute. This should not be viewed as a *feature* of these algorithms, instead it is an unnecessary *overhead* in the case of multi-agent systems (since the agent is only responsible for its local action).

A key insight in the design of LOCO is that is not necessary for an individual agent to compute the global solution in order to determine its individual action. Instead, *it is possible for an agent to compute its local "piece" of the solution x_j without computing the full global solution x.* To achieve this, the fundamental idea of LOCO is to, for a given distributed agent, define a *local problem* associated with the agent's action (variable) x_j, which is defined on a subset X_j of the primal variables and a subset Y_j of the data (constraints). The agent j then solves its local problem using a given algorithm that is purely local. This produces the local action/estimate, x_j, that is a piece of an (approximately) optimal solution to the global problem x. Further, if every distributed agent runs the same local algorithm, then x is computed.

Note that the sets X_j and Y_j used by LOCO are much smaller in size than n and m respectively (the dimensions of the original problem), resulting in a dramatic dimension reduction and thus a reduction in communication and computation when the matrix A is sparse. We show (Theorem 19) that, when the data matrix A is sparse, i.e., the maximum number of non-zero entries in a row or column is bounded by a constant, X_j and Y_j both have sizes on the order of $O(\log m)$. We utilize this to guarantees that a small number of messages needs to be passed, and that the messages passed are small in size.

We provide worst-case guarantees on the performance of LOCO with respect to the amount of communication it requires and the quality of the solution. Regarding communication, the process of determining sets X_j and Y_j requires $O(\log m)$ messages with high probability. After this step, solving the local problems at each node requires no communication. Note that this is an exponential reduction compared to the $O(n)$ communication required during *each round* of traditional approaches such as consensus and dual descent when A is sparse. We also provide worst-case guarantees on the performance of LOCO with respect to the quality of the solution. Since the nodes do not have access to the entire problem under LOCO, it is unreasonable to expect an exact solution. Instead, LOCO produces a feasible, α-approximation of the optimal solution, where α depends on the given algorithm $\mathcal{A}$ used to solve the local problem of an agent (Theorem 19). Our numeric results in Section 3.6 highlight that the approximation error of LOCO matches that of ADMM in many cases, while using orders-of-magnitude less communication.

3.1.2 Approach

To develop algorithms to solve the local problem of an agent, we prove a reduction that allows generic online algorithms to be "converted" into local optimization algorithms. This approach is based on an insight in a foundational result in the local computation literature, which shows that online algorithms can be converted into local algorithms with the same performance guarantee in *graph problems* with bounded degree [126, 169]. Our contribution is to, for the first time, show that a similar reduction is possible for *optimization problems*, where the bounded degree property is replaced by the sparsity of the constraint matrix. This enables us to prove that if an online algorithm, $\mathcal{A}$, running on global information is guaranteed to output an α-approximate solution, then when LOCO uses the algorithm to compute the local solution of an agent the resulting solution is also an α-approximation. Thus, the LOCO framework inherits the approximation ratio of $\mathcal{A}$.

To illustrate the power of this reduction and the generality of LOCO, we provide specific results for two different classes of optimization problems of significant practical interest (Corollaries 20 and 21). These two results use two different online algorithms as the algorithm for solving the local problem of an individual agent in LOCO, thus highlighting the generality of the LOCO framework. Specifically, we show that LOCO achieves an $O(\log m)$-approximation in the case that the objective functions are linear and an ϵ-approximation in the case of linear SVM problems. Beyond these theoretical guarantees, we also provide numerical case studies for these two examples in Section 3.6. The case studies show order-of-magnitude improvements in communication time are possible using LOCO, and that this is possible without incurring excessive approximation error.

3.2 Related literature

Distributed optimization is a field with a long history. In the 1960s, approaches emerged for solving large scale convex programs in a distributed manner. Early approaches include [56, 22, 74, 173, 195, 24].

Distributed optimization algorithms can be broadly categorized into dual decomposition methods [191], subgradient methods [140, 142], and proximal gradient methods [181]. Many of these distributed algorithms use consensus methods as a way to distribute computation among the agents. For example, ADMM is a popular dual decomposition method, introduced by [77] that can be implemented in a consensus setting [31]. Variants of consensus ADMM have been studied in the context of support vector machines [76] and generally in distributed model fitting [72, 139, 83]. ADMM has also found broad applications in denoising images [187] and signal processing [52, 179]. Despite the success of ADMM and other techniques for distributed optimization, they tend to require significant memory storage at each node, and suffer form large communication costs. For example, distributed dual decomposition methods typically requires several rounds of communication between neighbors, and use as many as $O(n)$ messages at each round, where n is the number of nodes in the graph. In our work, we propose a technique that is lighter in both communication and computation, and is more robust to stragglers and the entry/exit of agents.

Within the networked control and communication networks literature, there is a large body of work on distributed algorithms [99, 119, 44]. Dual decomposition algorithms are particularly prominent in this setting. For example, [199] propose a novel approach for solving the network utility maximization problem. See [156] for a survey of distributed algorithms for NUM. Additional recent distributed dual decomposition algorithms include [138, 42].

More broadly, distributed computation is an active field today. Some recent work that is connected to the work in this chapter includes [124], which proposes a distributed decomposition method based on passing gradient information between nodes with the goal of limiting communication. Work by [101] also propose a gradient based approach, one in which gradient compression techniques are utilized to improve iteration and communication complexity for the gradient descent algorithm. More recent consensus based asynchronous distributed approaches include [210, 41, 20]. Additionally, [94] introduce a decomposition method which seeks to decrease required communication by solving smaller subproblems at each node. The subproblems are defined on a subset of the variables, which is similar to our approach.

We emphasize that the above approaches and other decomposition based methods [124, 123, 101, 94, 210, 41, 20] differ from our approach in the form of messages passed. The methods discussed above send local copies of the solution vector $x \in \mathbb{R}^n$, or gradient information, to maintain consensus. Thus, these messages are typically a vector in $\mathbb{R}^n$. We however send small pieces of the constraint matrix, A. Since the A matrix is very sparse, this amounts to only sending several matrix coefficients a_{ij} at a time, along with their index information (i, j). Thus, our messages are extremely lightweight, and throughout this chapter, any comparison via the number of messages to other algorithms, is a conservative estimate of the benefits of LOCO.

Another key difference between the approaches used in [94, 101] and LOCO is a trade-off between the cost of sending messages versus the cost of doing heavy computation at each node in the graph. In [94], messages are sent between all neighboring nodes at each iteration, making it relatively message heavy; in contrast, LOCO sends very few messages. In terms of computation, however, [94] does very light computation in each step at each node while LOCO does more computation locally. The choice of which approach to use depends on which is more expensive: communication or computation.

Another way in which our framework differs from dual decomposition is that it does not require every node to converge to the full, global solution, i.e., consensus. In particular, our framework provably produces an α-approximation to the optimal local action in a logarithmic number of steps, whereas dual decomposition and consensus algorithms require analysis of convergence rates and stopping criteria.

Stragglers and failures have been major obstacles for the distributed optimization literature during the past decades. Two prominent goals are (i) the design of asynchronous algorithms and (ii) providing Byzantine faulty tolerance. For both of these goals, the challenge is to be robust to communication delays or unreliability in the system. In asynchronous computation, the goal is to compute the solution when distributed agents do not report updates in a reliable way [195, 148, 157]. In the Byzantine faulty tolerance, some components of the distributed system are unreliable and perhaps adversarial [43]. Our work shares these goals, but approaches them in a different way; we design a new way to distribute the computation between the agents, requiring less communication and thus approaching robustness to failures differently.

Some of the key insights behind LOCO are based on a field of theoretical computer science: *local computation algorithms* (LCAs) [174]. Most of the focus of research in this field has been on graph problems such as matching, maximal independent set, and coloring [7, 112, 169, 75, 1]. Our work contributes to the

LCA literature by moving from graph problems to the more general domain of distributed convex optimization, which has not been studied previously.

Two other related lines of work are the distributed $\mathcal{LOCAL}$ and $\mathcal{CONGEST}$ models [159], in which the complexity of a protocol is measured by the number of *rounds* required. Of particular relevance is [103], which concerns solving packing linear programs in a distributed manner in the $\mathcal{LOCAL}$ model. We note that our algorithm can be implemented in the $\mathcal{LOCAL}$ (and $\mathcal{CONGEST}$) models, in $O(\log n)$ rounds; the algorithm of [103], while using a polylogarithmic number of rounds, can use as much as linear communication if the diameter of the network is small.

Lastly, our approach shares characteristics with *sketching* and *leverage score sampling* [201], in that a subset of the rows of the data matrix A are selected and a problem of smaller dimension is solved. However, our work differs from these approaches significantly. For example, we select a block of the matrix, or a subset of both rows and columns. Thus the dimension of the problem is reduced in both the number of variables and the number of data points.

3.3 Problem Formulation

We study a multi-agent system with N distributed agents and no central control. The agents may communicate with neighbors, but communication with a centralized processing unit is prohibited. The system is designed such that the agents seek to compute their local actions or estimates x_j for each of the N agents and that the set of all agent actions solves a constrained convex optimization problem with a separable objective and coupled constraints. Specifically, the agents together seek to solve an optimization problem of form (3.1).

The problem is defined over a network G, and each agent j is associated with a node j, a variable x_j, and a function f_j. The problem data (constraints), A and b, are distributed over the agents. Each agent has a subset of the constraints, or rows of the A matrix. There may be copies of the same constraints at different nodes.

In our algorithm, the problem is represented as a hypergraph $\mathcal{H} = (V, H)$. The set of nodes in the hypergraph $V = \{1 \dots N\}$ is the same as that of G, where each node corresponds to a variable x_j. Hyperedges $H = \{1 \dots m\}$ correspond to constraints. We associate each constraint with a dual variable $y_i \; \forall i \in [m]$, and refer to primal constraints and dual variables interchangeably.

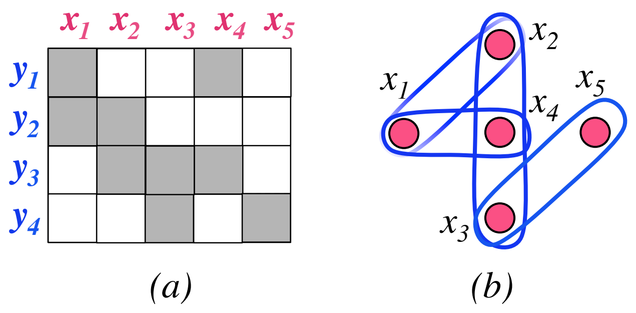


Figure 31: The constraint matrix is depicted in (a), and the *hypergraph* $\mathcal{H}$ in (b). Red shaded nodes represent the primal variables and blue hyperedges represent the constraints, or dual variables. Hyperedges encircle primal variables which appear together in a constraint.

As an example, in Figure 31, nodes encircled by a hyperedge correspond to primal variables that appear together in a constraint.

We measure the performance of an algorithm in this setting with respect to the amount of communication it requires and the quality of the solution it produces. To measure the amount of communication, we define a *message* to be information that is sent between neighbors in a graph and we define *message complexity* to be the number of messages sent across edges in order to compute the solution. In our setting, small pieces of the constraint matrix, A, are passed between nodes. Since the A matrix is very sparse, this amounts to only sending several matrix coefficients a_{ij} at a time, along with their index information (i, j), and the coefficient b_i. We define a message with respect to each ith constraint to be the list of matrix coefficients $\{a_{ij} \forall j \in [n] : a_{ij} \neq 0\}$, for a given ith row of the matrix A, along with the coefficient b_i.

When the algorithm uses randomization, we prove bounds on the message complexity that hold with probability at least $1 - \frac{1}{m^\gamma}$, where m is the number of constraints and $\gamma > 0$ can be an arbitrarily large constant. We denote this by $1 - \frac{1}{\operatorname{poly} m}$. We do not bound the size of the messages, but note that in both our algorithm and most dual descent and consensus algorithms the message lengths are of order $O(\log(n + m))$.

By default, the graph we consider communication over is the hypergraph $\mathcal{H}$. However, we can also describe communication with respect to the physical network G. The difference between these is a function of the *sparsity* of A, which we define as $d = \max\{d_r, d_c\}$, where d_r and d_c denote the maximum number of nonzero entries in rows and columns of A respectively. We say that

A is *sparse* if the sparsity of A is bounded by a constant. Thus, given that the constraint matrix A has sparsity d, the number of messages required on G compared to $\mathcal{H}$ differs by a factor of at most d^2.

To measure the quality of the solution of an algorithm in this setting we use the *approximation ratio*. An algorithm is said to produce an α-approximate solution if its solution is guaranteed to be at most αOPT, where OPT is the value of the optimal solution. In our empirical results, we compare the performance of LOCO to the dual decomposition method ADMM, for which approximation ratio is not a standard measure. Thus, empirical comparisons are made using *relative error*, defined in Section 4.7, which is related to, but different from, the approximation ratio.

This setting and the performance measures we use are of broad interest in multi-agent systems. For example, the setting has been considered in regression problems and support vector machines [53, 95, 93], distributed inference in sensor networks, which has broad applications to the Internet of Things [87, 158, 152, 85], inference in graphical models [166, 5], relaxations of maximum a posteriori (MAP) estimation problems [178], Network Utility Maximization (NUM) problems, [99, 119], management of content distribution networks and data centers [28, 154], and control of power systems [71, 160]:

In this thesis, we use two examples to highlight the generality of the LOCO framework: NUM and SVM, which we describe in Subsections 3.3.1 and 3.3.2 respectively. With the example of SVM, we also highlight the potential for LOCO to be used in settings that are not fully distributed.

3.3.1 Network Utility Maximization (NUM)

To illustrate the application of LOCO to multi-agent systems, we focus on the example of NUM, which is a general class of optimization problems that has seen widespread applications in multi-agent systems, from the design of TCP congestion control [99, 119, 120, 186] to understanding of protocol layering as optimization decomposition [44, 156] and power system demand response [177, 114]. For a recent survey on NUM see [206].

The NUM framework considers a network containing a set of sources (agents) $\mathcal{S} = \{1, \dots, m\}$ and links $\mathcal{L} = \{1, \dots, n\}$ of capacity c_j, for $j \in \mathcal{L}$. Source $i \in \mathcal{S}$ is characterized by $(L_i, f_i, \underline{x}_i, \bar{x}_i)$: $L_i \subseteq \mathcal{L}$ is a path in the network; $f_i : \mathbb{R}_+ \to \mathbb{R}$ is a concave utility function; $\underline{x}_i$ and $\bar{x}_i$ are the minimum and maximum transmission rates of source i respectively.

The goal of a source is to determine its rate x_i such that the aggregate utility of all sources is maximized. Source i attains a concave utility $f_i(x_i)$ when it transmits at rate x_i along path L_i, within the minimum and maximum rates allowed. The maximization of aggregate utility is formulated as

$$\begin{aligned} \text{maximize} \quad & \textstyle\sum_{i=1}^{m} f_i(x_i) \\ \text{subject to} \quad & A^T x \leq c \\ & \underline{x} \leq x \leq \bar{x}, \end{aligned} \tag{3.2}$$

where $A \in \mathbb{R}_{+}^{m \times n}$ is defined as $A_{ij} = 1$ if $j \in L_i$ and 0 otherwise.

Different choices of f_i correspond to different network goals. Some of the most common in networking settings are (i) setting $f_i(x_i) = x_i$ to maximize throughput; (ii) setting $f_i(x_i) = \log(x_i)$ to achieve proportional fairness; (iii) setting $f_i(x_i) = -1/x_i$ to minimizes potential delay [119, 129].

While consensus and dual descent methods have received considerable attention in the NUM literature, note that sources *do not* need to know the global solution. They only need to know their local rate, x_i. Thus, NUM is a natural application where local computation can provide significantly reduced communication and improved robustness by eliminating the demand that every agent converge to the full, global solution.

We use numerics in Section 3.6 to show the improvements LOCO provides compared to classical approaches for NUM. In these examples, we focus on $f_i(x_i) = x_i$, i.e., maximizing throughput, since it is typically viewed as the most challenging. However, the LOCO framework can be applied to any NUM objective.

3.3.2 Support vector machines (SVMs)

Federated machine learning is an increasingly prominent framework that seeks to train machine learning models in settings where data is distributed among multiple agents due to privacy concerns. This approach has received significant attention from researchers in recent years, e.g., [102], and appears in industry as well, e.g., [130]. Inspired by this, our second example illustrates how LOCO can be used for distributed training of an SVM.

SVMs represent a core model in machine learning that is crucial for applications in both regression and classification. While there are many variations of SVMs, we use the following classical version to illustrate the application of LOCO. We consider the task of fitting an SVM to data pairs $S = \{(z_i, y_i)\}_{i=1}^m$, where $z_i \in \mathbb{R}^n$ and $y_i \in \{+1, -1\}$ is a label for each data pair. Traditionally, this problem is presented as a regularized optimization problem of the following form:

$$\text{minimize}_x \textstyle\sum_{(z_i,y_i)\in S} \max\{0, 1 - y(x^T z)\} + \lambda ||x||_2^2. \tag{3.3}$$

As stated the above optimization does not match the form of (3.1), however there are a number of standard tranformations that lead to matching forms. For example, we use the case of linear SVMs to illustration LOCO. For linear SVMs, (3.3) can be written in the following form [95], which matches (3.1):

$$\begin{aligned} \underset{x,\xi_i \geq 0}{\text{minimize}} \quad & \frac{1}{m} \textstyle\sum_{i=1}^m \xi_i + \lambda ||x||_2^2 \\ \text{subject to} \quad & y_i(x^T z_i) \geq 1 - \xi_i, \quad \forall i \in [m]. \end{aligned} \tag{3.4}$$

Here, the local variables associated with the agents are ξ_i, and these can be computed in a completely distributed way using the LOCO framework, see Section 3.6 for experiments demonstrating the performance and robustness improvements of this approach.

This application highlights another point about LOCO. It can be applied in both distributed and parallel settings. In particular, if the goal is to determine the whole global solution, i.e., the full SVM model, then one simple "join" step where each agent sends the solution to a central entity accomplishes this. Thus, LOCO can be used to provide a parallel SVM implementation that is robust to stragglers and failures of compute nodes.

3.4 A Local Optimization Framework

In this section we introduce the framework that is the main contribution of this chapter of the thesis: *LOcal Convex Optimization* (LOCO). We describe the framework and give intuition for it in this section and then, in the next section, we focus on providing provable guarantees on communication and accuracy.

LOCO consists of two steps. In the first, LOCO generates a (small) localized neighborhood for each variable or source. In the second, LOCO simulates an online algorithm on the localized neighborhood. Note that the first step is independent of the online algorithm, and the second is independent of the

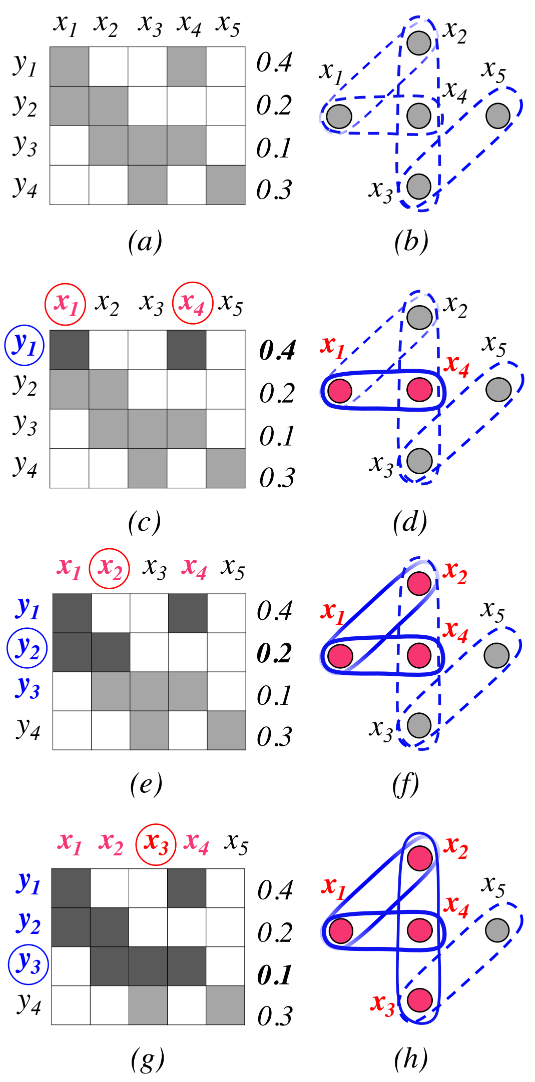


Figure 32: An illustration of LOCO. The constraint matrix A is depicted in (a), where shaded entries represent non-zeros. The rankings of the constraints are indicated next to their corresponding rows in A. The *hypergraph* $\mathcal{H}$ is depicted in (b). Figures (c)-(h) illustrate the construction of sets X_1 and Y_1 for the local problem associated with variable x_1. The darkest shaded matrix elements in (c), (e), and (g) indicate constraints as they are received by agent 1. Blue emboldened hyperedges in (d), (f), and (h) represent constraints added to Y_1. Red shaded nodes represent variables added to set X_1. The process stops in (g) because rankings $0.1 < 0.3$. The local problem associated with variable x_1 is defined on variables $X_1 = \{1, 2, 3, 4\}$ and constraints, or dual variables, $Y_1 = \{1, 2, 3\}$.

method used to generate the localized neighborhoods. Therefore, one should think of LOCO as a general framework that can yield a variety of algorithms for different classes of optimization problems depending on the online algorithm it is instantiated with. For example, we can use different online algorithms for the second step of LOCO depending on whether we consider NUM or SVM, as we do in the next section.

More specifically, the details of the two follow and are summarized in Algorithm 3 below.

Step 1: Set up the Local Problems. For each agent $j \in V$, define an associated *local problem*, consisting of a subset X_j of the primal variables and a subset Y_j of the constraints, or dual variables. The local problem is of the form:

$$\begin{array}{rlr} \text{minimize} & \sum_{k \in X_j} f_k(x_k) & \\ \text{subject to} & \sum_{k \in X_j} a_{ij} x_k \geq b_i & i \in Y_j \\ & x_k \geq 0 & k \in X_j, \end{array} \tag{3.5}$$

In order to construct sets X_j and Y_j, first generate a random ordering on the constraints. Let $r : [m] \to [0,1]$ be a function that assigns each constraint, or dual variable y_i, a real number between 0 and 1 uniformly at random. We call $r(i)$ y_i's *rank*. For more about generating r efficiently, see Section 3.8.3. We assume that all of the nodes have access to r, and hence can compute the rank of any constraint.

Construct X_j and Y_j as follows. At node j, calculate the rank of each of the constraints in which variable x_j appears, i.e., for all i such that $a_{ij} \neq 0$. Among these constraints, identify the index of the highest ranked constraint: $h = \text{argmax}_j\{r(i)|a_{ij} \neq 0\}$. Add constraint h to set Y_j. In a recursive fashion, at a given node j', contact j''s neighbors in $\mathcal{H}$ to learn which constraints they appear in: $i \in [m]$ s.t. $a_{ij'} \neq 0$. At node j', calculate the ranks of each of these constraints. Add to Y_j the constraints that have lower rank than the constraint most recently added to Y_j, i.e., $r(i) < r(h)$, and add to X_j the primal variables that appear in those constraints. Repeat this process until all visited neighbors appear in constraints that have higher rank than the last constraint added to Y_j. This process is stated concretely in Algorithm 3; see Figure 32 for an example.

Step 2: Solve the Local Problems. The jth agent solves the jth local problem (3.5) using any existing convex optimization algorithm that is a local sequential algorithm in the following sense.

Definition 17. *A* local sequential algorithm *for problems of form* (3.1) *is one that observes input sequentially. Assume that the constraints arrives according to some order π, for simplicity, we set $\pi(i) = i$; that is, the constraint associated with the dual variable y_i arrives at step $i \in [m]$. At step $i = 1, \ldots, m$, only y_i and x_j such that $a_{ij} \neq 0$ are possibly updated, and their new values depend* only *on the value (at step i) of primal variables x_j such that $a_{ij} \neq 0$.*

Note that all the updates the local sequential algorithm makes at step i are based only on the values of x_j $\forall j \in V$ for which $a_{ij} \neq 0$ when y_i arrives. Local sequential algorithms include most online algorithms, such as the algorithms in [34] for covering or packing linear programs; those in [14] for convex covering and packing problems with linear constraints; and in [68] for general convex conic covering problems. For example, NUM is a packing problem with linear constraints, and thus LOCO can be run with the algorithms of [34] or [14]. Local sequential algorithms also include many stochastic gradient descent methods, where data is drawn randomly at each step. An example is the Pegasos algorithm for SVMs [180], which at each iteration operates on a single training example. Note that our setting is *offline*; however, if we use an online algorithm, we *simulate* it in an offline setting.

Let r be the ranking function for constraints as defined in Step 1 and let $\mathcal{A}$ be any sequential algorithm that receives the constraints in the order defined by r. Note that the jth local problem contains precisely the variables and constraints that $\mathcal{A}$ considers when deciding the value of x_j.

In order to solve the jth local problem, the constraints Y_j are considered sequentially in the order assigned by the ranking r. At each step, LOCO simulates the arrival of a constraint in Y_j, in the order implied by r, and the variables in X_j are updated. We assume the univariate non-negativity constraints do not arrive sequentially and are known initially. In $|Y_j|$ steps, the algorithm produces some solution for all the variables in X_j, which includes the desired x_j component of the solution. At this point, x_j's value is identical to its value in the solution produced by $\mathcal{A}$: the construction of Y_j and X_j guarantees that it has been updated precisely as it would have been in the execution of $\mathcal{A}$, up to the point when h, the highest ranked constraint that contains x_j arrived. Clearly, x_j will not be updated at any point afterwards, by the definition of the sequential algorithm. As the value of x_j when solving the jth local problem is identical to its value when executing $\mathcal{A}$ on the entire problem (1) for every j, we get the following lemma, whose proof is deferred until Section 18.

Lemma 18. *Let $[x_1^*, \dots, x_N^*]$ be the solution obtained if the sequential algorithm $\mathcal{A}$ is run on the original problem (3.1), and let $\hat{x}_j$ be the solution obtained by solving the jth local problem (3.5). Then if $[x_1^*, \dots, x_N^*]$ is an $h(n,m)$- approximate solution to (3.1) then $[\hat{x}_1, \dots, \hat{x}_N]$ is also an $h(n,m)$- approximate solution to (3.1).*

Contrasting LOCO with classical approaches. From the description above, it is clear that LOCO fundamentally differs from dual decomposition and consensus methods. Dual ascent and consensus methods iterate until *global* optimality conditions are met. In order to check for global optimality, methods such as ADMM typically require communication among all nodes in the distributed network at each iteration. LOCO operates in a completely different way; when constructing sets X_j and Y_j, the jth node only interacts with nodes in X_j, and then solves its local problem without requiring further communication beyond that set of nodes. Thus, communication is strictly localized and there are no multiple rounds of communication.

As a result, there is a difference in the form of the theoretical guarantees for LOCO and dual decomposition/consensus algorithms. Convergence rate bounds are the goal when studying dual decomposition and consensus methods. In contrast LOCO is a framework that inherits the convergence or stopping criterion of the local sequential algorithm employed. LOCO executes for a predetermined number of steps, which is the size of set Y_j. In contrast, for ADMM the number of iterations required is unknown a priori (though it can be bounded). LOCO produces an $h(n,m)$-approximation to the solution in exactly $|Y_j|$ steps.

3.5 Theoretical Results

In this section we provide results that bound the communication demands of LOCO and the quality of the solution it produces. The key insight in the design of LOCO is that it is possible to convert any local sequential algorithm into a distributed algorithm. We prove that the resulting distributed algorithm has the same approximation ratio as the original local sequential algorithm. In particular, our main theoretical result shows that LOCO provides solutions to convex optimization problems that are as close to optimal as those

Algorithm 3: LOCO (LOcal Convex Optimization)

```
Input: Convex Program of form (3.1), sequential algorithm $\mathcal{A}$, ranking
$r : [m] \to [0, 1]$, index of agent $j$
Output: $\hat{x}_j$
Initialize: Calculate the rank of the constraints for all $i$ such that $a_{ij} \neq 0$.
Let $h$ be the index of the highest ranked constraint: $h =$
$\text{argmax}_j \{r(i) | a_{ij} \neq 0\}$
Step 1: Find sets $X_j$ and $Y_j$ associated with $x_j$.
$X_j = \emptyset$; $Y_j = \{h\}$
ptr = 1; endptr = 2
while ptr < endptr do
    $h = Y_j(\text{ptr})$ ;
    ptr++
    for all $j' \in [n]$ s.t. $a_{hj'} \neq 0$ do
        for all $i \in [m]$ s.t. $a_{ij'} \neq 0$ do
            if $j' \notin X_j$ then
                $X_j \leftarrow X_j \cup \{j'\}$
            if $r(i) < r(h)$ then
                if $i \notin Y_j$ then
                    $Y_j \leftarrow Y_j \cup \{i\}$
                    endptr++
Step 2: Use $\mathcal{A}$ to solve the local problem (3.5) defined on $X_j$ and $Y_j$.
Constraints arrive in the order determined by $r$.
```

of the best local sequential algorithms for the problems, while using exponentially less communication than classical distributed optimization algorithms. Further, because each agent computes its local piece of the solution without global communication, LOCO provides significant improvements in robustness compared to traditional consensus-based and dual descent-based approaches.

3.5.1 Results on communication complexity and solution quality

Our main result is summarized in the following theorem.

Theorem 19. *Let P be a convex problem of form* (3.1)*, where $A \in \mathbb{R}^{m \times n}$ has sparsity d. Consider LOCO instantiated with a local sequential algorithm $\mathcal{A}$ for P with approximation ratio $h(n, m)$. Each agent $j \in V$, where $|V| = N$, independently computes $\hat{x}_j$ using at most $2^{O(d^2)} q_j \log m$ messages with probability $1 - 1/\operatorname{poly}(m)$. The resulting complete solution $[\hat{x}_1, \ldots, \hat{x}_N] \in \mathbb{R}^n$ provides an $h(n, m)$-approximate solution to P.*

This result shows that there is no performance loss when converting the local sequential algorithm to a distributed algorithm using LOCO. Further, for sparse graphs (where d is a constant), the communication demands are logarithmic, as opposed to linear like in consensus based algorithms.

Theorem 19 provides a general result, but it is also useful to illustrate this result for specific local sequential algorithms. In particular, LOCO can be used broadly for any class of optimization problems for which local sequential algorithms exist. Thus, improvements to local sequential and online algorithms immediately yield improved distributed algorithms.

We illustrate this with the following corollaries for the cases of linear programs and linear SVMs. These two corollaries provide the basis for the case studies for NUM and SVMs in Section 3.6.

In the case of NUM, we focus on the goal of throughput maximization, which means that the objective is linear. In this case, we can use the online algorithm from [34] for packing linear programs, which yields the following corollary.

Corollary 20. *Given a linear program with n variables, m constraints, and a sparse constraint matrix, each agent $j \in V$, where $|V| = N$, independently computes $\hat{x}_j$ using at most $O(\log m)$ messages with probability $1 - 1/\operatorname{poly}(m)$. The resulting complete solution $[\hat{x}_1, \ldots, \hat{x}_N]$ provides an $O(\log m)$-approximation.*

In the case of SVM, we focus on the linear SVM problem described in (4.24). In that case, we can apply the Pegasos [180] algorithm, which yields the following result.

Corollary 21. *Given a linear SVM problem with n variables, m constraints, where d is a bound on the number of nonzero features in each example, and λ is the regularization parameter, each agent $j \in V$, where $|V| = N$, independently computes $\hat{x}_j$ using at most $O(\log m)$ messages with probability $1 - 1/\operatorname{poly}(m)$. The resulting complete solution $[\hat{x}_1, \ldots, \hat{x}_N]$ provides an ϵ-approximation.*

Note that we focus on NUM with a linear objective, but LOCO is not limited to linear objectives and Theorem 19 can be applied to NUM with a general convex objective function,for example, using the algorithm in [14].

Now that we have concretely stated both the algorithm and results, we see how LOCO lends itself to the robustness properties outlined in the introduction. As stated in Theorem 19, setting up each local problem requires at most $2^{O(d^2)} \log m$ messages. This bound on communication implies that an agent will only communicate with a logarithmically bounded number of agents, constituting a small neighborhood around the agent. This behavior makes the computation robust to failures or delays; a failure will only effect nearby agents, leaving agents outside of the logarithmically bounded neighborhood unaffected. Similarly, if a new agent enters the system, only agents in the logarithmically bounded neighborhood must share new data and recompute. This is in contrast to the large body of distributed optimization algorithms, which typically require all of the agents to update computations if a new agent enters the system. Additionally, after the initial round of communication, each local problem is solved independently at the corresponding node with no further communication. This design increases the robustness of LOCO to failures; the computation is done completely locally, never disrupted by failures or delays.

A final note about these results is that our analysis is based on worst-case *adversarial* input for local sequential algorithms. Thus, it is natural to expect LOCO to achieve a much better approximation ratio in practice, as LOCO randomizes constraint arrival order and so adversarial inputs are extremely unlikely. We verify this intuition in Section 3.6, confirming that our empirical results outperform the theoretical guarantees by a considerable margin. An interesting open problem is to give better theoretical bounds for the local sequential for stochastic inputs. If such results are obtained they would immediately improve the bounds in Theorem 19.

3.5.2 Additional Results and Proofs

Here we outline the analysis used the prove the results presented in the previous section. Core technical parts of the proofs are deferred until Section 3.8.

To begin, in order to bound the communication complexity in Theorem 19, the core argument needed is a bound on the size of sets Y_k. First, we need to define some terminology for hypergraphs. Given a hypergraph $\mathcal{H} = (V, H)$, the *neighbors of a hyperedge* $y \in H$, denoted $\mathcal{N}(y)$, are the hyperedges with vertices in common with y. The *hyperedge degree* of y is its number of neighbors, $|\mathcal{N}(y)|$.

Using this terminology, we proceed to prove some technical lemmas.

Lemma 22. *Let $\mathcal{H} = (V, H)$ be a hypergraph, $|H| = m$, whose hyperedge degree is bounded by d', and let $r : H \to [0, 1]$ be a function that assigns to each hyperedge $y \in H$ a number between 0 and 1 independently and uniformly at random. Let Y_{max} be the size of the largest set of constraints Y_y chosen for a local problem: $Y_{max} = \max\{|Y_y| : y \in H\}$. Then, for $\lambda = 4(d' + 1)$,*

$$\Pr[|Y_{max}| > 2^{\lambda} \cdot 15\lambda \log m] \leq \frac{1}{m^2}.$$

The proof of Lemma 22 uses ideas from a proof in [169], and employs a *quantization* of the rank function. Due to space constraints, the proof of Lemma 22 is deferred until Section 3.8.2.

We are now ready to prove Theorem 19.

Proof of Theorem 19. First, consider the communication complexity. The result in Lemma 22 refers to communication on the hypergraph, $\mathcal{H}$. However, messages will be sent on the physical network, G. Thus we can set $d' = d^2$ in Lemma 22 to describe communication on the physical network.

Lemma 22 establishes the communication required for an individual agent when computing one scalar component of the solution. However, recall that each agent computes the solution to the vector $x_j \in \mathbb{R}^{q_j}$. Taking the union bound over the size of this vector, we see that $\Pr[|Y_k| > 2^{O(d^2)} q_j \log m] \leq \frac{1}{m^2}$.

Due to the sparsity of the constraint matrix, it holds that $|X_k| < d|Y_k|$. Thus, the number of messages is upper bounded by $|X_k|$, and thus, $\Pr[|X_k| > 2^{O(d^2)} q_j \log m] \leq \frac{1}{m^2}$. Finally, the approximation ratio is established by Lemma 18, completing the proof. □

In addition to Lemmas 18 and 22, the following technical lemma is needed to complete the proof of Corollary 20. We restate Theorem 14.1 from [34].

Lemma 23. *For any $B > 0$, there exists a B-competitive online algorithm for linear programs with m constraints; each constraint is violated by a factor at most $\frac{2\log(1+m)}{B}$.*

Proof of Corollary 20. The approximation ratio is due to the online algorithm presented and analyzed in [34] (see Lemma 23). Theorem 19 and Lemma 23, setting $B = 2\log(1 + m)$ imply Corollary 20. □

Proof of Corollary 21. The approximation ratio is due to the online algorithm presented and analyzed in [180]. Theorem 19 implies Corollary 21. □

3.6 Case Studies

The previous section provides worst-case bounds on the performance of LOCO. Here, we illustrate the performance that can be expected in real applications. To do this, we use both synthetic and real data to look at linear programs, NUM, and SVM The results demonstrate an orders-of-magnitude reduction in communication with LOCO as compared to ADMM, while maintaining nearly optimal solutions. We demonstrate the performance of our algorithm on linear programs, a network utility maximization problem, an on training support vector machines. Experiments were run on a server with Intel E5-2623V3@3.0GHz 8 cores and 64GB RAM.

3.6.1 Experimental Setup

Linear Programming. Our first set of experimental results use synthetic linear programming examples. We generate random synthetic instances of linear programs as follows. To generate $A \in \mathbb{R}^{m \times n}$, we set $a_{ij} \sim_{i.i.d.} U(0, 1)$ with probability p and $a_{ij} = 0$ otherwise. We then add $\min\{m, n\}$ i.i.d. draws from $U(0, 1)$ to the main diagonal, to ensure each row of A has at least one nonzero entry. Similarly we set $b_i \sim_{i.i.d.} U[0, 1]$. We set the minimum and maximum transmission rates to be $\underline{x}_i = 0$ and $\bar{x}_i = 1$. Unless otherwise stated, we set $n = m$ and $f_j(x_j) = c_j x_j$ with $c_j \sim_{i.i.d.} U[0, 1]$.

For the case of linear programs in Step 2 of our algorithm, we employ an online algorithm for covering and packing linear programs proposed by [34], pseudocode for which is in Section 3.8.4. Running this algorithm requires tuning one parameter: B, discussed in Lemma 23 in Section 3.5.2, which governs the worst-case guarantee for the online algorithm used in Step 2. A smaller B gives a better guarantee in terms of message complexity, however some constraints may be violated. Setting $B = 2\ln(1 + m)$ provides the best worst-case guarantee, and is our choice in the experiments unless stated otherwise. In fact, it is possible to tune B (akin to tuning ADMM) to specific data, as the constraints are often still satisfied for smaller B. In Figure 34 (c), we show the improvement in performance guarantee by tuning B, while keeping the dual solution feasible.

Throughout all experiments, each point in the figures is averaged over 50 executions, and the ranking function r is a random permutation of the vertex IDs.[1]

Network Utility Maximization (NUM). Our second set of experiments focus on the linear network utility maximization (NUM) problem. We consider the graph of Autonomous System (AS) relationships in [192]. The graph has 8020 nodes and 36406 edges. To interpret the graph in a NUM framework, associate each source node with a path of edges, ending at a destination node. For each source i in the graph, we randomly select a destination which is at distance ℓ_i, sampled *i.i.d.* from Unif$[\ell - 0.5\ell, \ell + 0.5\ell]$. Here $f_j(x_j) = c_j x_j$, which corresponds to throughput maximization. We draw $c \in \mathbb{R}^n$ *i.i.d.* from Unif$[0, 1]$, and set the minimum and maximum transmission rates to be 0 and 1.

Note that Step 2 of LOCO is implemented using the same online algorithm as for linear programming, described above.

Support Vector Machines (SVMs). Our final example is the linear SVM problem, as described in (4.24). We run experiments on both randomly generated synthetic data, and real data. For the synthetic data, we define a matrix $Z \in \mathbb{R}^{m\times n}$ as follows. We set $z_{ij} \sim_{i.i.d.} N(0, 1)$ with probability p and $z_{ij} = 0$ otherwise. We then add $\min\{m, n\}$ i.i.d. draws from $N(0, 1)$ to the main diagonal, to ensure each row of Z has at least one nonzero entry.[2] We set $y_i = +1$ with probability 0.5 and $y_i = -1$ otherwise.

We also run LOCO to train SVMs on the Reuters RCV1 Text Categorization Test Data Set [113], for classification tasks CCAT and C11. This data set has sparsity $p = 0.16\%$, with $n = 47,236$ features, $m = 781,265$ training examples, and $m_{test} = 23149$ testing examples.

When implementing Step 2 of our algorithm for the case of SVM, we employ the well known Pegasos [180] algorithm. Note that in Pegasos, at each step a data point is selected uniformly at random. Our setting is also designed to do this, as the ranking function r is also a collection of values drawn from $[m]$ uniformly at random. However in Pegasos [180], the stopping criterion can

[1]For the purposes of our simulations, such a permutation can be efficiently sampled, and guarantees perfect randomness. For larger n and m, it is possible to use pseudo-randomness with almost no loss in message complexity [169].

[2]Note that the sparsity of A is not necessarily a constant; however, this can only increase the message complexity.

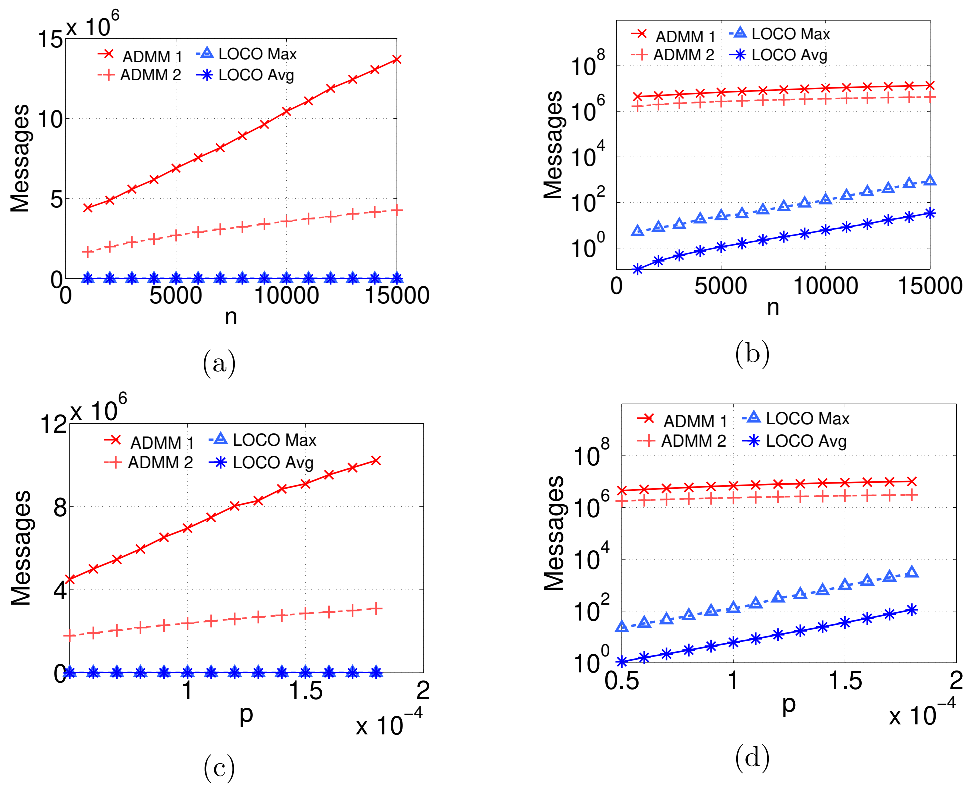


Figure 33: Messages required by LOCO and ADMM for random linear programming instances. Plots (a) and (b) vary n while fixing sparsity $p = 10^{-4}$, showing the results in linear-scale and log-scale respectively. Plots (c) and (d) fix $n = 10^3$ and vary the sparsity p.

be varied along with accuracy requirements, while in our case, we run exactly $|Y_k|$ iterations of Pegasos to solve each local problem. Unless specified, we set the regularization parameter to be $\lambda = 0.0001$.

3.6.2 Benchmark & Performance Metrics

We use ADMM as a benchmark for comparison in this thesis given its prominence in applications. For completeness, the pseudocode for ADMM is included in Section 3.8.5. Running ADMM requires tuning four parameters [31]. Unless otherwise specified, we set the relative and absolute tolerances to be $\epsilon^{rel} = 10^{-4}$ and $\epsilon^{abs} = 10^{-2}$, the penalty parameter to be $\rho = 1$, and the maximum number of allowed iterations to be $t_{max} = 10000$. This is done to provide the best performance for ADMM: the parameters are tuned in the typical fashion to optimize ADMM [31].

We evaluate ADMM and LOCO with respect to the quality of the solution

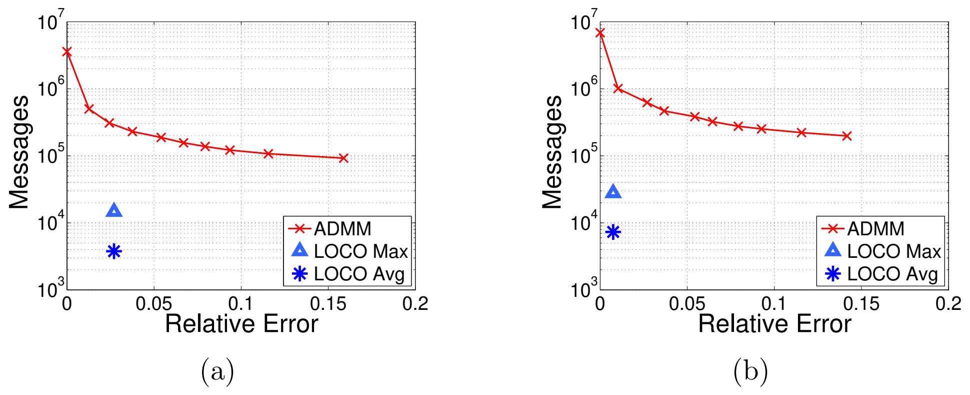


Figure 34: Comparison of the relative error and messages required by LOCO and ADMM for random linear programming instances. Plots (a) and (b) show the Pareto optimal curve for ADMM for two different settings of the relative tolerance parameter: $\epsilon^{rel} = 10^{-4}$ and $\epsilon^{rel} = 10^{-1}$ respectively.

provided and the number of messages sent. To assess the quality of the solution we measure the *relative error*, which is defined as $\frac{|p^* - p^{LOCO}|}{|p^*|}$, where p^* is the optimal solution. For problem instances of small dimension, one can run an interior point method to check the optimal solution, but this is tedious for large problem sizes. In the large dimension cases we consider, we regard p^* to be ADMM's solution with small tolerances, such that the maximum number of allowed iterations is never needed. Note that the relative error is an empirical, normalized version of the approximation ratio for a given instance.

We now explain how we count the number of *messages* used by each of the algorithms. As defined in Section 3.3, a message is a list of matrix coefficients $\{a_{ij} \forall j \in [n] : a_{ij} \neq 0\}$, for a given ith row of the matrix A, along with the coefficient b_i. Since the A matrix is very sparse, this amounts to only sending several matrix coefficients a_{ij} at a time. In contrast, a message in ADMM passes a local copy of the primal and dual solution vectors, which are vectors in $\mathbb{R}^n$ and $\mathbb{R}^m$. Thus, the size of the messages passed by LOCO is smaller than that of ADMM, and any comparison we make between the two is a conservative estimate for the improved communication efficiency of LOCO.

For a distributed implementation of ADMM, two sets of n variables are updated on separate processors (see Chapter 7.1 of [31]). The number of messages required by ADMM is twice the number of nodes in the network G, multiplied by the number of iterations required by ADMM. In contrast, LOCO communicates only in order to construct the local problems; running the online

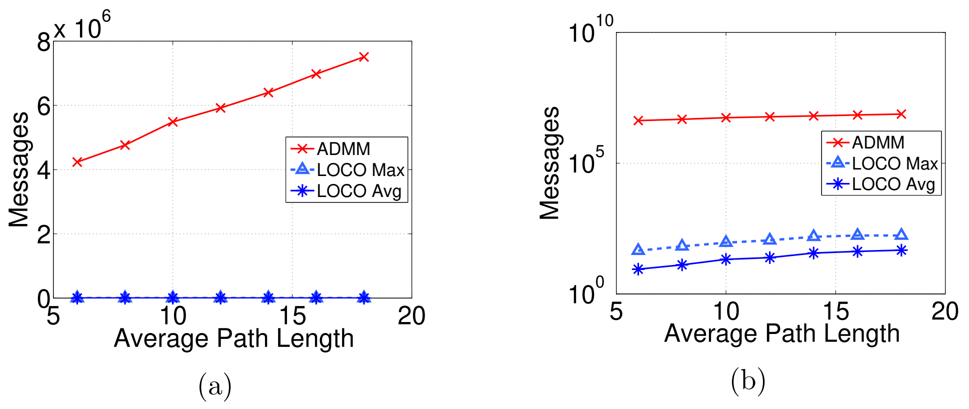


Figure 35: Illustration of the number of messages required by LOCO and ADMM for NUM using an Autonomous System (AS) graph.

algorithm does not require any communication. The number of messages required to construct the kth local problem is proportional to the size of set X_k. When communicating over the hypergraph $\mathcal{H}$, at most $|X_k| + d$ messages are required, and over any general network G, at most $d^2(|X_k| + d)$ messages are required.

Finally, we compare the running times of LOCO and ADMM. We define the *speedup* as the running time of ADMM divided by the running time of LOCO. In all cases, we allow the n nodes to compute in parallel.

3.6.3 Experimental Results

This section describes the results for our case studies. In each case our results highlight order-of-magnitude reductions in communication overhead compared to ADMM with minimal decrease in accuracy. Further, this happens while providing significantly improved robustness.

3.6.3.1 Linear Programming

Our first experimental results focus on synthetic examples of linear programs. Figure 33 illustrates our results, showing that LOCO requires considerably fewer messages than ADMM, across both small and large n and varying levels of sparsity. In these plots, we also chose to plot not only the average messages over all the subproblems, LOCOAvg, but also the maximum amount, LOCO-Max, for the problem with the largest sets X_k and Y_k.

The performance of ADMM depends significantly on the tolerance used, and so the figure includes ADMM with tolerances ϵ^{rel} of both 10^{-4} (ADMM 1) and

10^{-3} (ADMM 2). Note that even with suboptimal tolerance, which results in fewer iterations, ADMM still requires orders of magnitude more communication than LOCO.

We additionally explore the tradeoff between message complexity and relative error. Figures 34(a) and (b) illustrate the Pareto optimal frontier for ADMM: the minimal messages needed in order to obtain a particular relative error. We tune the parameters of ADMM such that the algorithms have comparable relative error to enable a fair comparison. Unlike ADMM, LOCO does not have a comparable parameter to tune, thus LOCO corresponds to a single point in the figures. This point is beyond the Pareto frontier of ADMM, highlighting the order-of-magnitude reduction in communication provided by LOCO. In all the plots, we note that the standard deviations are small enough that they are not visible on the plots.

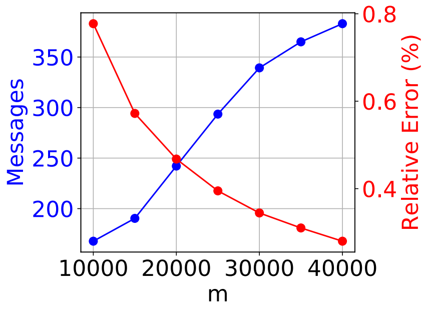
In all of these plots, in some sense ADMM is doing “more” than LOCO. These plots show the communication necessary for an agent to compute its local action. However, when using ADMM the agent is computing the full, global solution, while when using LOCO the agent is computing precisely what is desired: the local action of the agent.

3.6.3.2 Network Utility Maximization (NUM)

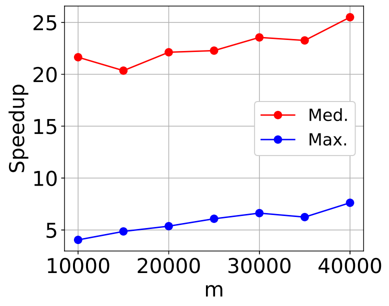
Our second set of results focuses on throughput maximization in NUM. Figure 35 demonstrates and order of magnitude difference in the messages required by LOCO compared to ADMM. The number of messages is shown as a function of the average path length in the instances of the NUM problems. Here, the average path length serves as a metric to describe the sparsity of the constraint matrix, as it has a nonzero component for every utilized edge in the graph. LOCO greatly outperforms ADMM for all tested average path lengths. In all instances, the relative error was 0.4% or less, and so the improvement comes with minimal cost in terms of accuracy. Similar results hold for other objectives beyond throughput maximization, but we omit these due to space constraints.

3.6.3.3 Support Vector Machines (SVMs)

To evaluate LOCO in the context of SVMs, we use both synthetic and real data. Our first set of experiments focus on synthetic data and illustrate the number of messages required and the quality of the solution produced by

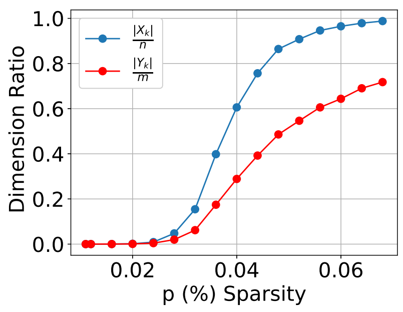


(a) Messages and Relative Error

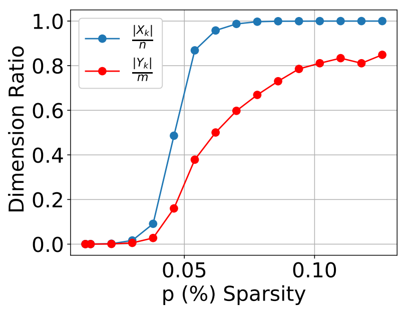


(b) Runtime Ratio

Figure 36: Plot (a) illustrates of the number of messages required by LOCO and the relative error between LOCO and ADMM, in the case of synthetic SVM data, when $n = 10,000$ and m is varied, and $p = 0.03\%$. In all instances, the number of messages required by ADMM was over $100K$, an order of magnitude larger than LOCO, and are not plotted due to being out of range for the plot. Plot (b) illustrates the speedup provided by LOCO compared to ADMM. 'Max' and 'Med' refer to the largest and median sized subproblem respectively.



(a)



(b)

Figure 37: Comparison of the local problem dimension ($|X_k| \times |Y_k|$) to the original problem size ($m \times n$), averaged over all the local problems, in the case of synthetic SVM data. In (a) $n = m = 10,000$, in (b) $n = 5,000$ and $m = 10,000$.

Table 31: SVM on CCAT and C11 from Reuters RCV1

Alg.	**CCAT Train (Test)**	**C11 Train (Test)**	**Mess-ages**
ADMM	0.31% (10.74%)	0.14% (3.09%)	330K
LOCO	4.84% (7.88%)	2.64% (3.01%)	18K

LOCO compared to ADMM. Figure 36(a) shows both the number of messages for LOCO and relative error between the LOCO and ADMM as m varies. The messages are averaged over the local problems. The messages required by for ADMM are all above $100K$, and out of range for this plot. In general ADMM requires an order of $O(nT)$ messages, where T is the number of iterations. We also see that as the problem sizes increases, the relative error between the LOCO and ADMM decreases. Figure 36(b) shows the speedup provided by LOCO compared to ADMM. As the problem size m increases, the speedup increases.

Next, we evaluate the performance of LOCO on real data using the Reuters RCV1 Text Categorization Test Data Set [113]. Results are found in Table 41.

We found that for task CCAT LOCO produces a test error of 6.16% when run on the original dataset. However, when generating sets Y_k, we found that the dataset could be thresholded to increase sparsity and reduce communication overhead further. To do this, we thresholded the values in the matrix below 0.1, setting all such values equal to 0. The thresholding value was chosen so that the test error did not change significantly, while the sparsity of the resulting matrix decreased. We tried several different thresholds, and found 0.1 to be representative for this dataset. We note that thresholding is a valuable tool in practice only when it does not increase the test error significantly. This resulted in a matrix with sparsity $p = 0.045\%$. Running LOCO on this thresholded data set led to a slight increase in the test error; 6.16% originally, and 7.88% with thresholding. However, the local problems reduced to an average size of $|Y_k| = 18\text{K}$, which is a order of magnitude reduction of the original problem dimension of $m = 781,265$, and consequently yields an order-of-magnitude reduction in communication.

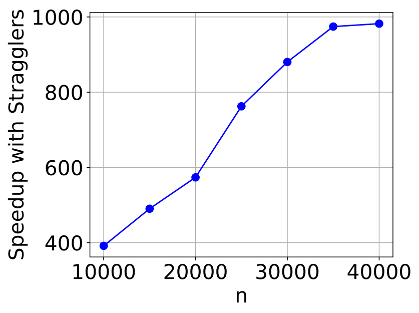
3.6.3.4 Sparsity

The performance of LOCO is dependent on the sparsity of the constraint matrix A. As seen in Figure 33 (c) and (d), the number of messages increases as p increases. Many real world problems, such as NUM and SVM discussed above, involve very sparse matrices which are appropriate for LOCO. We further investigate the effects of sparsity in Figure 37. The figure highlights that the improvement in communication achieved by LOCO is possible because the dimensions of the local problems, $(|X_k| \times |Y_k|)$, are small compared to the original $(m \times n)$ problem. The dimensions of the local problems are dependent on the sparsity, p, due to the way in which we use the sparsity structure to determine when new constraints are added to set Y_k in Algorithm 3. In Figure 37, as p increases, the local problems get larger, and plateau when $|Y_k| = m$. For the synthetic SVM data described in Section 3.6.1, we empirically observe a phase transition like behavior as p varies. We note that this transition depends on the distribution of the placement of the nonzero elements in matrix A. For example in Figure 37(a) and (b), $|Y_k|$ varies differently for different n and m.

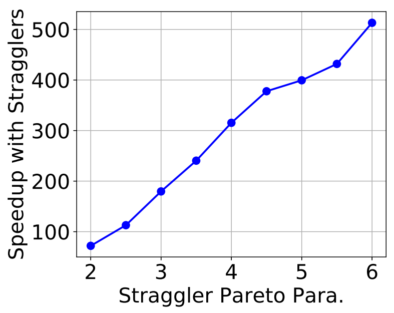
3.6.3.5 Stragglers & Failures

Our last results highlight the robustness of LOCO to stragglers and failures. In modern distributed systems, stragglers are a fact of life. Conflicts and congestion lead to unpredictable delays in local parts of the system, which can then delay the progression of distributed algorithms globally. Figure 38 illustrates the robustness of LOCO to stragglers by plotting the speedup of LOCO as compared to ADMM. In these experiments, we model the distribution of delays caused by stragglers using a Pareto distribution, which is motivated by empirical studies of stragglers in real systems such as [8]. The figure highlights that, as n increases, the speedup provided by LOCO is more pronounced and that as the tail of the distribution of stragglers becomes heavier the difference becomes less pronounced.

We also consider the effect of node failures on LOCO. In LOCO a failure at node j affects all the nodes that share common nodes found in set X_j. We experimented with a variety of settings, and the comparison between ADMM and LOCO is dramatic. A representative example is with $n = m = 10,000$ and $p = 0.03$. The results from the other settings we considered are qualitatively the same. In this setting, the largest set X_j in LOCO has about 5% of all the nodes. As a result, the failure of a single node has the capacity, in the worst



(a)



(b)

Figure 38: Illustration of the impact of stragglers. The plots illustrate how the speedup of LOCO relative to ADMM varies with (a) the problem dimension n, when the Pareto shape parameter is set to 5, and (b) the shape parameter of the straggler distribution. Synthetic SVM data is used with $n = m = 10,000$ and $p = 0.03\%$.

case, to affect about 15% of the nodes. In contrast, in ADMM, a single failure stops the whole process as the central node waits for the failed node, and thus no nodes obtain solutions.

3.7 Discussion

We introduced a new approach for the design of multi-agent systems using distributed optimization based on ideas from the emerging field of local computation algorithms. In our framework, LOCO, each agent in a network computes its local piece of the solution, using exponentially less communication than existing techniques, and produces a provably nearly optimal solution without the need for iterative rounds of communication. Additionally, LOCO is robust to network stragglers and failures due to the independent nature of the local problems. Our empirical case studies demonstrate that LOCO requires orders of magnitude fewer messages than ADMM, while maintaining high quality solutions in random linear programming instances, and NUM and SVM problems.

We remark that the reduction in this work holds for worst-case guarantees of sequential algorithms, when the constraints arrive in adversarial order. However, in LOCO we determine the order of arrival internally, and so the worst-case guarantees may be too conservative. Our reduction also holds for average-case guarantees of sequential algorithms, when the constraints arrive uniformly at random. Hence, any such guarantees immediately apply to LOCO. Currently, there are few such theoretical guarantees to problems to which LOCO is applicable, and we believe this is an interesting research direction. Further, it may be possible to improve performance by optimizing the order in which constraints arrive or by choosing the form of randomness used in the order of arrivals in order to avoid the adversarial behavior underlying the worst-case bounds in this work.

We view this work as a first step towards the investigation of local computation algorithms for distributed optimization. In future work, we plan to study the performance of LOCO on more general network optimization problems. Further, it would be interesting to apply other techniques from the field of local computation algorithms to develop algorithms for other settings in which distributed computing is useful, such as power systems and federated machine learning.

3.8 Proofs of Technical Results

3.8.1 Proof of Lemma 18

By the definition of local sequential algorithms, the last time a variable x_j can be updated is the last arrival time of a constraint y_i such that $a_{ij} \neq 0$ and its new values depend *only* on the value (at step i) of the primal variables x_j such that $a_{ij} \neq 0$. Assume that LOCO simulates $\mathcal{A}$, a local sequential algorithm. It suffices to show that when y_i "arrives" during the execution of LOCO, the primal variables x_j such that $a_{ij} \neq 0$ have the same value as they do when y_i arrives during the execution of $\mathcal{A}$. We show this by contradiction.

Denote the constraints in Y_j by $y_1, \ldots, y_{|Y_j|}$, and assume that they are sorted by arrival time, i.e., y_1 arrives first out of the constraints in Y_j. Let i' be the smallest value such that there exists some x_j for which $a_{i'j} \neq 0$ that has a different value in the two executions when $y_{i'}$ arrives. If x_j was never updated in the execution, this is because there exists no constraint $y_{i''}$ that is a neighbor of $y_{i'}$ in H that arrived before $y_{i'}$, hence x_j was never updated in the execution of $\mathcal{A}$ it was not updated by $\mathcal{A}$ before $y_{i'}$ arrived. Otherwise, consider the last time x_j was updated by LOCO. Assume this was when $y_{i''}$ arrived. As i' is the smallest value such that there exists some x_j for which $a_{i'j} \neq 0$ that has a different value in the two executions when $y_{i'}$ arrives, it must hold that all of the variables x_j such that $a_{i''j} \neq 0$ were correctly valued, but then by the definition of the local sequential algorithm, $x_{j'}$ must have been updated correctly, a contradiction.

3.8.2 Proof of Lemma 22

Logarithms are base e. Let $\mathcal{H} = (V, H)$ be a hypergraph. Recall that the *neighbors of a hyperedge* $y \in H$ are the hyperedges with vertices in common with y, denoted $\mathcal{N}(y)$. For any set of hyperedges $S \subseteq H$, let $\mathcal{N}(S)$ denote the set of hyperedges that are not in S but are neighbors of some hyperedge in S: $\mathcal{N}(S) = \{\mathcal{N}(y) : y \in S\} \setminus S$. For a set $S \subseteq H$ and a function $g : H \to \mathbb{N}$, we use $S \cap g^{-1}(i)$ to denote the set $\{y \in S : g(y) = i\}$.

Let $\mathcal{H} = (V, H)$ be a hypergraph, and let $g : H \to \mathbb{N}$ be some function on the hyperedges. An *adaptive hyperedge exposure procedure* is one that does not know g a priori. The procedure is given an edge $y \in H$ and $g(y)$. Edges from $H \setminus S$ are iteratively added to S; for every edge y' added, $g(y')$ is revealed immediately after y' is added. Let S_t denote S after the addition of the t-th edge. The following is a concentration bound that shows that for a random g, any sufficiently large set of adaptively exposed hyperedges, less than half will have the same value of g w.h.p. Its short proof is given for completeness.

Lemma 24. *Let $\mathcal{H} = (V, H)$ be a hypergraph for which $|H| = m$, let $Q > 0$ be some constant, let $\gamma = 15Q$, and let $g : H \to [Q]$ be a function chosen uniformly at random from all such possible functions. Consider an adaptive hyperedge exposure procedure that is initialized with an edge $y \in H$. Then, for any $q \in [Q]$, the probability that there is some t, $\gamma \log m \leq t \leq m$ for which $|S_t \cap g^{-1}(q)| > \frac{2|S_t|}{Q}$ is at most $\frac{1}{m^4}$.*

Proof. Let y_i be the ith edge added to S by the adaptive hyperedge exposure procedure, and let I_i be the indicator variable whose value is 1 iff $g(y_i) = q$. For any $t \leq m$, $\mathbb{E}\left[[]\sum_{i=1}^{t} I_j\right] = \dfrac{t}{Q}$. As I_i and I_j are independent for all $i \neq j$, by the Chernoff bound, for $\gamma \log m \leq t \leq m$,

$$\Pr\left[\sum_{i=1}^{t} I_j > \frac{2t}{Q}\right] \leq e^{\frac{-t}{3Q}} \leq e^{-5\log m}.$$

A union bound over all possible values of $t : \gamma \log m \leq t \leq m$ completes the proof. $\square$

Recall that d' is the upper bound on the hyperedge degree. Let $r : V \to [0, 1]$ be a function chosen uniformly at random from all such possible functions.

Partition $[0, 1]$ into $Q = 4(d' + 1)$ segments of equal measure, $W_1, \ldots, W_Q$. For every $v \in V$, set $g(v) = q$ if $r(v) \in W_q$ (g is a quantization of r).

Consider the following method of generating two sets of vertices: Y and Z, where $Y \subseteq Z$. Set Z can be thought of as a set S_t for some t as described in Lemma 24. For some edge h, set $Y = Z = \{h\}$. Continue inductively: choose some edge $w \in Y$, add all $\mathcal{N}(w)$ to Z and compute $g(u)$ for all $u \in \mathcal{N}(w)$. Add the edges u such that $u \in \mathcal{N}(w)$ and $g(u) \geq g(w)$ to Y. The process ends when no more edges can be added to Y.

Y is generated with respect to g, the quantization of r. The actual sets of constraints constructed in LOCO for the local problems are defined with respect to r. Here, $|Y|$ is an upper bound on the size of the sets constructed in LOCO. It is difficult to reason about the size of Y directly, as the ranks of its edges are not independent. The edges of the vertices in Z, though, *are* independent, as Z is generated by an adaptive hyperedge exposure procedure. Z is a superset of Y that includes Y and its boundary, hence $|Z|$ is also an upper bound on the size of the query set.

We now define $Q + 1$ "layers" - $Y_{\leq 0}, \ldots, Y_{\leq Q}$: $Y_{\leq q} = Y \cap \bigcup_{i=0}^{q} g^{-1}(i)$. That is, $Y_{\leq q}$ is the set of vertices in Y whose rank is at most q. (The range of g is $[Q]$, hence $Y_{\leq 0}$ will be empty, but we include it to simplify the proof.)

Claim 25. *Set $Q = 4(d' + 1)$, $\gamma = 15Q$. Assume without loss of generality that $g(v) = 0$. Then for all $0 \leq i \leq Q - 1$,*

$$\Pr[|Y_{\leq i}| \leq 2^i \gamma \log m \wedge |Y_{\leq i+1}| \geq 2^{i+1} \gamma \log m] \leq \frac{1}{m^4}.$$

Proof. For all $0 \leq i \leq Q$, let $Z_{\leq i} = Y_{\leq i} \cup N(Y_{\leq i})$. Note that

$$Z_{\leq i} \cap g^{-1}(i) = Y_{\leq i} \cap g^{-1}(i), \tag{3.6}$$

because if there had been some $u \in N(Y_{\leq i}), g(u) = i$, u would have been added to $Y_{\leq i}$.

Note that $|Y_{\leq i}| \leq 2^i \gamma \log m \wedge |Y_{\leq i+1}| \geq 2^{i+1} \gamma \log m$ implies that

$$|Y_{\leq i+1} \cap g^{-1}(i + 1)| > \frac{|Y_{\leq i+1}|}{2}. \tag{3.7}$$

In other words, the majority of vertices $v \in Y_{\leq i+1}$ must have $g(v) = i + 1$.

Given $|Y_{\leq i+1}| > 2^{i+1}\gamma \log m$, it holds that $|Z_{\leq i+1}| > 2^{i+1}\gamma \log m$ because $Y_{\leq i+1} \subseteq Z_{\leq i+1}$. Furthermore, $Z_{\leq i+1}$ was constructed by an adaptive hyperedge exposure procedure and so the conditions of Lemma 24 hold for $Z_{\leq i+1}$. From Equations (3.6) and (3.7) we get

$$\begin{aligned}
&\Pr[|Y_{\leq i}| \leq 2^{i}\gamma \log m \wedge |Y_{\leq i+1}| \geq 2^{i+1}\gamma \log m] \\
&\leq \Pr\left[\left|Z_{\leq i+1} \cap g^{-1}(i+1)\right| > \frac{|Y_{\leq i+1}|}{2}\right] \\
&\leq \Pr\left[\left|Z_{\leq i+1} \cap g^{-1}(i+1)\right| > \frac{2\,|Z_{\leq i+1}|}{Q}\right] \\
&\leq \frac{1}{m^4},
\end{aligned}$$

where the second inequality is because $|Z_{\leq i+1}| \leq (d+1)|Y_{\leq i+1}|$, as G's degree is at most d'; the last inequality is due to Lemma 24. $\square$

Lemma 26. *Set* $Q = 4(d'+1)$. *Let* $\mathcal{H} = (V, H)$ *be a hypergraph with degree bounded by* d', *where* $|H| = m$. *For any edge* $h \in H$, $\Pr\left[Y_h > 2^Q \cdot 15Q \log m\right] < \frac{1}{m^3}$.

Proof. To prove Lemma 26, we need to show that, for $\gamma = 15Q$,

$$\Pr[|Y_{\leq Q}| > 2^L \gamma \log m] < \frac{1}{m^3}.$$

We show that for $0 \leq i \leq Q, \Pr[|Y_{\leq i}| > 2^i \gamma \log m] < \frac{i}{m^4}$, by induction. For the base of the induction, $|S_0| = 1$, and the claim holds. For the inductive step, assume that $\Pr[|Y_{\leq i}| > 2^i\gamma \log m] < \frac{i}{m^4}$. Then, denoting by $\mathbb{I}$ the event $|Y_{\leq i}| > 2^i \gamma \log m$ and by $\bar{\mathbb{I}}$ the event $|Y_{\leq i}| \leq 2^i\gamma \log m$,

$$\begin{aligned}
&\Pr[|Y_{\leq i+1}| > 2^{i+1}\gamma \log m] \\
&= \Pr[|Y_{\leq i+1}| > 2^{i+1}\gamma \log m : \mathbb{I}] \Pr[\mathbb{I}] \\
&+ \Pr[|Y_{\leq i+1}| > 2^{i+1}\gamma \log m : \bar{\mathbb{I}}] \Pr[\bar{\mathbb{I}}].
\end{aligned}$$

From the inductive step and Claim 25, using the union bound, the lemma follows. $\square$

Applying a union bound over all the hyperedges gives that the size of set Y_k pertaining to the kth local problem (3.5) is $O(\log m)$ with probability at least $1 - 1/m^2$, completing the proof of Lemma 22.

Algorithm 4: General Online Fractional Linear Packing

```
Input: Linear Program defined on $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$, and
approximation parameter $B$
Output: $x$, $y$
Initialize: $x = \mathbf{0_n}$, $y = \mathbf{0_m}$

for $i = 1...m$ do
  for $j = 1...n$ do
    $a^j(\max) \leftarrow \max_{k=1}^{i}\{a_{kj}\}$;

  while $\sum_{j=1}^{n} a_{ij}x_j < 1$ do
    Increase $y_i$ continuously;
    for $j = 1...n$ do
      $\delta = \exp(\frac{B}{2c_i}\sum_{k=1}^{i} a_{kj}y_k) - 1$ ;
      $x_j = \max\left\{x_j, \frac{1}{na^j(\max)}\delta\right\}$
```

3.8.3 Supplementary information

A note on ranking the constraints. In the Introduction, we describe generating a random permutation over the constraints. However, storing a random permutation requires $\Omega(n)$ space, and we would need to store this permutation on every node. Instead, we can approximate a random permutation with a random ordering by assigning a real number uniformly at random to each constraint, using function r as we described in Section 3.4. We note that in practice, such an r does not exist. It has been shown in e.g., [7, 169] that r can be approximated arbitrarily well by a hash function by a random hash function of polylogarithmic length. We do not formally define what we mean by "arbitrarily well" here; we refer the reader to [169] for an in depth discussion. In this thesis we assume for simplicity that each node has access to an r function.

3.8.4 Pseudocode for General Online Fractional Packing

In our experiments in Section 3.6 we use an online algorithm from [34] for the cases of linear programming and NUM. For completeness we give the details of the algorithm in Algorithm 4.

In this online problem, constraints arrive in an online fashion over a sequence of rounds. During the ith round, the packing variable y_i and the covering variables x_j $\forall j \in [n]$ for which $a_{ij} > 0$ are increased. The minimum y_i is found such that the covering constraints are satisfied.

3.8.5 ADMM

In our numerical results we compare LOCO to ADMM. For completeness, we describe the application of ADMM to problem (2.1).

To apply ADMM, we introduce a slack variable $s \geq 0$ such that the inequality constraint becomes $Ax - s = b$. Let $x' = \begin{bmatrix} x, s \end{bmatrix}^T$, $A' = [A \quad -I]$ and $b = \begin{bmatrix} \mathbf{1}_n, \mathbf{0}_n \end{bmatrix}^T$ where this notation indicates a stack of vectors. We can now write the problem in standard ADMM form,

$$\begin{aligned} \min_{x',z} \quad & g(x') + h(z) \\ \text{s.t.} \quad & x' - z = 0, \end{aligned}$$

where $g = (x)_+$ is the indicator function associated with the constraints $x \geq 0$ and $h(z') = -\sum_{j=1}^{n} f_j(z_j)$ where dom $h = \{z | A'z = b'\}$.

Writing down the scaled augmented Lagrangian $L_\rho(x', z, u) = g(x') + h(z) + u^T(z - x') + \frac{\rho}{2}\|x' - z\|^2$, we can see that all the update steps have closed form solution (see Chapter 5.2 of [31]). The updates become

$$\begin{aligned} x'^{k+1} &= (z^{k+1} + u^k)_+ \\ z^{k+1} &= \begin{bmatrix} \rho I & A'^T \\ A' & 0 \end{bmatrix}^{-1} \begin{bmatrix} \rho(x'^k - u^k) - b \\ c' \end{bmatrix} \\ u^{k+1} &= u^k + (x'^{k+1} - z^{k+1}). \end{aligned}$$

The solution to problem (2.1) is recovered from the first n entries of x'.

Chapter 4

SPEEDING UP IPMS FOR LINEAR PROGRAMMING USING RANDOMIZATION

Internally, commercial linear program solvers use both Interior Point Methods (IPMs) and the simplex method. Thus designing more efficient IPMs is of practical importance. Especially as datasets get larger, IPMs suffer from scaling issues. We present a provably accurate long-step infeasible IPM for LPs. The approach uses randomization, while still producing an optimal solution. First, we provide background on IPMs, and discuss the computation bottlenecks involved.

4.1 Background on Interior Point Methods

Interior Point Methods (IPMs) were pioneered by Karmarkar in the mid 1980s [97]. A family of IPMs has grown since then. For an introduction to IPMs, see Chapter 14 of Wright's book, [149] and [202]. Consider the standard form of the primal LP problem:

$$\min c^{\mathsf{T}}x\,,\ \text{subject to } Ax = b\,, x \geq \mathbf{0}\,, \tag{4.1}$$

where $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, and $c \in \mathbb{R}^n$ are the inputs, and $x \in \mathbb{R}^n$ is the vector of the primal variables. The associated dual problem is

$$\max b^{\mathsf{T}}y\,,\ \text{subject to } A^{\mathsf{T}}y + s = c\,, s \geq \mathbf{0}\,, \tag{4.2}$$

where $y \in \mathbb{R}^m$ and $s \in \mathbb{R}^n$ are the vectors of the dual and slack variables respectively. Triplets (x, y, s) that uphold both (4.1) and (4.2) are called *primal-dual solutions.*

IPMs work by searching the interior of the feasible region in order to get close to an optimal point. IPM algorithms start at an initial point, either inside or outside of the feasible region, and at each iteration, compute a new Newton search direction $(\Delta x, \Delta y, \Delta s)$ and step length. Thus, a path in the feasible region is iteratively traversed until the algorithm identifies an almost optimal solution close to the boundary of the feasible region. To guide the choice of new search direction, the algorithm strives to satisfy (or nearly satisfy) the optimality conditions of the optimization problem. The solutions (x, y, s)

of (4.1) and (4.2) are characterized by the well-known Karush-Kuhn-Tucker (KKT) conditions. We propose to focus on *Infeasible* IPMs, where the goal is to satisfy approximate KKT conditions, by solving the following linear system:

$$Ax - b = r_p \quad A^\mathsf{T} y + s - c = r_d\,, \quad XS\,\mathbf{1}_n = \sigma\mu\mathbf{1}_n\,, \quad (x, s) > \mathbf{0}\,, \tag{4.3a}$$

where r_p and r_d are the residuals of the primal and dual feasibility constraints respectively, and σ and μ are parameters of the algorithm. In order to obtain a search direction $(\Delta x, \Delta y, \Delta s)$, one standard approach [149] involves solving the following linear system at each iteration of the IPM:

$$AD^2A^\mathsf{T}\Delta y = p, \tag{4.4a}$$

$$\Delta s = -r_d - A^\mathsf{T}\Delta y\,, \tag{4.4b}$$

$$\Delta x = -x + \sigma\mu S^{-1}\mathbf{1}_n - D^2\Delta s\,, \tag{4.4c}$$

where $D = X^{\frac{1}{2}}S^{-\frac{1}{2}}$, and $X, S \in \mathbb{R}^{n\times n}$ are diagonal matrices having i-th diagonal elements equal to i-th components of x and s respectively for $i = 1, 2, \ldots, n$, and $p \in \mathbb{R}^m$.

Solving the linear system (4.4a), which is often referred as the *normal equations*[1], is the core computational task. Given Δy, computing Δs and Δx involves matrix-vector products.

4.1.1 Computational Bottleneck for IPMs

The core computational bottleneck in IPMs is the need to solve the linear system of eqn. (4.4a) at each iteration. This leads to two key challenges: first, for high-dimensional matrices A, solving the linear system is computationally prohibitive. Most implementations of IPMs use a *direct solver*; see Chapter 6 of [149]. However, if AD^2A^T is large and dense, direct solvers are computationally impractical. If AD^2A^T is sparse, specialized direct solvers have been developed, but these do not apply to many LP problems arising in machine learning applications due to irregular sparsity patterns. Second, an alternative to direct solvers is the use of iterative solvers, but the situation is further complicated since AD^2A^T is typically ill-conditioned. Indeed, as IPM algorithms approach the optimal primal-dual solution, the diagonal matrix D is

[1]Another widely used approach is to solve the augmented system [149]. This approach is less relevant for this thesis.

ill-conditioned, which also results in the matrix AD^2A^T being ill-conditioned. Additionally, using approximate solutions for the linear system of eqn. (4.4a) causes certain invariants, which are crucial for guaranteeing the convergence of IPMs, to be violated; see Section 4.2.1 for details.

We propose to address both challenges by sketching the matrix AD in order to solve a smaller linear system at each iteration. We will use iterative linear solvers (e.g., Richardson's iteration, Conjugate Gradients, etc.) to solve the sketched version of (4.4a). It is well-known that the matrix AD^2A^T becomes ill-conditioned as we approach optimality: our sketching strategy will take this into account in order to make the condition number of the pre-conditioned system *provably* small. This is key for developing an algorithm that works well in practice.

Specifically, we construct a sketching matrix $W \in \mathbb{R}^{n\times w}$, for an appropriate choice of the sketching dimension $w \ll n$ and an accuracy parameter ε, such that the structural constraint $\left\|V^\mathsf{T}WW^\mathsf{T}V - I_m\right\|_2 \leq \varepsilon$ is satisfied, where we let $V \in \mathbb{R}^{n\times m}$ be the matrix of right singular vectors of AD. We define $Q = ADWW^\mathsf{T}DA^\mathsf{T}$. If the structural constraint is satisfied for each iteration of the interior point method, we show that equation (4.4a) can be solved using Richardson iteration with a randomized pre-conditioner $Q^{-\frac{1}{2}}$. Thus, rather than solving (4.4a), we solve the preconditioned system

$$Q^{-\frac{1}{2}}\, AD^2A^\mathsf{T}\Delta y = \; Q^{-\frac{1}{2}}p\,, \tag{4.5}$$

and recover an approximate solution $\hat{\Delta y}$, which we will then use to approximate solutions $\hat{\Delta x}$ and $\hat{\Delta s}$. Let $\tilde{f} = Q^{-\frac{1}{2}}(AD^2A^\mathsf{T}\hat{\Delta y} - p)$ be the the residual of the preconditioned system. In general, our goal is to show that $\|\tilde{f}\|_2$ is small, and ensure that despite generating approximate solutions at each iteration, we produce an accuracy guarantee for the IPM after a given number of iterations.

4.2 Setting of interest: under/over constrained LPs

In this work, we address the aforementioned challenges for the special case where $m \ll n$, i.e., the number of constraints is much smaller than the number of variables; see Section 4.6 for a generalization. This is a common setting in ML applications of LP solvers, since ℓ_1-SVMs and basis pursuit problems often exhibit such structure when the number of available features (n) is larger than the number of objects (m). This setting has been of interest in recent work on LPs [64, 26, 117]. For simplicity of exposition, we also assume that the constraint matrix A has full rank, equal to m.

First, we propose and analyze a preconditioned Conjugate Gradient (CG) iterative solver for the normal equations of eqn. (4.4a), using matrix sketching constructions from the Randomized Linear Algebra (RLA) literature. We develop a preconditioner for AD^2A^T using matrix sketching which allows us to prove strong convergence guarantees for the *residual error* of CG solvers.

Second, building upon the work of [137], we propose and analyze a provably accurate long-step *infeasible* IPM algorithm. The proposed IPM solves the normal equations using iterative solvers. In this thesis, for brevity and clarity, we primarily focus our description and analysis on the CG iterative solver. We note that a non-trivial concern is that the use of iterative solvers and matrix sketching tools implies that the normal equations at each iteration will be solved only approximately. In our proposed IPM, we develop a novel way to *correct* for the error induced by the approximate solution in order to guarantee convergence. Importantly, this correction step is relatively computationally light, unlike a similar step proposed in [137]. Third, we empirically show that our algorithm performs well in practice. We consider solving LPs that arise from ℓ_1-regularized SVMs and test them on a variety of synthetic and real datasets. Several extensions of our work are discussed in Section 4.6.

4.2.1 Our contributions

Our point of departure in this work is the introduction of preconditioned, iterative solvers for solving eqn. (4.4a). Preconditioning is used to address the ill-conditioning of the matrix AD^2A^T. Iterative solvers allow the computation of approximate solutions using only matrix-vector products while avoiding matrix inversion, Cholesky or LU factorizations, etc. A preconditioned formulation of eqn. (4.4a) is

$$Q^{-1}AD^2A^\mathsf{T}\Delta y = Q^{-1}p, \tag{4.6}$$

where $Q \in \mathbb{R}^{m\times m}$ is the preconditioning matrix; Q should be easily invertible (see [13, 84] for background). An alternative yet equivalent formulation of eqn. (4.6), which is more amenable to theoretical analysis, is

$$Q^{-1/2}AD^2A^\mathsf{T}Q^{-1/2}z = \; Q^{-1/2}p, \tag{4.7}$$

where $z \in \mathbb{R}^m$ is a vector such that $\Delta y = Q^{-1/2} z$. Note that the matrix in the left-hand side of the above equation is always symmetric, which is not necessarily the case for eqn. (4.6). We do emphasize that one can use eqn. (4.6) in the actual implementation of the preconditioned solver; eqn. (4.7) is much more useful in theoretical analyses.

Recall that we focus on the special case where $A \in \mathbb{R}^{m \times n}$ has $m \ll n$, i.e., it is a short-and-fat matrix. Our first contribution starts with the design and analysis of a preconditioner for the Conjugate Gradient solver that satisfies, with high probability,

$$\frac{2}{2+\zeta} \leq \sigma^2_{\min}(Q^{-\frac{1}{2}} AD) \leq \sigma^2_{\max}(Q^{-\frac{1}{2}} AD) \leq \frac{2}{2-\zeta}, \tag{4.8}$$

for some error parameter $\zeta \in [0, 1]$. In the above, $\sigma_{\min}(\cdot)$ and $\sigma_{\max}(\cdot)$ correspond to the smallest and largest singular value of the matrix in parentheses. The above condition says that the preconditioner effectively reduces the condition number of AD to a constant. We note that the particular form of the lower and upper bounds in eqn. (4.8) was chosen to simplify our derivations. RLA matrix-sketching techniques allow us to construct preconditioners for all short-and-fat matrices that satisfy the above inequality *and* can be inverted efficiently. Such constructions go back to the work of [11]; see Section 4.4 for details on the construction of Q and its inverse. Importantly, given such a preconditioner, we then prove that the resulting CG iterative solver satisfies

$$\|Q^{-1/2} AD^2 A^\mathsf{T} Q^{-1/2} \tilde{z}^t - Q^{-1/2} p\|_2 \leq \zeta^t \|Q^{-1/2} p\|_2. \tag{4.9}$$

Here $\tilde{z}^t$ is the approximate solution returned by the CG iterative solver after t iterations. In words, the above inequality states that the *residual error* achieved after t iterations of the CG iterative solver drops exponentially fast. To the best of our knowledge, this result is not known in the CG literature: indeed, it is actually well-known that the residual error of CG may oscillate, even in cases where the energy norm of the solution error decreases monotonically. However, we prove that if the preconditioner is sufficiently good, i.e., it satisfies the constraint of eqn. (4.8), then the residual error decreases as well.

Our second contribution is the analysis of a novel variant of a long-step *infeasible* IPM algorithm proposed by [137]. Recall that such algorithms can, in general, start with an initial point that is not necessarily feasible, but does need to satisfy some, more relaxed, constraints. Following the lines of [211, 137], let $\mathcal{S}$ be the set of feasible and optimal solutions of the form (x^*, y^*, s^*) for the primal and dual problems of eqns. (4.1) and (4.2) and assume that $\mathcal{S}$ is not empty. Then, long-step infeasible IPMs can start with any initial point (x^0, y^0, s^0) that satisfies $(x^0, s^0) > 0$ *and* $(x^0, s^0) \geq (x^*, s^*)$, for some feasible and optimal solution $(x^*, s^*) \in \mathcal{S}$. In words, the starting primal and slack variables must be strictly positive *and* larger (element-wise) when compared to some feasible, optimal primal-dual solution. See Chapter 6 of [202] for an extensive discussion regarding why such starting points that are *sufficiently positive* can be identified in practice more efficiently than feasible points.

The flexibility of infeasible IPMs comes at a cost: long-step *feasible* IPMs converge in $\mathcal{O}(n \log 1/\epsilon)$ iterations, while long-step *infeasible* IPMs need $\mathcal{O}(n^2 \log 1/\epsilon)$ iterations to converge [211, 137]. (Here ϵ is the accuracy of the approximate LP solution returned by the IPM; see Algorithm 6 for the exact definition.) Let

$$Ax^0 - b = r_p^0, \tag{4.10}$$

$$A^\mathsf{T} y^0 + s^0 - c = r_d^0, \tag{4.11}$$

where $r_p^0 \in \mathbb{R}^n$ and $r_d^0 \in \mathbb{R}^m$ are the *primal* and *dual* residuals, respectively, and characterize how far the initial point is from being feasible. As long-step infeasible IPM algorithms iterate and update the primal and dual solutions, the residuals are updated as well. Let $r^k = (r_p^k, r_d^k) \in \mathbb{R}^{n+m}$ be the primal and dual residual at the k-th iteration: it is well-known that the convergence analysis of infeasible long-step IPMs critically depends on r^k lying on the line segment between 0 and r^0. Unfortunately, using approximate solvers (such as the CG solver proposed above) for the normal equations violates this invariant. [137] proposed a simple solution to fix this problem by adding a perturbation vector v to the current primal-dual solution that guarantees that the invariant is satisfied. Again, we use RLA matrix sketching principles to propose an efficient construction for v that provably satisfies the invariant. Next, we combine the above two primitives to prove that Algorithm 6 in Section 4.5 satisfies the following theorem.

Theorem 27. *Let $0 \leq \epsilon \leq 1$ be an accuracy parameter. Consider the long-step infeasible IPM Algorithm 6 (Section 4.5) that solves eqn. (4.7) using the CG solver of Algorithm 5 (Section 4.4). Assume that the CG iterative solver runs with accuracy parameter $\zeta = 1/2$ and iteration count $t = \mathcal{O}(\log n)$.*

Then, with probability at least 0.9, the long-step infeasible IPM converges after $\mathcal{O}(n^2 \log {}^1/_\epsilon)$ *iterations.*

We note that the 0.9 success probability above is for simplicity of exposition and can be easily amplified using standard techniques. Also, at each iteration of our infeasible long-step IPM algorithm, the running time is $\mathcal{O}((\mathsf{nnz}(A) + m^3)\log n)$, ignoring constant terms. See Section 4.5 for a detailed discussion of the overall running time.

Our empirical evaluation demonstrates that our algorithm requires an order of magnitude fewer inner CG iterations than a standard IPM using CG, while producing a comparably accurate solution (see Section 4.7).

4.2.2 Comparison with Related Work

There is a large body of literature on solving LPs using IPMs. We only review literature that is immediately relevant to our work. Recall that we solve the normal equations inexactly at each iteration, and develop a way to *correct* for the error incurred. We also focus on IPMs that can use an sufficiently positive, infeasible initial point (see Section 4.2.1). We discuss below two papers that present related ideas.

Work by [137] proposed the use of an approximate iterative solver for eqn. (4.4a), followed by a correction step to “fix” the approximate solution (see our discussion in Section 4.2.1). We propose efficient, RLA-based approaches to precondition and solve eqn. (4.4a), as well as a novel approach to correct for the approximation error in order to guarantee the convergence of the IPM algorithm. Specifically, [137] propose to solve eqn. (4.4a) using the so-called *maximum weight basis* preconditioner [170]. However, computing such a preconditioner needs access to a maximal linearly independent set of columns of AD in each iteration, which is costly, taking $\mathcal{O}(m^2 n)$ time in the worst-case. More importantly, while [136] was able to provide a bound that depends only on properties of A, and is independent of D, this bound might, in general, be very large. In contrast, our bound is a constant and it does not depend on properties of A or its dimension. In addition, [137] assumed a bound on the two-norm of the residual of the preconditioned system, but it is unclear how their preconditioner guarantees such a bound. Similar concerns exist for the construction of the correction vector v proposed by [137], which our work alleviates.

The line of research in the Theoertical Computer Science literature that is closest to our work is [55], who presented an IPM that uses an approximate solver in each iteration. They present an accuracy guarantee for the final objective value that is comparable to ours. However, their final solution satisfies the constraints approximately and thus is only approximately feasible, whereas our approach returns a final solution that satisfies the constraints exactly and is primal-dual feasible. Since [55] allows an approximately feasible solution as the end result, it does not need to correct for the error incurred in each iteration due to the approximate solver. Finally, [55] focuses on *short-step*, feasible IPMs (unlike our work) and their proposed approximate solver only works for the special case of input matrices that correspond to graph Laplacians, following the lines of [184, 183].

We also note that in the Theoretical Computer Science liteature, [107, 108, 109, 106, 110] and [51] proposed and analyzed theoretically ground-breaking algorithms for LPs based on novel tools such as the so-called *inverse maintenance* for accelerating the linear system solvers in IPMs. However, these endeavors are primarily focused on the theoretically fast but practically inefficient short-step feasible IPMs. In contrast, our work is focused on infeasible *long-step* IPMs, known to work efficiently in practice.

Another relevant line of research is the work of [54], which proposed solving eqn. (4.4a) using preconditioned Krylov subspace methods, including variants of *generalized minimum residual* (GMRES) or CG methods. Indeed, [54] conducted extensive numerical experiments on LP problems taken from standard benchmark libraries, but did not provide any theoretical guarantees.

From a matrix-sketching perspective, our work was also partially motivated by [45], which presented an iterative, sketching-based algorithm to solve under-constrained ridge regression problems, but did not address how to make use of such approaches in an iterative framework, as we do here. [162, 203] proposed the so-called *Newton sketch* to construct an approximate Hessian matrix for more general convex objective functions of which LP is a special case. Nevertheless, these randomized second-order methods are significantly faster than the conventional approach only when the data matrix is over-constrained, *i.e.* $m \gg n$. It is unclear whether the approach of [162, 203] is faster than IPMs when the optimization problem to be solved is linear. [197] proposed a probabilistic algorithm to solve LP approximately in a random projection-based reduced feature-space. A possible drawback of this paper is that the approximate solution is infeasible with respect to the original region. Finally, we refer the interested reader to the surveys [201, 65, 92, 125, 66] for more background on Randomized Linear Algebra.

4.3 Notation and Background

$A, B, \ldots$ denote matrices and $\mathbf{a}, \mathbf{b}, \ldots$ denote vectors. For vector a, $\|\mathbf{a}\|_2$ denotes its Euclidean norm; for a matrix A, $\|A\|_2$ denotes its spectral norm and $\|A\|_F$ denotes its Frobenius norm. We use $\mathbf{0}$ to denote a null vector or null matrix, dependent upon context, and $\mathbf{1}$ to denote the all-ones vector. For any matrix $X \in \mathbb{R}^{m\times n}$ with $m \leq n$ of rank m its thin Singular Value Decomposition (SVD) is the product $U\Sigma V^\mathsf{T}$, with $U \in \mathbb{R}^{m\times m}$ (the matrix of the left singular vectors), $V \in \mathbb{R}^{n\times m}$(the matrix of the top-$m$ right singular vectors), and $\Sigma \in \mathbb{R}^{m\times m}$ a diagonal matrix whose entries are equal to the singular values of X. We use $\sigma_i(\cdot)$ to denote the i-th singular value of the matrix in parentheses.

We now briefly discuss a result on matrix sketching [50, 49] that is particularly useful in our theoretical analyses. In our parlance, [50] proved that, for any matrix $Z \in \mathbb{R}^{m\times n}$, there exists a sketching matrix $W \in \mathbb{R}^{n\times w}$ such that

$$\left\|ZWW^\mathsf{T}Z^\mathsf{T} - ZZ^\mathsf{T}\right\|_2 \leq \frac{\zeta}{4}\Big(\,\|Z\|_2^2 + \frac{\|Z\|_F^2}{r}\Big) \tag{4.12}$$

holds with probability at least $1-\delta$ for any $r \geq 1$. Here $\zeta \in [0,1]$ is a (constant) accuracy parameter. Ignoring constant terms, $w = \mathcal{O}(r\log(^r/_\delta))$; W has $\mathcal{O}(\log(r/\delta))$ non-zero entries per row; and the product ZW can be computed in time $\mathcal{O}(\log(r/\delta)\cdot \mathsf{nnz}(Z))$.

4.4 Conjugate Gradient Solver

In this section, we discuss the computation of the preconditioner Q (and its inverse), followed by a discussion on how such a preconditioner can be used to satisfy eqns. (4.8) and (4.9). Algorithm 5 takes as input the sketching matrix $W \in \mathbb{R}^{n\times w}$, which we construct as discussed in Section 4.3. Our preconditioner Q is equal to

$$Q = ADWW^\mathsf{T}DA^\mathsf{T}. \tag{4.13}$$

Notice that we only need to compute $Q^{-1/2}$ in order to use it to solve eqn. (4.7). Towards that end, we first compute the sketched matrix $ADW \in \mathbb{R}^{m\times w}$. Then, we compute the SVD of the matrix ADW: let U_Q be the matrix of its left

Algorithm 5: Solving eqn. (4.7) via CG

```
Input: $AD \in \mathbb{R}^{m\times n}$, $p \in \mathbb{R}^m$, sketching matrix $W \in \mathbb{R}^{n\times w}$, iteration count $t$;
1. Compute $ADW$ and its SVD: let $U_Q$ be the matrix of its left singular vectors and let $\Sigma_Q^{-1/2}$ be the matrix of its singular values;
2. Compute $Q^{-1/2} = U_Q\Sigma_Q^{-1/2}$;
3. Initialize $\tilde{z}^0 \leftarrow \mathbf{0}_m$ and run standard CG on the preconditioned system of eqn. (4.7) for $t$ iterations;
Output: return $\tilde{z}^t$;
```

singular vectors and let $\Sigma_Q^{-1/2}$ be the matrix of its singular values. Notice that the left singular vectors of $Q^{-1/2}$ are equal to U_Q and its singular values are equal to $\Sigma_Q^{-1/2}$. Therefore, $Q^{-1/2} = U_Q\Sigma_Q^{-1/2}$.

Let $AD = U\Sigma V^\mathsf{T}$ be the thin SVD representation of AD. We apply the results of [50] (see Section 4.3) to the matrix $Z = V^\mathsf{T} \in \mathbb{R}^{m\times n}$ with $r = m$ to get that, with probability at least $1-\delta$,

$$\left\|V^\mathsf{T} W W^\mathsf{T} V - I_m\right\|_2 \le \zeta/2. \tag{4.14}$$

The running time needed to compute the sketch ADW is equal to (ignoring constant factors) $\mathcal{O}(\mathsf{nnz}(A)\cdot\log(m/\delta))$. Note that $\mathsf{nnz}(AD) = \mathsf{nnz}(A)$. The cost of computing the SVD of ADW (and therefore $Q^{-1/2}$) is $\mathcal{O}(m^3\log(1/\delta))$. Overall, computing $Q^{-1/2}$ can be done in time

$$\mathcal{O}(\mathsf{nnz}(A)\cdot\log(m/\delta) + m^3\log(1/\delta)). \tag{4.15}$$

Given these results, we now discuss how to satisfy eqns. (4.8) and (4.9) using the sketching matrix W. We start with the following bound, which is relatively straight-forward given prior RLA work (see Section 4.8.1 for a proof).

Lemma 28. *If the sketching matrix W satisfies eqn.* (4.14)*, then, for all $i = 1\ldots m$,*

$$(1+\zeta/2)^{-1} \le \sigma_i^2(Q^{-1/2}AD) \le (1-\zeta/2)^{-1}.$$

This lemma directly implies eqn. (4.8). We now proceed to show that the above construction for $Q^{-1/2}$, when combined with the conjugate gradient solver to solve eqn. (4.7), indeed satisfies eqn. (4.9)[2]. We do note that in prior work most

[2]See Chapter 9 of [122] for a detailed overview of CG.

of the convergence guarantees for CG focus on the error of the approximate solution. However, in our work, we are interested in the convergence of the *residuals* and it is known that even if the energy norm of the error of the approximate solution decreases monotonically, the norms of the CG residuals may oscillate. Interestingly, we can combine a result on the residuals of CG from [30] with Lemma 28 to prove that in our setting the norms of the CG residuals also decrease monotonically.

Let $\tilde{f}^{(j)}$ be the residual at the j-th iteration of the CG algorithm, i.e., $\tilde{f}^{(j)} = Q^{-1/2}AD^2A^\mathsf{T}Q^{-1/2}\tilde{z}^j - Q^{-1/2}p$. Recall from Algorithm 5 that $\tilde{z}^0 = \mathbf{0}$ and thus $\tilde{f}^{(0)} = -Q^{-1/2}p$. In our parlance, Theorem 8 of [30] proved the following bound.

$$\|\tilde{f}^{(j)}\|_2 \leq \frac{\kappa^2(Q^{-1/2}AD) - 1}{2}\|\tilde{f}^{(j-1)}\|_2\,,$$

where $\kappa(Q^{-1/2}AD)$ is the condition number of $Q^{-1/2}AD$. Combined with Lemma 28 (see Section 4.8.2 for details), we get

$$\|\tilde{f}^{(t)}\|_2 \leq \zeta\|\tilde{f}^{(t-1)}\|_2 \leq \cdots \leq \zeta^t\|\tilde{f}^{(0)}\|_2 = \zeta^t\|Q^{-1/2}p\|_2\,,$$

which proves the condition of eqn. (4.9).

We remark that one can consider using MINRES [155] instead of CG. Our results hinges on bounding the two-norm of the residual. MINRES finds, at each iteration, the optimal vector with respect the two-norm of the residual inside the same Krylov subspace of CG for the corresponding iteration. Thus, the bound we prove for CG applies to MINRES as well.

4.5 The Infeasible IPM algorithm

In order to avoid spurious solutions, primal-dual path-following IPMs bias the search direction towards the *central path* and restrict the iterates to a neighborhood of the central path. This search is controlled by the *centering parameter* $\sigma \in [0,1]$. At each iteration, given the current solution (x^k, y^k, s^k), a standard infeasible IPM obtains the search direction $(\Delta x^k, \Delta y^k, \Delta s^k)$ by solving the following system of linear equations:

$$AD^2A^\mathsf{T}\Delta y^k = p^k\,, \tag{4.16a}$$

$$\Delta s^k = -r_d^k - A^\mathsf{T}\Delta y^k\,, \tag{4.16b}$$

$$\Delta x^k = -x^k + \sigma\mu_k S^{-1}\mathbf{1}_n - D^2\Delta s^k. \tag{4.16c}$$

Here D and S are computed given the current iterate (x^k and s^k). After solving the above system, the infeasible IPM Algorithm 6 proceeds by computing a step-size $\bar{\alpha}$ to return:

$$(x^{k+1}, y^{k+1}, s^{k+1}) = (x^k, y^k, s^k) + \bar{\alpha}(\Delta x^k, \Delta y^k, \Delta s^k). \tag{4.17}$$

Recall that $r^k = (r_p^k, r_d^k)$ is a vector with $r_p^k = Ax^k - b$ and $r_d^k = A^\mathsf{T} y^k + s^k - c$ (the primal and dual residuals). We also use the *duality measure* $\mu_k = {x^k}^\mathsf{T} s^k / n$ and the vector

$$p^k = -r_p^k - \sigma \mu_k A S^{-1} \mathbf{1}_n + Ax^k - AD^2 r_d^k. \tag{4.18}$$

Given Δy^k from eqn. (4.16a), Δs^k and Δx^k are easy to compute from eqns. (4.16b) and (4.16c), as they only involve matrix-vector products. However, since we will use Algorithm 5 to solve eqn. (4.16a) approximately using the sketching-based preconditioned CG solver, the primal and dual residuals *do not* lie on the line segment between $\mathbf{0}$ and r^0. This invalidates known proofs of convergence for infeasible IPMs.

For notational simplicity, we now drop the dependency of vectors and scalars on the iteration counter k. Let $\hat{\Delta y} = Q^{-1/2} \tilde{z}^t$ be the approximate solution to eqn. (4.16a). In order to account for the loss of accuracy due to the approximate solver, we compute $\hat{\Delta x}$ as follows:

$$\hat{\Delta x} = \quad -x + \sigma \mu S^{-1} \mathbf{1}_n - D^2 \hat{\Delta s} - S^{-1} v. \tag{4.19}$$

Here $v \in \mathbb{R}^n$ is a perturbation vector that needs to exactly satisfy the following invariant at each iteration of the infeasible IPM:

$$AS^{-1} v = AD^2 A^\mathsf{T} \hat{\Delta y} - p\,. \tag{4.20}$$

We note that the computation of $\hat{\Delta s}$ is still done using eqn. (4.16b), which does not change. [137] argued that if v satisfies eqn. (4.20), the primal and dual residuals lie in the correct line segment.

Construction of v. There are many choices for v satisfying eqn. (4.20). A general choice is $v = (AS^{-1})^\dagger (AD^2 A^\mathsf{T} \hat{\Delta y} - p)$, which involves the computation of the pseudoinverse $(AS^{-1})^\dagger$, which is expensive, taking time $\mathcal{O}(m^2 n)$. Instead, we propose to construct v using the sketching matrix W of Section 4.3. More precisely, we construct the perturbation vector

$$v = (XS)^{1/2} W (ADW)^\dagger (AD^2 A^\mathsf{T} \hat{\Delta y} - p). \tag{4.21}$$

The following lemma proves that the proposed v satisfies eqn. (4.20); see Section 4.8.3 for the proof.

Lemma 29. *Let $W \in \mathbb{R}^{n\times w}$ be the sketching matrix of Section 4.3 and v be the perturbation vector of eqn. (4.21). Then, with probability at least $1-\delta$, $\operatorname{rank}(ADW) = m$ and v satisfies eqn.* (4.20).

We emphasize here that we will use the same exact sketching matrix $W \in \mathbb{R}^{n\times w}$ to form the preconditioner used in the CG algorithm of Section 4.4 *as well as* the vector v in eqn.(4.21). This allows us to form the sketching matrix only once, thus saving time in practice. Next, we present a bound for the two-norm of the perturbation vector v of eqn. (4.21); see Section 4.8.4 for the proof.

Lemma 30. *With probability at least $1-\delta$, our perturbation vector v in Lemma 29 satisfies*

$$\|v\|_2 \le \sqrt{3n\mu}\,\|\tilde{f}^{(t)}\|_2, \tag{4.22}$$

with $\tilde{f}^{(t)} = Q^{-1/2}AD^2A^\mathsf{T}Q^{-1/2}\tilde{z}^t - Q^{-1/2}p$.

The above result is particularly useful in proving the convergence of Algorithm 6. More precisely, combining a result from [137] with our preconditioner $Q^{-1/2}$, we can prove that $\|Q^{-1/2}p\|_2 \le \mathcal{O}(n)\sqrt{\mu}$. This bound allows us to prove that if we run Algorithm 5 for $\mathcal{O}(\log n)$ iterations, then $\|\tilde{f}^{(t)}\|_2 \le \frac{\gamma\sigma}{4\sqrt{n}}\sqrt{\mu}$ and $\|v\|_2 \le \frac{\gamma\sigma}{4}\mu$. The last two inequalities are critical in the convergence analysis of Algorithm 6; see Section 4.9 for details.

We are now ready to present the infeasible IPM algorithm. We will need the following definition for the neighborhood

$$\mathcal{N}(\gamma) = \left\{(x^k, y^k, s^k) : x_i^k s_i^k \ge (1-\gamma)\mu \text{ and } \frac{\|r^k\|_2}{\|r^0\|_2} \le \frac{\mu_k}{\mu_0}, \right\}.$$

Here $\gamma \in (0,1)$ and we note that the duality measure μ_k steadily reduces at each iteration.

Algorithm 6: Infeasible IPM

```
Input: $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, $\gamma \in (0,1)$, tolerance $\epsilon > 0$, centering parameter $\sigma \in (0, 4/5)$;
Initialize: $k \leftarrow 0$; initial point $(x^0, y^0, s^0)$;
while $\mu_k > \epsilon$ do
    (a) Compute sketching matrix $W \in \mathbb{R}^{n \times w}$ (Section 4.3) with $\zeta = 1/2$ and $\delta = O(n^{-2})$;
    (b) Compute $r_p^k = Ax^k - b$; $r_d^k = A^\mathsf{T} y^k + s^k - c$; and $p^k$ from eqn. (4.18);
    (c) Solve the linear system of eqn. (4.7) for $z$ using Algorithm 5 with $W$ from step (a) and $t = \mathcal{O}(\log n)$. Compute $\hat{\Delta y} = Q^{-1/2} z$;
    (d) Compute $v$ using eqn. (4.21) with $W$ from step (a); $\hat{\Delta s}$ using eqn. (4.16b); $\hat{\Delta x}$ using eqn. (4.19);
    (e) Compute $\tilde{\alpha} = \text{argmax}\{\alpha \in [0,1] : (x^k, y^k, s^k) + \alpha(\hat{\Delta x}^k, \hat{\Delta y}^k, \hat{\Delta s}^k) \in \Gamma\}$.
    (f) Compute $\bar{\alpha} = \text{argmin}\{\alpha \in [0, \tilde{\alpha}] : (x^k + \alpha \hat{\Delta x}^k)^\mathsf{T} (s^k + \alpha \hat{\Delta s}^k)\}$.
    (g) Compute $(x^{k+1}, y^{k+1}, s^{k+1}) = (x^k, y^k, s^k) + \bar{\alpha}(\hat{\Delta x}^k, \hat{\Delta y}^k, \hat{\Delta s}^k)$; set $k \leftarrow k+1$;
```

Running time. We start by discussing the running time to compute v. As discussed in Section 4.4, $(ADW)^\dagger$ can be computed in $\mathcal{O}(\mathsf{nnz}(A) \cdot \log(m/\delta) + m^3 \log(m/\delta))$ time. Now, as W has $\mathcal{O}(\log(m/\delta))$ non-zero entries per row, pre-multiplying by W takes $\mathcal{O}(\mathsf{nnz}(A) \log(m/\delta))$ time (assuming $\mathsf{nnz}(A) \geq n$). Since X and S are diagonal matrices, computing v takes $\mathcal{O}(\mathsf{nnz}(A) \cdot \log(m/\delta) + m^3 \log(m/\delta))$ time, which is asymptotically the same as computing $Q^{-1/2}$ (see eqn. (4.15)).

We now discuss the overall running time of Algorithm 6. At each iteration, with failure probability δ, the preconditioner $Q^{-1/2}$ and the vector v can be computed in $\mathcal{O}(\mathsf{nnz}(A) \cdot \log(m/\delta) + m^3 \log(m/\delta))$ time. In addition, for $t = \mathcal{O}(\log n)$ iterations of Algorithm 5, all the matrix-vector products in the CG solver can be computed in $\mathcal{O}(\mathsf{nnz}(A) \cdot \log n)$ time. Therefore, the computational time for steps (a)-(d) is given by $\mathcal{O}(\mathsf{nnz}(A) \cdot (\log n + \log(m/\delta)) + m^3 \log(m/\delta))$. Finally, taking a union bound over all iterations with $\delta = \mathcal{O}(n^{-2})$ (ignoring constant factors), Algorithm 6 converges with probability at least 0.9. The running time at each iteration is given by $\mathcal{O}((\mathsf{nnz}(A) + m^3) \log n)$.

4.6 Discussion and Extensions

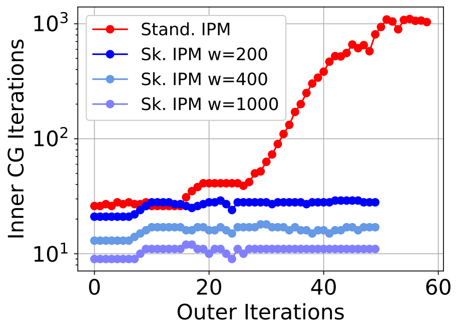
We briefly discuss extensions of our work. First, there is nothing special about using a CG solver for solving eqn. (4.7). We can replace the proposed CG solver with another iterative solver, without any loss in accuracy or any increase in the number of iterations for the long-step infeasible IPM Algorithm 6 of Section 4.5.

Additionally, recall that our approach focused on full rank input matrices $A \in \mathbb{R}^{m\times n}$ with $m \ll n$. Our overall approach still works if A in any $m \times n$ matrix that is low-rank, e.g., $\text{rank}(A) = k \ll \min\{m, n\}$. In that case, using the thin SVD of A, we can rewrite the linear constraints as follows $U_A\Sigma_A V_A^\mathsf{T} x = b$, where $U_A \in \mathbb{R}^{m\times k}$ and $V_A \in \mathbb{R}^{n\times k}$ are the matrices of left and right singular vecors of A respectively; $\Sigma_A \in \mathbb{R}^{k\times k}$ is the diagonal matrix with the k non-zero singular values of A as its diagonal elements. The LP of eqn. (4.1) can be restated as

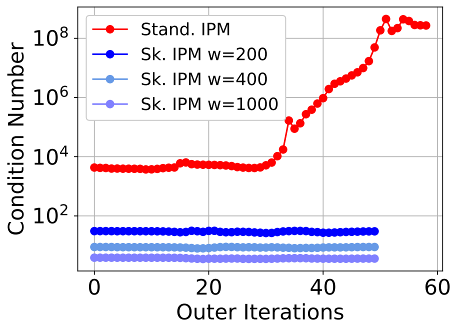
$$\min\, c^\mathsf{T} x\,,\ \text{subject to } V_A^\mathsf{T} x = \widetilde{b}\,, x \geq \mathbf{0}\,, \tag{4.23}$$

where $\widetilde{b} = \Sigma_A^{-1} U_A^\mathsf{T} b$. Note that, $\text{rank}(V_A) = k \ll n$ and therefore eqn. (4.23) can be solved using our framework. The matrices U_A, V_A, and Σ_A can be approximately recovered using the fast SVD algorithms of [92, 29, 48].

Finally, even though we chose to use the Count-Min sketch and its analysis from [50] (Section 4.3), there are many other alternative sketching matrix constructions that would lead to similar results. A particularly simple one is the Gaussian sketching matrix $W_G \in \mathbb{R}^{n\times w}$, where every entry is a $\mathcal{N}(0, 1)$ random variable. Setting $w = O\left({}^{m+\log(1/\delta)}/_{\varepsilon^2}\right)$ would result in the same accuracy guarantees as the sketching matrix of Section 4.3. However, the (theoretical) running time needed to compute ADW increases to $O(m\cdot\mathsf{nnz}(A))$. In practice, at least for relatively small matrices, using Gaussian sketching matrices is a reasonable alternative; see the discussion in [132] which argued that the Gaussian matrix sketching-based solvers are considerably better than direct solvers. We also opted to use Gaussian matrices in our empirical evaluation, since we primarily interested in measuring the accuracy of the final solution as a function of the number of iterations of the solver and the IPM algorithm. Other known constructions of sketching matrices that are also applicable in our setting include (any) sub-gaussian sketching matrix; the Subsampled Randomized Hadamard transform (SRHT); and any of the Sparse Subspace Embeddings of [47, 144, 131, 49].

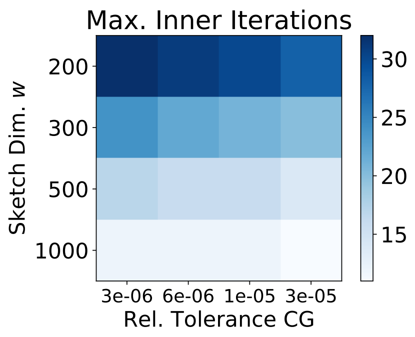


(a)

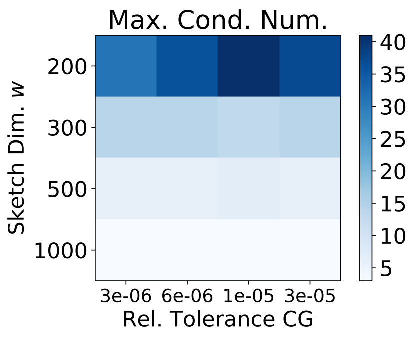


(b)

Figure 41: *ARCENE data set:* Our Algorithm 6 (Sk. IPM) requires an order of magnitude fewer (a) inner iterations than the Standard IPM with CG, at each outer iteration, due to the improved (b) conditioning of $Q^{-1/2}AD^2A^\mathsf{T}Q^{-1/2}$ compared to AD^2A^T.



(a) Max. Inner CG Iterations.



(b) Max. Condition Number.

Figure 42: *ARCENE data set*: for various (w, *tolCG*) settings, (a) the maximum number of inner iterations used by our algorithm and (b) the maximum condition number of $Q^{-1/2}AD^2A^\mathsf{T}Q^{-1/2}$, across outer iterations. The standard IPM, across all settings, needed on the order of 1,000 iterations and $\kappa(AD^2A^T)$ was on the order of 10^8. The relative error was fixed to 0.04%.

4.7 Experiments

Here we demonstrate the empirical performance of our algorithm on a variety of real-world datasets from the UCI ML Repository [67], such as ARCENE, DEXTER [91], DrivFace [61], and a gene expression cancer RNA-Sequencing dataset that is part of the PANCAN dataset [200]. See Table 41 for a description of the datasets.

We also generated random synthetic instances of linear programs as follows. To generate $A \in \mathbb{R}^{m\times n}$, we set $a_{ij} \sim_{i.i.d.} U(0,1)$ with probability p and $a_{ij} = 0$ otherwise. We then add $\min\{m,n\}$ i.i.d. draws from $U(0,1)$ to the main diagonal, to ensure each row of A has at least one nonzero entry. We set $b = Ax + 0.1z$, where x and z are random vectors drawn from $N(0,1)$. Finally, we set $c \sim N(0,1)$.

We observed that the results for both synthetic data and real-world data were qualitatively similar. Thus, we highlight results on several representative real datasets. The experiments were implemented in Python and run on a server with Intel E5-2623V3@3.0GHz 8 cores and 64GB RAM.

Support Vector Machines (SVMs) As an application, we consider ℓ_1-regularized SVMs. All of the datasets are concerned with binary classification with $m \ll n$, where n is the number of features. Below, we describe the ℓ_1-SVM problem and how it can be formulated as an LP. Here, m is the number of training points, n is the feature dimension, and the size of the constraint matrix in the LP becomes $m \times (2n+1)$.

The classical ℓ_1-SVM problem is as follows. We consider the task of fitting an SVM to data pairs $S = \{(x_i, y_i)\}_{i=1}^m$, where $x_i \in \mathbb{R}^N$ and $y_i \in \{+1, -1\}$ is a label for each data pair. Here, m is the number of training points, and N is the feature dimension. The SVM problem with an ℓ_1 regularizer has the following form:

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & \|w\|_1 \\ \text{subject to} \quad & y_i(w^T x_i + b') \geq 1, \quad \forall i \in [m]. \end{aligned} \tag{4.24}$$

This problem can be written as an LP by introducing the variables w^+ and w^-, where $w = w^+ - w^-$. The objective becomes $\sum_j^n w_j^+ + w_j^-$, and we constrain $w_i^+ \geq 0$ and $w_i^- \geq 0$. Note that the size of the constraint matrix in the LP becomes $(m \times (2N+1))$, where m is the number of training points, and N is the feature dimension.

Comparisons and Metrics. We compare our Algorithm 6 with a standard IPM (see Chapter 10, [164]) using CG, and a standard IPM using a direct solver. We also use CVXPY as a benchmark to compare the accuracy of the solutions; we define the *relative error* $\|\hat{x}-x^\star\|_2/\|x^\star\|_2$, where $\hat{x}$ is our solution and $x^\star$ is the solution generated by CVXPY. We also consider the number of *outer iterations*, namely the number of iterations of the IIPM algorithm, as well as the number of *inner iterations*, namely the number of iterations of the CG solver. We denote the relative stopping tolerance for CG by *tolCG* and we denote the outer iteration residual error by τ. If not specified: $\tau = 10^{-9}$, $tolCG = 10^{-5}$, and $\sigma = 0.5$. We evaluated a Gaussian sketching matrix, and the initial triplet (x, y, s) for all IPM algorithms was set to be all ones.

Experimental Results. Figure 41(a) shows that our Algorithm 6 uses an order of magnitude fewer *inner* iterations than the un-preconditioned standard solver. This is due to the improved conditioning of the respective matrices in the normal equations, as demonstrated in Figure 41(b). Across various real and synthetic data sets, the results were qualitatively similar to those shown in Figure 41. Results for several real data sets are summarized in Table 41.

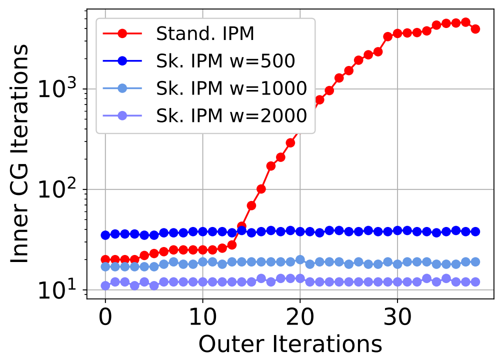
The number of *outer* iterations is unaffected by our internal approximation methods, and is generally the same for our Algorithm 6, the standard IPM with CG, and the standard IPM with a direct linear solver (denoted IPM w/Dir), as seen in Table 41. Figure 41 also demonstrates the relative insensitivity to the choice of w (the sketching dimension, i.e., the number of columns of the sketching matrix W of Section 4.3). For smaller values of w, our algorithm requires more inner iterations. However, across various choices of w, the number of inner iterations is always an order of magnitude smaller than the number required by the standard solver.

Figure 42 shows the performance of our algorithm for a range of (w, *tolCG*) pairs. Figure 42(a) demonstrates that the number of the inner iterations is robust to the choice of *tolCG* and w. The number of inner iterations varies between 15 and 35 for the ARCENE data set, while the standard IPM took on the order of $1,000$ iterations across all parameter settings. Across all settings, the relative error was fixed at 0.04%. In general, our sketched IPM is able to produce an extremely high accuracy solution across parameter settings. Thus we do not report additional numerical results for the relative error, which was consistently 10^{-3} or less. Figure 42(b) demonstrates a tradeoff of our approach: as both *tolCG* and w are increased, the condition number $\kappa(Q^{-1/2}AD^2A^\mathsf{T}Q^{-1/2})$ decreases, corresponding to better conditioned systems. As a result, fewer inner iterations are required.

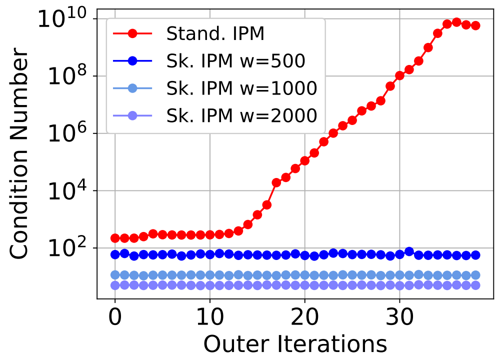
Figure 43 illustrates the convergence and conditioning behavior for the DEXTER data set. We see a similar behavior as found for the ARCENE data set in Figure 41. Figure 44 displays more results for the ARCENE data set.

The following is how we made use of a gene expression cancer RNA-Sequencing data set, taken from the UCI Machine Learning repository. It is part of the RNA-Seq (HiSeq) PANCAN data set [200], and is a random extraction of gene expressions from patients who have different types of tumors: BRCA, KIRC, COAD, LUAD, and PRAD. We considered the binary classification task of identifying BRCA versus other types.

The following is how we made use of the DrivFace data set taken from the UCI Machine Learning repository. In the DrivFace data set, each sample corresponds to an image of a human subject, taken while driving in real scenarios. Each image is labeled as corresponding to one of three possible gaze directions: left, straight, or right. We considered the binary classification task of identifying two different gaze directions: straight, or to either side left or right.

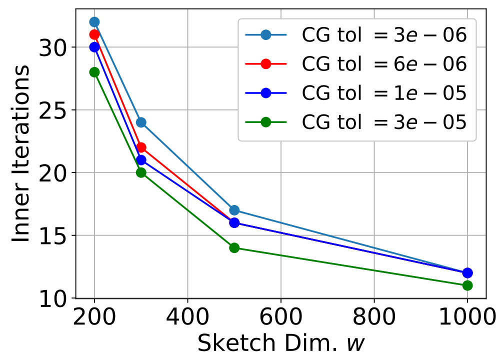


(a)

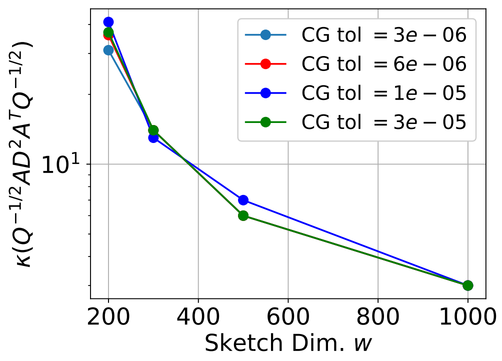


(b)

Figure 43: *DEXTER data set*: Our algorithm (Sk. IPM) requires an order of magnitude fewer inner iterations than the Standard IPM with CG, at each outer iteration, as demonstrated in (a). This is possible due to the improved conditioning of $Q^{-1/2}AD^2A^{\mathsf{T}}Q^{-1/2}$ compared to AD^2A^T, demonstrated in (b). For all, $tolCG\ = 10^{-5}$, $\tau = 10^{-9}$.



(a)



(b)

Figure 44: *ARCENE data set*: As w increases, (a) the number of inner iterations decreases, and is relatively robust to *tolCG*, and, (b) the condition number decreases as well.

Table 41: Comparison of (our) sketched IPM with CG, standard IPM with CG, and Standard IPM with a direct solver, for the ℓ_1-SVM problem on UCI Machine Learning Repository [67] data sets. Across all, $\tau = 10^{-9}$ and a relative error of 10^{-3} or less was achieved. We define $\kappa_{\text{Sk}} = \kappa(Q^{-1/2}AD^2A^\mathsf{T}Q^{-1/2})$ and $\kappa_{\text{Stan}} = \kappa(AD^2A^T)$.

Problem	**Size**	**Sketch IPM w/ Precond. CG**				**Stand. IPM w/ Unprec. CG**			**IPM w/ Dir.**
	$(m \times N)$	w	In. It.	Out. It.	κ_{Sk}	In. It.	Out. It.	κ_{Stan}	Out. It.
ARCENE	$(100 \times 10K)$	200	**30**	50	38.09	**1.1K**	59	4.4×10^8	50
DEXTER	$(300 \times 20K)$	500	**39**	39	75.42	**4.6K**	39	7.6×10^9	39
DrivFace	(606×6400)	1000	**50**	42	68.87	**139K**	43	17×10^{12}	42
Gene RNA	(801×20531)	2000	**27**	44	20.03	**101K**	208	4.7×10^{12}	44

4.8 Proofs of Technical Results

First, we define some additional notation. For simplicity, we assume $B = AD$ in most of the proofs in Section 4.8. We also take $B = U\Sigma V^\mathsf{T}$ to be the thin SVD representation of B. Additionally, for vectors $a = (a_1, \ldots, a_\ell)^\mathsf{T}$ and $b = (b_1, \ldots, b_\ell)^\mathsf{T}$, we denote $a \circ b = (a_1 b_1, \ldots, a_\ell b_\ell)^\mathsf{T}$. For any vector $a \in \mathbb{R}^n$, the ℓ_∞ norm is defined as, $\|a\|_\infty = \max_i |a_i|$ and we will heavily use the following standard inequality to prove results in the upcoming sections,

$$\left|\frac{a^\mathsf{T}\mathbf{1}_n}{n}\right| \le \|a\|_\infty \le \|a\|_2 \,. \tag{4.25}$$

4.8.1 Proof of Lemma 28

Proof. For simplicity, we assume $B = AD$. Consider the condition of eqn. (4.14):

$$\|V^\mathsf{T}WW^\mathsf{T}V - I_m\|_2 \le \frac{\zeta}{2} \;\Leftrightarrow\; -\frac{\zeta}{2}I_m \preccurlyeq V^\mathsf{T}WW^\mathsf{T}V - I_m \preccurlyeq \frac{\zeta}{2}I_m$$

$$\Rightarrow\; -\frac{\zeta}{2}BB^\mathsf{T} \preccurlyeq BWW^\mathsf{T}B^\mathsf{T} - BB^\mathsf{T} \preccurlyeq \frac{\zeta}{2}BB^\mathsf{T} \tag{4.26}$$

$$\Rightarrow\; \left(1-\frac{\zeta}{2}\right)BB^\mathsf{T} \preccurlyeq \underbrace{BWW^\mathsf{T}B^\mathsf{T}}_{Q} \preccurlyeq \left(1+\frac{\zeta}{2}\right)BB^\mathsf{T}\,, \tag{4.27}$$

where we obtain eqn. (4.26) by pre- and post-multiplying the previous inequality by $U\Sigma$ and ΣU^T respectively and using the facts that $B = U\Sigma V^\mathsf{T}$ and $BB^\mathsf{T} = U\Sigma^2 U^\mathsf{T}$. Next, pre- and post- multiplying eqn. (4.27) by $Q^{-1/2}$, we obtain

$$\left(1+\frac{\zeta}{2}\right)^{-1} I_n \preccurlyeq Q^{-1/2}BB^\mathsf{T}Q^{-1/2} \preccurlyeq \left(1-\frac{\zeta}{2}\right)^{-1} I_m\,. \tag{4.28}$$

Eqn. (4.28) implies that all the eigenvalues of $Q^{-1/2}BB^\mathsf{T}Q^{-1/2}$ are bounded between $\left(1+\frac{\zeta}{2}\right)^{-1}$ and $\left(1-\frac{\zeta}{2}\right)^{-1}$. Therefore, we have

$$\left(1+\frac{\zeta}{2}\right)^{-1} \le \sigma_i^2(Q^{-1/2}B) \le \left(1-\frac{\zeta}{2}\right)^{-1}$$

. □

4.8.2 Satisfying eqn. (4.9) using CG Solver

Let $\tilde{f}^{(j)}$ be the residual at the j-th iteration of the CG algorithm, i.e., $\tilde{f}^{(j)} = Q^{-1/2}AD^2A^\mathsf{T}Q^{-1/2}\tilde{z}^j - Q^{-1/2}p$. Recall from Algorithm 5 that $\tilde{z}^0 = \mathbf{0}$ and thus $\tilde{f}^{(0)} = -Q^{-1/2}p$. Theorem 8 of [30] proved the following bound.

Lemma 31 (Theorem 8 of [30]). *Let $\tilde{f}^{(j-1)}$ and $\tilde{f}^{(j)}$ be the residuals obtained by the CG solver at steps $j-1$ and j. Then,*

$$\|\tilde{f}^{(j)}\|_2 \leq \frac{\kappa^2(Q^{-1/2}AD) - 1}{2}\|\tilde{f}^{(j-1)}\|_2\,,$$

where $\kappa(Q^{-1/2}AD)$ is the condition number of $Q^{-1/2}AD$.

From Lemma 28, we get

$$\kappa^2(Q^{-1/2}AD) = \frac{\sigma^2_{\max}(Q^{-1/2}AD)}{\sigma^2_{\min}(Q^{-1/2}AD)} \leq \frac{1+\zeta/2}{1-\zeta/2}. \tag{4.29}$$

Combining eqn. (4.29) with Lemma 31,

$$\begin{aligned}\|\tilde{f}^{(j)}\|_2 &\leq \frac{\frac{1+\zeta/2}{1-\zeta/2} - 1}{2}\|\tilde{f}^{(j-1)}\|_2 \\ &= \frac{\zeta}{2-\zeta}\|\tilde{f}^{(j-1)}\|_2 \leq \ \zeta\|\tilde{f}^{(j-1)}\|_2\,,\end{aligned} \tag{4.30}$$

where the last inequality follows from $\zeta \leq 1$. Applying eqn. (4.30) recursively, we get

$$\|\tilde{f}^{(t)}\|_2 \leq \zeta\|\tilde{f}^{(t-1)}\|_2 \leq \cdots \leq \zeta^t\|\tilde{f}^{(0)}\|_2 = \zeta^t\|Q^{-1/2}p\|_2\,,$$

which proves the condition of eqn. (4.9).

4.8.3 Proof of Lemma 29

Proof. Let $AD = U\Sigma V^{\mathsf{T}}$ be the thin SVD representation of AD. We use the exact same W as discussed in Section 4.4. Therefore, eqn. (4.14) holds with probability $1-\delta$.

Next, using the using Weyl's inequality, we have for $i = 1, 2, \ldots, m$

$$\left\|V^{\mathsf{T}}WW^{\mathsf{T}}V - I_m\right\|_2 \geq \left|\sigma_i^2(V^{\mathsf{T}}W) - 1\right|\,. \tag{4.31}$$

Now, combining eqns. (4.14) and (4.31), we further have

$$(1 - \zeta/2)^{1/2} \leq \sigma_i(V^{\mathsf{T}}W) \leq (1 + \zeta/2)^{1/2}\,, \tag{4.32}$$

which implies the top m singular values of $V^{\mathsf{T}}W$ and non-zero. Therefore, $\text{rank}(V^{\mathsf{T}}W) = m$, as it has exactly m rows. Next, using SVD of AD, we rewrite ADW as

$$ADW = U\Sigma(V^{\mathsf{T}}W)\,. \tag{4.33}$$

Note that $U\Sigma$ is a non-singular matrix and we know rank of a matrix remains unaltered by pre (or post)-multiplying by a non-singular matrix. Therefore, we have $\operatorname{rank}(ADW) = \operatorname{rank}(V^\mathsf{T} W) = m$.

As ADW has full *row-rank*, right-inverse exists and $ADW\,(ADW)^\dagger = I_m$. Therefore, taking $v = (XS)^{-1/2} W (ADW)^\dagger (AD^2 A^\mathsf{T} \hat{\Delta y} - p)$, we finally have

$$\begin{aligned} AS^{-1}\,v &= AS^{-1}(XS)^{1/2} W (ADW)^\dagger (AD^2 A^\mathsf{T} \hat{\Delta y} - p) \\ &= ADW (ADW)^\dagger (AD^2 A^\mathsf{T} \hat{\Delta y} - p) \\ &= AD^2 A^\mathsf{T} \hat{\Delta y} - p\,, \end{aligned}$$

where the second last equality follows from the fact that $D = X^{1/2} S^{-1/2}$. This concludes the proof. $\square$

4.8.4 Proof of Lemma 30

Proof. We already have, $Q = ADW(ADW)^\mathsf{T}$. Let $ADW = \widehat{U}\widehat{\Sigma}\widehat{V}^\mathsf{T}$ be the thin SVD representation of ADW. Therefore, $Q = \widehat{U}\widehat{\Sigma}^2\widehat{U}^\mathsf{T}$. Now, from Lemma 29, we know ADW has full row rank. Therefore, $\widehat{\Sigma}$ has the all m diagonal entries positive which implies $Q^{1/2} Q^{-1/2} = I_m$.

Next, we bound $\|v\|_2$ in the following way

$$\begin{aligned} \|v\|_2 &= \|(XS)^{1/2} W (ADW)^\dagger (AD^2 A^\mathsf{T} \hat{\Delta y} - p)\|_2 \\ &= \|(XS)^{1/2} W (ADW)^\dagger Q^{1/2} Q^{-1/2} (AD^2 A^\mathsf{T} \hat{\Delta y} - p)\|_2 \\ &\le \|(XS)^{1/2} W (ADW)^\dagger Q^{1/2}\|_2\, \|\tilde{f}^{(t)}\|_2\,, \end{aligned} \tag{4.34}$$

where we have used the fact that $Q^{-1/2}(AD^2 A^\mathsf{T} \hat{\Delta y} - p) = \tilde{f}^{(t)}$ and the last inequality follows from the sub-multiplicativity property of spectral-norm.

Again using SVD of ADW, we have $(ADW)^\dagger = \widehat{V}\widehat{\Sigma}^{-1}\widehat{U}^\mathsf{T}$ and $Q^{1/2} = \widehat{U}\widehat{\Sigma}\widehat{U}^\mathsf{T}$, which implies $(ADW)^\dagger Q^{1/2} = \widehat{V}\widehat{V}^\mathsf{T}$. So, $(ADW)^\dagger Q^{1/2}$ is a projection matrix and we know that if we multiply a matrix with a projection matrix, it doesn't increase the matrix-norm. Using this, from eqn. (4.34) we further have,

$$\|v\|_2 \le \|(XS)^{1/2} W \widehat{V}\widehat{V}^\mathsf{T}\|_2 \|\tilde{f}^{(t)}\|_2 \le \|(XS)^{1/2} W\|_2 \|\tilde{f}^{(t)}\|_2 \tag{4.35}$$

As we use the exact same W discussed in Section 4.4 to construct v and note that eqn. (4.12) holds for any matrix Z. Therefore, using with $Z = (XS)^{1/2}$

with that W , eqn. (4.12) in Section 4.3 boils down to

$$\left\|(XS)^{1/2}WW^{\mathsf{T}}(XS)^{1/2}-(XS)\right\|_2 \le \frac{\zeta}{4}\left(\|(XS)^{1/2}\|_2^2+\frac{\|(XS)^{1/2}\|_F^2}{m}\right) \quad (4.36)$$

holds with probability at least $1-\delta$ for any $r \ge 1$.

Now, applying Weyl's inequality on the left hand side of the eqn. (4.36), we further have

$$\left|\left\|(XS)^{1/2}W\right\|_2^2-\left\|(XS)^{1/2}\right\|_2^2\right| \le \frac{\zeta}{4}\left(\|(XS)^{1/2}\|_2^2+\frac{\|(XS)^{1/2}\|_F^2}{m}\right). \quad (4.37)$$

Now, using the facts that $\zeta \le 1$, $\|(XS)^{1/2}\|_2 \le \|(XS)^{1/2}\|_F$, and $\frac{\|(XS)^{1/2}\|_F^2}{m} \le \|(XS)^{1/2}\|_F^2$, from eqn. (4.37),

$$\left\|(XS)^{1/2}W\right\|_2^2 \le 3\|(XS)^{1/2}\|_F^2 = 3n\mu\,, \quad (4.38)$$

where the last equality follows from $\|(XS)^{1/2}\|_F^2 = x^{\mathsf{T}}s = n\mu\,.$

Finally, combining eqns. (4.35) and (4.38), we conclude

$$\|v\|_2 \le \sqrt{3n\mu}\|\tilde{f}^{(t)}\|_2.$$

□

4.9 Convergence Analysis of Infeasible IPM Algorithm 6

4.9.1 Number of Iterations for the CG Solver

In this section, most of the proofs follow [137] except for the fact that we used our sketching based preconditioner $Q^{-1/2}$.

Lemma 32. *Let (x^0,y^0,s^0) be the initial point with $(x^0,s^0) > \mathbf{0}$ and $(x^*,y^*,s^*) \in \mathcal{S}$ such that $(x^*,s^*) \le (x^0,s^0)$ with $s^0 \ge |A^{\mathsf{T}}y^0-c|$. Then, for any point $(x,y,s) \in \mathcal{N}(\gamma)$ such that $r=\eta\, r^0$ and $0 \le \eta \le \min\left\{1, \frac{s^{\mathsf{T}}x}{s^{0\mathsf{T}}x^0}\right\}$, then we have*

$$(i)\quad \eta\,(x^{\mathsf{T}}s^0+s^{\mathsf{T}}x^0) \le\ 3n\mu\,, \quad (4.39a)$$

$$(ii)\quad \eta\,\|S(x^*-x^0)\|_2 \le \eta\,\|Sx^0\|_2 \le \eta s^{\mathsf{T}}x^0 \le\ 3n\mu\,, \quad (4.39b)$$

$$(iii)\quad \eta\,\|X(s^0+A^{\mathsf{T}}y^0-c)\|_2 \le\ 2\eta\,\|Xs^0\|_2 \le\ 2\eta\,x^{\mathsf{T}}s^0 \le\ 6n\mu\,. \quad (4.39c)$$

Proof. We prove eqns. (4.39a)–(4.39c) below.

Proof of eqn. (4.39a)**.** For completeness, we provide a proof of eqn. (4.39a) which is already discussed in [137]. Since $(x^*, s^*, y^*) \in \mathcal{S}$, the following equalities hold:

$$Ax^* = b \tag{4.40a}$$
$$A^\mathsf{T} y^* + s^* = c. \tag{4.40b}$$

Furthermore, $r = \eta r^0$ implies

$$Ax - b = \eta(Ax^0 - b) \tag{4.41a}$$
$$A^\mathsf{T} y + s - c = \eta(A^\mathsf{T} y^0 + s^0 - c). \tag{4.41b}$$

Combining eqn. (4.40a) with eqn. (4.41a) and eqn. (4.40b) with eqn. (4.41b), we get

$$A\big(x - \eta x^0 - (1-\eta)x^*\big) = \mathbf{0} \tag{4.42a}$$
$$A^\mathsf{T}(y - \eta y^0 - (1-\eta)y^*) + (s - \eta s^0 - (1-\eta)s^*) = \mathbf{0}. \tag{4.42b}$$

Multiplying the eqn. (4.42b) by $(x - \eta x^0 - (1-\eta)x^*)^\mathsf{T}$ on the left and using eqn. (4.42a), we get

$$\big(x - \eta x^0 - (1-\eta)x^*\big)^\mathsf{T} \big(s - \eta s^0 - (1-\eta)s^*\big) = 0\,,$$

expanding which we get

$$\begin{aligned}\eta\left(x^{0^\mathsf{T}} s + x^\mathsf{T} s^0\right) &= \eta^2 x^{0^\mathsf{T}} s^0 + (1-\eta)^2 (x^*)^\mathsf{T} s^* + x^\mathsf{T} s \\ &\quad + \eta(1-\eta)\left(x^{0^\mathsf{T}} s^* + (x^*)^\mathsf{T} s^0\right) - (1-\eta)\big((x^*)^\mathsf{T} s + x^\mathsf{T} s^*\big)\,.\end{aligned} \tag{4.43}$$

Next, we use the given conditions and rewrite eqn. (4.43) as

$$\begin{aligned}\eta\left(x^{0^\mathsf{T}} s + s^{0^\mathsf{T}} x\right) &\le \eta^2 x^{0^\mathsf{T}} s^0 + x^\mathsf{T} s + \eta(1-\eta)\left(x^{0^\mathsf{T}} s^* + s^{0^\mathsf{T}} x^*\right) \\ &\le \eta^2 x^{0^\mathsf{T}} s^0 + x^\mathsf{T} s + 2\eta(1-\eta) x^{0^\mathsf{T}} s^0 \\ &\le 2\eta x^{0^\mathsf{T}} s^0 + x^\mathsf{T} s \le 3x^\mathsf{T} s = 3n\mu\,,\end{aligned} \tag{4.44}$$

where the first inequality in eqn. (4.44) follows from from a couple of facts. First, $(1-\eta)((x^*)^\mathsf{T} s + x^\mathsf{T} s^*) \ge 0$ as $(x^*, s^*) \ge \mathbf{0}$ and $(x^0, s^0) \ge \mathbf{0}$; second, as $(x^*, s^*, y^*) \in \mathcal{S}$ (which implies $x^* \circ s^* = \mathbf{0}$), we have $(x^*)^\mathsf{T} s^* = 0$. Second inequality in eqn. (4.44) holds as $x^* \leq x^0$, $s^* \leq s^0$, $(x^*, s^*) \geq \mathbf{0}$ and $(x^0, s^0) \geq \mathbf{0}$; combining which we have $(x^{0^\mathsf{T}} s^* + s^{0^\mathsf{T}} x^*) \leq 2\, x^{0^\mathsf{T}} s^0$. Third inequality in eqn. (4.44) is true as we have $\eta^2 x^{0^\mathsf{T}} + 2\eta(1-\eta) x^{0^\mathsf{T}} s^0 = 2\eta x^{0^\mathsf{T}} s^0 - \eta^2 x^{0^\mathsf{T}} s^0 \leq 2\eta x^{0^\mathsf{T}} s^0$. Final inequality holds as $\eta \leq \frac{x^\mathsf{T} s}{x^{0^\mathsf{T}} s^0}$. □

Proof of eqn. (4.39b)**.** The last inequality directly follows from eqn. (4.39a); second last inequality is also easy to prove as

$$\|Sx^0\|_2 = \sqrt{\sum_{i=1}^{s} (s_i x_i^0)^2} \leq \sqrt{\left(\sum_{i=1}^{s} s_i x_i^0\right)^2} = s^\mathsf{T} x^0 . \tag{4.45}$$

To prove the first inequality in eqn. (4.39b), we use the fact $x^0 \geq x^*$ as follows

$$\begin{aligned}\|Sx^0\|_2^2 - \|S(x^* - x^0)\|_2^2 &= \sum_{i=1}^{n} (s_i x_i^0)^2 - \sum_{i=1}^{n} s_i^2 \left((x_i^*)^2 + (x_i^0)^2 - 2x_i^* x_i^0\right) \\ &= \sum_{i=1}^{n} s_i^2 \left(2x_i^* x_i^0 - (x_i^*)^2\right) \geq 0 .\end{aligned}$$

□

Proof of eqn. (4.39c)**.** This can be proven using a similar approach as in eqn. (4.39b). Last inequality directly follows from eqn. (4.39a); second last inequality is also easy to prove as

$$\|Xs^0\|_2 = \sqrt{\sum_{i=1}^{n} (x_i s_i^0)^2} \leq \sqrt{\left(\sum_{i=1}^{n} x_i s_i^0\right)^2} = x^\mathsf{T} s^0 . \tag{4.46}$$

For the first inequality, we proceed as follows

$$\begin{aligned}\|X(s^0 + A^\mathsf{T} y^0 - c)\|_2^2 &= \|Xs^0\|_2^2 + \|X(A^\mathsf{T} y^0 - c)\|_2^2 + 2 s^{0^\mathsf{T}} X^\mathsf{T} X (A^\mathsf{T} y^0 - c) \\ &= \|Xs^0\|_2^2 + \sum_{i=1}^{n} x_i^2 (A^\mathsf{T} y^0 - c)_i^2 + 2\sum_{i=1}^{n} x_i^2 s_i^0 (A^\mathsf{T} y^0 - c)_i \\ &\leq \|Xs^0\|_2^2 + \sum_{i=1}^{n} (x_i s_i^0)^2 + 2\sum_{i=1}^{n} (x_i s_i^0)^2 \\ &= \|Xs^0\|_2^2 + \|Xs^0\|_2^2 + 2\|Xs^0\|_2^2 = 4\|Xs^0\|_2^2 ,\end{aligned} \tag{4.47}$$

where the inequality in eqn. (4.47) follows from $x_i \geq 0$, $s_i^0 \geq 0$ and $\left|(A^\mathsf{T} y^0 - c)_i\right| \leq s_i^0$ for all $i = 1, 2, \ldots n$. This concludes the proof of Lemma 32. □

Our next result bounds $\|Q^{-1/2}p\|_2$ which will be instrumental in in proving the final convergence bound.

Lemma 33. *Let (x^0, y^0, s^0) be the initial point with $(x^0, s^0) > \mathbf{0}$ such that $x^0 \geq x^*$ and $s^0 \geq \max\{s^*, |c - A^\mathsf{T} y^0|\}$ for some $(x^*, y^*, s^*) \in \mathcal{S}$. Furthermore, let $(x, y, s) \in \mathcal{N}(\gamma)$ with $r = \eta\, r^0$ for some $0 \leq \eta \leq 1$. If the sketching matrix $W \in \mathbb{R}^{n \times w}$ satisfies the condition in eqn.* (4.8)*, then*

$$\|Q^{-1/2}p\|_2 \leq \sqrt{2}\left(\frac{9n}{\sqrt{1-\gamma}} + \sigma\sqrt{\frac{n}{1-\gamma}} + \sqrt{n}\right)\sqrt{\mu}\,.$$

Recall that, $r = (r_p, r_d) = (Ax - b, A^\mathsf{T} y + s - c)$ and $r^0 = (r_p^0, r_d^0) = (Ax^0 - b, A^\mathsf{T} y^0 + s^0 - c)$.

Proof. Note that in our case, after correcting the approximation error of the CG solver using v, the primal and dual residuals $r = (r_p, r_d)$ corresponding to an iterate $(x, y, s) \in \mathcal{N}(\gamma)$ always lie on the line segment between zero and $r^{(0)}$. In other words, $r = \eta r^{(0)}$ always holds for some $\eta \in [0, 1]$. This was formally proven in Lemma 35.

To bound $\|Q^{-1/2}p\|_2$, first we express p as in eqn. (4.4a) and rewrite

$$Q^{-1/2}p = Q^{-1/2}\left(-r_p - \sigma\mu B(XS)^{-1/2}\mathbf{1}_n + BD^{-1}x - BDr_d\right). \tag{4.48}$$

Then, applying triangle inequality on $\|Q^{-1/2}p\|_2$ in eqn. (4.48), we get

$$\|Q^{-1/2}p\|_2 \leq \Delta_1 + \Delta_2 + \Delta_3 + \Delta_4\,, \tag{4.49}$$

where

$$\begin{aligned}
\Delta_1 &= \|Q^{-1/2}r_p\|_2\,,\\
\Delta_2 &= \sigma\mu\|Q^{-1/2}B(XS)^{-1/2}\mathbf{1}_n\|_2\,,\\
\Delta_3 &= \|Q^{-1/2}BD^{-1}x\|_2\,,\\
\Delta_4 &= \|Q^{-1/2}BDr_d\|_2\,.
\end{aligned}$$

To bound Δ_1, Δ_2, Δ_3, and Δ_4 separately, we will heavily use the condition in eqn. (4.8). In particular, from eqn. (4.8), note that we have $\|Q^{-1/2}B\|_2 \leq \sqrt{2}$ as $\zeta \leq 1$.

Bounding Δ_1. Putting $r_p = \eta\, r_p^0$, $r_p^0 = Ax^0 - b$ and $b = Ax^*$, we rewrite Δ_1 as

$$\begin{aligned}\Delta_1 &= \eta\, \|Q^{-1/2}A(x^0 - x^*)\|_2 \\ &= \eta\, \|Q^{-1/2}BD^{-1}(x^0 - x^*)\|_2 \\ &\le \eta\, \|Q^{-1/2}B\|_2 \|D^{-1}(x^0 - x^*)\|_2 \\ &\le \sqrt{2}\eta\, \|D^{-1}(x^0 - x^*)\|_2 \\ &= \sqrt{2}\eta\, \|(XS)^{-1/2}S(x^0 - x^*)\|_2 \\ &\le \sqrt{2}\eta\, \|(XS)^{-1/2}\|_2\, \|S(x^0 - x^*)\|_2\,, \end{aligned} \tag{4.50}$$

where the above steps follow from submultiplicativity and eqn. (4.8). From eqn. (4.8), note that we have $\|Q^{-1/2}B\|_2 \le \sqrt{2}$ as $\zeta \le 1$. Now, applying eqn. (4.39b) and $\|(XS)^{-1/2}\|_2 = \max_{1\le i\le n} \frac{1}{\sqrt{x_i s_i}}$, we further have

$$\begin{aligned}\Delta_1 &\le \sqrt{2}\, \max_{1\le i\le n} \frac{1}{\sqrt{x_i s_i}} \cdot 3n\mu \\ &\le 3\sqrt{2}\, n\sqrt{\frac{\mu}{1-\gamma}}\,, \end{aligned} \tag{4.51}$$

where the last inequality follows from $(x, y, s) \in \mathcal{N}(\gamma)$.

Bounding Δ_2. Applying submultiplicativity, we have

$$\begin{aligned}\Delta_2 &= \sigma\mu\, \|Q^{-1/2}\, B\, (XS)^{-1/2}\mathbf{1}_n\|_2 \\ &\le \sigma\mu\, \|Q^{-1/2}\, B\|_2 \|(XS)^{-1/2}\mathbf{1}_n\|_2 \\ &\le \sqrt{2}\, \sigma\mu\, \|(XS)^{-1/2}\mathbf{1}_n\|_2 \\ &= \sqrt{2}\, \sigma\mu\, \sqrt{\sum_{i=1}^{n} \frac{1}{x_i s_i}} \le\; \sqrt{2}\, \sigma\mu\, \sqrt{\sum_{i=1}^{n} \frac{1}{(1-\gamma)\mu}} \\ &= \sqrt{2}\, \sigma\, \sqrt{\frac{n\, \mu}{(1-\gamma)}}\,, \end{aligned} \tag{4.52}$$

where the second last inequality follows from eqn. (4.8) and last inequality holds as $(x, y, s) \in \mathcal{N}(\gamma)$.

Bounding Δ_3. Putting $D = S^{-1/2}X^{1/2}$; $x = X\,\mathbf{1}_n$ and

$$\begin{aligned}\Delta_3 &= \|Q^{-1/2}\,B\,(S^{1/2}X^{-1/2})\,X\,\mathbf{1}_n\|_2\\ &= \|Q^{-1/2}\,B\,(SX)^{1/2}\,\mathbf{1}_n\|_2\\ &\le \|Q^{-1/2}\,B\|_2\|(SX)^{1/2}\,\mathbf{1}_n\|_2\\ &\le \sqrt{2}\sqrt{\sum_{i=1}^{n} x_i s_i} = \sqrt{2n\,\mu}\,, \end{aligned} \tag{4.53}$$

where the inequalities follows respectively from submultiplicativity and eqn. (4.8).

Bounding Δ_4. Putting $r_d = \eta\, r_d^0$, we have

$$\begin{aligned}\Delta_4 &= \eta\|Q^{-1/2}\,B\,Dr_d^0\|_2\\ &\le \eta\|Q^{-1/2}\,B\|_2\|(XS)^{-1/2}Xr_d^0\|_2\\ &\le \sqrt{2}\eta\,\|(XS)^{-1/2}X(A^\mathsf{T}y^0+s^0-c)\|_2\\ &\le \sqrt{2}\eta\,\|(XS)^{-1/2}\|_2\,\|X(A^\mathsf{T}y^0+s^0-c)\|_2\,,\end{aligned}$$

where the above inequalities follow from submultiplicativity and eqn. (4.8). Now, applying eqn. (4.39c) and $\|(XS)^{-1/2}\|_2 \le \frac{1}{\sqrt{(1-\gamma)\mu}}$, we further have

$$\Delta_4 \le 6\sqrt{2}n\sqrt{\frac{\mu}{1-\gamma}} \tag{4.54}$$

Final bound. Combining eqns. (4.49), (4.51), ,(4.52), (4.53) and (4.54)

$$\|Q^{-1/2}p\|_2 \le \sqrt{2}\left(\frac{9n}{\sqrt{1-\gamma}} + \sigma\sqrt{\frac{n}{1-\gamma}} + \sqrt{n}\right)\sqrt{\mu}\,. \tag{4.55}$$

This concludes the proof of Lemma 33. □

Lemma 34. *Let the the sketching matrix W satisfies the conditions in eqns.* (4.8) *and* (4.9)*. Then, after $t \ge \frac{\log(4\sqrt{6n}\,\psi/\gamma\sigma)}{\log(1/\zeta)}$ iterations of the CG solver in Algorithm 5, we have the following:*

$$\|\tilde{f}^{(t)}\|_2 \le \frac{\gamma\sigma}{4\sqrt{n}}\sqrt{\mu} \quad \textit{and} \quad \|v\|_2 \le \frac{\gamma\sigma}{4}\mu\,,$$

where $\psi = \left(\frac{9n}{\sqrt{1-\gamma}} + \sigma\sqrt{\frac{n}{1-\gamma}} + \sqrt{n}\right)$ *and* $\tilde{f}^{(t)} = Q^{-1/2}AD^2A^\mathsf{T}Q^{-1/2}\tilde{z}^t - Q^{-1/2}p$ *is the residual of the solver.*

Proof. Combining Lemma 33 and the condition in eqn. (4.9), we have

$$\|\tilde{f}^{(t)}\|_2 \le \zeta^t \psi \sqrt{2\mu}. \tag{4.56}$$

Now, $\|\tilde{f}^{(t)}\|_2 \le \frac{\gamma\sigma}{4\sqrt{n}}\sqrt{\mu}$ holds if

$$\sqrt{2}\psi\,\zeta^t\sqrt{\mu} \le \frac{\gamma\sigma}{4\sqrt{n}}\sqrt{\mu} \quad \text{holds, } i.e. \text{ if } \left(\frac{1}{\zeta}\right)^t \ge \frac{4\sqrt{2\,n}\,\psi}{\gamma\sigma} \quad \text{holds.}$$

$$i.e. \text{ if } t \ge \frac{\log(4\sqrt{2n}\,\psi/\gamma\sigma)}{\log(1/\zeta)} \quad \text{holds,} \quad \text{which is true.}$$

Next, combining Lemma 30 and eqn. (4.56) we get

$$\|v\|_2 \le \sqrt{3n\mu}\,\|\tilde{f}^{(t)}\|_2 \le \sqrt{6n}\,\zeta^t\psi\mu.$$

Therefore, $\|v\|_2 \le \frac{\gamma\sigma\mu}{4}$ holds if

$$\sqrt{6n}\psi\,\zeta^t\psi\mu \le \frac{\gamma\sigma\mu}{4} \quad \text{holds, if } \left(\frac{1}{\zeta}\right)^t \ge \frac{4\sqrt{6\,n}\,\psi}{\gamma\sigma} \quad \text{holds.}$$

$$i.e. \text{ if } t \ge \frac{\log(4\sqrt{6n}\,\psi/\gamma\sigma)}{\log(1/\zeta)} \quad \text{holds,} \quad \text{which is also true.}$$

Now, fixing γ and σ and ζ, we indeed need $t = \mathcal{O}(\log n)$ iterations of Algorithm 5 for $\|\tilde{f}^{(t)}\|_2 \le \frac{\gamma\sigma}{4\sqrt{n}}\sqrt{\mu}$ and $\|v\|_2 \le \frac{\gamma\sigma\mu}{4}$ to hold. □

4.9.2 Determining Step-size and Final Convergence

In this section, most of the proofs match that of [137], which we reproduce here:

Let $(\hat{\Delta x}, \hat{\Delta y}, \hat{\Delta s})$ respectively satisfies eqns. (4.19), (4.16a) and (4.16b). We rewrite the systen in the following alternative form

$$A\hat{\Delta x} = -\,r_p\,, \tag{4.57a}$$

$$A^{\mathsf{T}}\hat{\Delta y} + \hat{\Delta s} = -\,r_d\,, \tag{4.57b}$$

$$X\hat{\Delta s} + S\hat{\Delta x} = -\,XS\,\mathbf{1}_n + \sigma\mu\,\mathbf{1}_n - v. \tag{4.57c}$$

We define each new point traversed by the algorithm as $(x(\alpha), y(\alpha), s(\alpha))$, where

$$(x(\alpha), y(\alpha), s(\alpha)) := (x, y, s) + \alpha(\hat{\Delta x}, \hat{\Delta y}, \hat{\Delta s}) \tag{4.58}$$

$$\mu(\alpha) := x(\alpha)^T s(\alpha)/n \tag{4.59}$$

$$r(\alpha) := r\,(x(\alpha), s(\alpha), y(\alpha))\,. \tag{4.60}$$

The goal in this section is to bound the number of outer iterations required by Algorithm 6. To do so, we make arguments about the magnitude of the step size, α. First, we provide an upper bound on α, which allows us to show that each new point $(x(\alpha), s(\alpha), y(\alpha))$ traversed by the algorithm stays within the neighborhood Γ. Second, we provide a lower bound on α, which allows us to bound the number of iterations required.

The following Lemma will be used throughout the subsequent arguments.

Lemma 35 (Lemma 3.3 of [137]). *Assume $(\hat{\Delta x}, \hat{\Delta s}, \hat{\Delta y})$ satisfies equations* (4.57) *for some $\sigma \in \mathbb{R}$, $v \in \mathbb{R}^n$ and (x, y, s) be any point such that $(x, s) > 0$. Then, for every $\alpha \in \mathbb{R}$,*

$$\begin{aligned}
&(a) \quad x(\alpha) \circ s(\alpha) = (1-\alpha) x \circ s + \alpha\sigma\mu \mathbf{1}_n - \alpha v + \alpha^2 \hat{\Delta x} \circ \hat{\Delta s}\,, \\
&(b) \quad \mu(\alpha) = [1 - \alpha(1-\sigma)]\mu - \frac{\alpha\, v^{\mathsf{T}} \mathbf{1}_n}{n} + \frac{\alpha^2\, \hat{\Delta x}^{\mathsf{T}} \hat{\Delta s}}{n}\,, \\
&(c) \quad r(\alpha) = (1-\alpha) r.
\end{aligned}$$

Proof. Considering (4.57c), we obtain (a) due to the following:

$$\begin{aligned}
x(\alpha) \circ s(\alpha) &= (x + \alpha\hat{\Delta x}) \circ (s + \alpha\hat{\Delta s}) \\
&= x \circ s + \alpha(x \circ \hat{\Delta s} + s\hat{\Delta x}) + \alpha^2 \hat{\Delta x} \circ \hat{\Delta s} \\
&= x \circ s + \alpha(-x \circ s + \sigma\mu\mathbf{1}_n - v) + \alpha^2 \hat{\Delta x} \circ \hat{\Delta s} \\
&= (1-\alpha) x \circ s + \alpha\sigma\mu\mathbf{1}_n - \alpha v + \alpha^2 \hat{\Delta x} \circ \hat{\Delta s}.
\end{aligned}$$

To obtain (b), left multiply the above inequality by $\mathbf{1}_n^{\mathsf{T}}$ and divide by n. Lastly (c) follows from (4.57a) and (4.57b). □

We begin with providing an upper bound on α, ensuring that each new point $(x(\alpha), y(\alpha), s(\alpha))$ traversed by the algorithm stays within the neighborhood Γ.

Lemma 36 (Lemma 3.5 of [137]). *Assume $(\hat{\Delta x}, \hat{\Delta y}, \hat{\Delta s})$ satisfies equations* (4.57) *for some $\sigma > 0$, $(x, y, s) \in \Gamma$ for $\gamma \in (0, 1)$ and $\|v\|_2 \le \frac{\gamma\sigma\mu}{4}$. Then, $(x(\alpha), y(\alpha), s(\alpha)) \in \Gamma$ for every scalar α such that*

$$0 \le \alpha \le \min\left\{1, \frac{\gamma\sigma\mu}{4\|\hat{\Delta x} \circ \hat{\Delta s}\|_\infty}\right\}. \tag{4.61}$$

Proof. We begin by showing that $\frac{\|r(\alpha)\|_2}{\|r\|_2} \le \frac{\mu(\alpha)}{\mu}$. The assumptions $\gamma \in (0,1)$ and the inequality in eqn. (4.25) imply that $\frac{v^\mathsf{T}\mathbf{1}_n}{n} \le \frac{\sigma\mu}{4}$. Thus, considering Lemma 35(b),

$$\begin{aligned}
\mu(\alpha) &= [1-\alpha(1-\sigma)]\mu - \alpha\,\frac{v^\mathsf{T}\mathbf{1}_n}{n} + \alpha^2\,\frac{\hat{\Delta x}^\mathsf{T}\hat{\Delta s}}{n} \\
&\ge [1-\alpha(1-\sigma)]\mu - \alpha\,\frac{\sigma\mu}{4} + \alpha^2\,\frac{\hat{\Delta x}^\mathsf{T}\hat{\Delta s}}{n} \\
&\ge (1-\alpha)\mu + \alpha\,\frac{3\,\sigma\mu}{4} - \alpha^2\|\hat{\Delta x}\circ\hat{\Delta s}\|_\infty \\
&\ge (1-\alpha)\mu + \alpha\,\|\hat{\Delta x}\circ\hat{\Delta s}\|_\infty\left(\frac{3\,\sigma\mu}{4\,\|\hat{\Delta x}\circ\hat{\Delta s}\|_\infty} - \alpha\right) \\
&\ge (1-\alpha)\mu, \qquad (4.62)
\end{aligned}$$

where the first inequality follows from $\frac{v^\mathsf{T}\mathbf{1}_n}{n} \le \frac{\sigma\mu}{4}$, and the last inequality holds as $\alpha \le \frac{\gamma\sigma\mu}{4\|\hat{\Delta x}\circ\hat{\Delta s}\|_\infty}$ and $\gamma < 1$. Now, due to eqn. (4.62) and Lemma 35(c), we see that

$$\frac{\|r(\alpha)\|_2}{\|r\|_2} \le \frac{\mu(\alpha)}{\mu} \qquad (4.63)$$

for all α satisfying eqn. (4.61). Thus, we argue that $\Big(x(\alpha), y(\alpha), s(\alpha)\Big) \in \Gamma$ by the following. First, we see that

$$\frac{\|r(\alpha)\|_2}{\|r^0\|_2} = \frac{\|r(\alpha)\|_2}{\|r\|_2}\frac{\|r\|_2}{\|r^0\|_2} \le \frac{\mu(\alpha)}{\mu}\frac{\mu}{\mu_0} = \frac{\mu(\alpha)}{\mu_0}. \qquad (4.64)$$

Next, we have to show that $x(\alpha)\circ s(\alpha) \ge (1-\gamma)\mu(\alpha)\mathbf{1}_n$. Applying Lemma 35(a) and Lemma 35(b), we compute

$$\begin{aligned}
&x(\alpha)\circ s(\alpha) - (1-\gamma)\mu(\alpha)\mathbf{1}_n \\
&= (1-\alpha)\left(x\circ s - (1-\gamma)\mu\,\mathbf{1}_n\right) + \alpha\gamma\sigma\mu\,\mathbf{1}_n - \alpha\left(v - (1-\gamma)\frac{v^\mathsf{T}\mathbf{1}_n}{n}\mathbf{1}_n\right) \\
&\qquad\qquad + \alpha^2\left(\hat{\Delta x}\circ\hat{\Delta s} - (1-\gamma)\frac{\hat{\Delta x}^\mathsf{T}\hat{\Delta s}}{n}\mathbf{1}_n\right) \\
&\ge \alpha\left(\gamma\sigma\mu - \left\|v - (1-\gamma)\frac{v^\mathsf{T}\mathbf{1}_n}{n}\,\mathbf{1}_n\right\|_\infty - \alpha\left\|\hat{\Delta x}\circ\hat{\Delta s} - (1-\gamma)\frac{\hat{\Delta x}^\mathsf{T}\hat{\Delta s}}{n}\,\mathbf{1}_n)\right\|_\infty\right)\mathbf{1}_n \\
&\ge \alpha\Bigg(\gamma\sigma\mu - 2\|v\|_\infty - 2\alpha\|\hat{\Delta x}\circ\hat{\Delta s}\|_\infty\Bigg)\mathbf{1}_n \\
&\ge \alpha\Bigg(\gamma\sigma\mu - \frac{\gamma\sigma\mu}{2} - \frac{\gamma\sigma\mu}{2}\Bigg)\mathbf{1}_n = \mathbf{0},
\end{aligned}$$

where the first inequality follows from $x \circ s \geq (1-\gamma)\mu\,\mathbf{1}_n$ (as we have, $(x,y,s) \in \mathcal{N}(\gamma)$ and $a \leq \|a\|_\infty\,\mathbf{1}_n$ for any vector $a \in \mathbb{R}^n$). In the second last inequality, we apply the fact that for any $u \in \mathbb{R}^n$ and $\delta \in [0,n]$, it holds that $\left\|u - \delta\frac{u^\mathsf{T}\mathbf{1}_n}{n}\,\mathbf{1}_n\right\|_\infty \leq (1+\delta)\|u\|_\infty$. □

We now turn to the task of providing a lower bound on the values of $\bar{\alpha}$ and the corresponding $\mu(\bar{\alpha})$ value.

Lemma 37 (Lemma 3.6 of [137]). *In each iteration of Algorithm 6, if $\|v\|_2 \leq \frac{\gamma\sigma\mu}{4}$, then the step size $\bar{\alpha}$ satisfies*

$$\bar{\alpha} \geq \min\left\{1, \frac{\min\{\gamma\sigma, (1-\frac{5}{4}\sigma)\}\mu}{4\|\hat{\Delta x} \circ \hat{\Delta s}\|_\infty}\right\} \tag{4.65}$$

and

$$\mu(\bar{\alpha}) = \left[1 - \frac{\bar{\alpha}}{2}(1-\frac{5}{4}\sigma)\right]\mu. \tag{4.66}$$

Proof. Applying the inequality in eqn. (4.25), we have the following bound,

$$-\frac{v^\mathsf{T}\mathbf{1}_n}{n} \leq \|v\|_\infty \leq \|v\|_2 \leq \frac{1}{4}\gamma\sigma\mu. \tag{4.67}$$

Recall that $\tilde{\alpha}$ describes the maximum possible step size allowed, as calculated in step (e) of Algorithm 6, and $\bar{\alpha}$ describes the actual step size used, calculated in step (f). The relation between them is $\bar{\alpha} \in [0,\tilde{\alpha}]$. Here, our goal is to describe $\bar{\alpha}$ and $\mu(\bar{\alpha})$. First, recall that by Lemma 36 and step (e) of Algorithm 6,

$$\tilde{\alpha} \geq \min\left\{1, \frac{\gamma\sigma\mu}{4\|\hat{\Delta x} \circ \hat{\Delta s}\|_\infty}\right\}. \tag{4.68}$$

Also recall from Lemma 35(b) the expression for $\mu(\alpha)$:

$$\mu(\alpha) = [1-\alpha(1-\sigma)]\mu - \alpha v^\mathsf{T}\mathbf{1}_n/n + \alpha^2\hat{\Delta x}^\mathsf{T}\hat{\Delta s}/n. \tag{4.69}$$

We will evaluate $\bar{\alpha}$ and $\mu(\bar{\alpha})$ for the following two cases.

Case 1: $\hat{\Delta x}^\mathsf{T}\hat{\Delta s} \leq 0$. In this case, we rewrite the definition of $\bar{\alpha}$ from step (f) of Algorithm 6

$$\bar{\alpha} = \operatorname*{argmin}_{\alpha}\ \underbrace{(x+\alpha\hat{\Delta x})^\mathsf{T}(s+\alpha\hat{\Delta s})}_{g(\alpha)}, \text{ subject to } \alpha \leq \tilde{\alpha}\,.$$

Next, we show that $g(\alpha)$ is decreasing in α whenever $\hat{\Delta x}^{\mathsf{T}} \hat{\Delta s} \leq 0$. We rewrite $g(\alpha)$ as follows:

$$\begin{aligned} g(\alpha) &= (x + \alpha \hat{\Delta x})^{\mathsf{T}} (s + \alpha \hat{\Delta s}) \\ &= x^{\mathsf{T}} s + \alpha \left(x^{\mathsf{T}} \hat{\Delta s} + s^{\mathsf{T}} \hat{\Delta x} \right) + \alpha^2 \, \hat{\Delta x}^{\mathsf{T}} \hat{\Delta s} \\ &= x^{\mathsf{T}} s + \alpha \left(-x^{\mathsf{T}} s + n\sigma\mu - v^{\mathsf{T}} \mathbf{1}_n \right) + \alpha^2 \, \hat{\Delta x}^{\mathsf{T}} \hat{\Delta s} \,, \end{aligned} \tag{4.70}$$

where the last equality follows from pre-multiplying eqn. (4.57c) by $\mathbf{1}_n^{\mathsf{T}}$. Now, taking derivatives with respect to α on the both sides of eqn. (4.70), we have

$$\begin{aligned} g'(\alpha) = &\ \left(-x^{\mathsf{T}} s + n\sigma\mu - v^{\mathsf{T}} \mathbf{1}_n \right) + 2\alpha \, \hat{\Delta x}^{\mathsf{T}} \hat{\Delta s} \\ \leq &\ \left(-n\mu + n\sigma\mu + \frac{n\sigma\mu}{4} \right) + 2\alpha \, \hat{\Delta x}^{\mathsf{T}} \hat{\Delta s} \\ = &\ \left(\frac{5\sigma}{4} - 1 \right) n\mu + 2\alpha \, \hat{\Delta x}^{\mathsf{T}} \hat{\Delta s} \ \leq \ 0 \,, \end{aligned} \tag{4.71}$$

where the first inequality is due to eqn. (4.67) and the fact that $\gamma < 1$; the last inequality follows from the facts that $\sigma \leq \frac{5}{4}$, $\alpha \geq 0$ and $\hat{\Delta x}^{\mathsf{T}} \hat{\Delta s} \leq 0$. This concludes that $g(\alpha)$ is indeed decreasing in $0 \leq \alpha \leq 1$ which implies $\bar{\alpha} = \tilde{\alpha}$. Combining eqn. (4.68) with this fact yields

$$\bar{\alpha} \geq \min \left\{ 1, \frac{\gamma\sigma\mu}{4 \| \hat{\Delta x} \circ \hat{\Delta s} \|_\infty} \right\}.$$

Now, we use eqn. (4.69) and $\hat{\Delta x}^{\mathsf{T}} \hat{\Delta s} \leq 0$ to evaluate $\mu(\bar{\alpha})$

$$\begin{aligned} \mu(\bar{\alpha}) &\leq [1 - \bar{\alpha}(1-\sigma)]\mu - \bar{\alpha} \, \frac{v^{\mathsf{T}} \mathbf{1}_n}{n} = \mu - \left((1-\sigma)\mu + \frac{v^{\mathsf{T}} \mathbf{1}_n}{n} \right) \bar{\alpha} \\ &\leq \mu + \Big(-(1-\sigma)\mu + \|v\|_\infty \Big) \bar{\alpha} \leq \mu + \left(-(1-\sigma)\mu + \frac{\gamma\sigma\mu}{4} \right) \bar{\alpha} \\ &\leq \mu + \left(-(1-\sigma)\mu + \frac{\sigma\mu}{4} \right) \bar{\alpha} = \mu + \left(-\mu + \frac{5\,\sigma\mu}{4} \right) \bar{\alpha} = \left(1 - \Big(1 - \frac{5\,\sigma}{4} \Big) \bar{\alpha} \right) \mu \\ &\leq \left(1 - \Big(1 - \frac{5\,\sigma}{4} \Big) \frac{\bar{\alpha}}{2} \right) \mu, \end{aligned} \tag{4.72}$$

where we applied eqn. (4.67) along with the fact that $\sigma < \frac{4}{5}$.

Case 2: $\hat{\Delta x}^{\mathsf{T}} \hat{\Delta s} > 0$. In this case, first we consider the minimizer of the following unconstrained optimization problem

$$\alpha_{\min} = \operatorname*{argmin}_{\alpha} \underbrace{(x + \alpha \hat{\Delta x})^{\mathsf{T}} (s + \alpha \hat{\Delta s})}_{g(\alpha)} \,.$$

Taking the first derivative of $g(\alpha)$ and solving for α, we see that

$$\alpha_{\min} = \frac{n\mu(1-\sigma) + v^{\mathsf{T}}\mathbf{1}_n}{2\hat{\Delta x}^{\mathsf{T}}\hat{\Delta s}} \geq \frac{n\mu(1-\sigma) - \frac{1}{4}\sigma n\mu}{2\hat{\Delta x}^{\mathsf{T}}\hat{\Delta s}} \geq \frac{\mu(1-\frac{5}{4}\sigma)}{2\|\hat{\Delta x} \circ \hat{\Delta s}\|_\infty}, \tag{4.73}$$

where the first inequality follows from eqn. (4.67) and $\gamma \leq 1$, the last inequality holds as $\frac{\hat{\Delta x}^{\mathsf{T}}\hat{\Delta s}}{n} \leq \|\hat{\Delta x} \circ \hat{\Delta s}\|_\infty$. Now, there can be two possibilities: whenever $\alpha_{\min} \leq \tilde{\alpha}$, we have $\bar{\alpha} = \alpha_{\min}$ and if $\alpha_{\min} > \tilde{\alpha}$, then we have $\bar{\alpha} = \tilde{\alpha}$. Therefore, we get $\bar{\alpha} = \min\{\alpha_{\min}, \tilde{\alpha}\}$. Finally, we combine eqns. (4.68) and (4.73) to get

$$\bar{\alpha} = \min\{\alpha_{\min}, \tilde{\alpha}\} \geq \min\left\{1, \frac{\min\{\gamma\sigma, (1-\frac{5}{4}\sigma)\}\mu}{4\|\hat{\Delta x} \circ \hat{\Delta s}\|_\infty}\right\}.$$

Now we would like to produce an upper bound on $\mu(\bar{\alpha})$. Note that $\mu(\alpha)$ is convex,[3] and so the function must lie beneath a linear function interpolating $\mu(\bar{\alpha})$ at $\alpha = 0$ and $\alpha = \alpha_{\min}$ *i.e.* let $\phi(\alpha) = p\,\alpha + q$ be the line joining the points $\Big(0, \mu(0)\Big)$ and $\Big(\alpha_{\min}, \mu(\alpha_{\min})\Big)$. Therefore, using the expression of $\mu(\alpha)$ in Lemma 35(b), we determine p and q through the following equations

$$\phi(0) = \mu(0) \quad \text{and} \quad \phi\left(\alpha_{\min}\right) = \mu\left(\alpha_{\min}\right).$$

First equation gives $q = \mu$ and using this we find p from the second equation as follows

$$\begin{aligned}
&\phi\left(\alpha_{\min}\right) = \mu\left(\alpha_{\min}\right) \\
&\Rightarrow p\,\alpha_{\min} + \mu = \left(1 - \alpha_{\min}(1-\sigma)\right)\mu - \alpha_{\min}\frac{v^\top\mathbf{1}_n}{n} + \alpha_{\min}^2\frac{\hat{\Delta x}^{\mathsf{T}}\hat{\Delta s}}{n} \\
&\Rightarrow p\,\alpha_{\min} + \mu = \mu - \alpha_{\min}\left((1-\sigma)\mu + \frac{v^\top\mathbf{1}_n}{n}\right) + \alpha_{\min}^2\frac{\hat{\Delta x}^{\mathsf{T}}\hat{\Delta s}}{n} \\
&\Rightarrow p = -\left((1-\sigma)\mu + \frac{v^\top\mathbf{1}_n}{n}\right) + \alpha_{\min}\frac{\hat{\Delta x}^{\mathsf{T}}\hat{\Delta s}}{n},
\end{aligned}$$

where the last step follows from $\alpha_{\min} \neq 0$. Next, putting the value of $\alpha_{\min}$ from eqn. (4.73), we further have

$$\begin{aligned}
p =& -\left((1-\sigma)\mu + \frac{v^\top\mathbf{1}_n}{n}\right) + \frac{n\mu(1-\sigma) + v^\top\mathbf{1}_n}{2\hat{\Delta x}^{\mathsf{T}}\hat{\Delta s}} \cdot \frac{\hat{\Delta x}^{\mathsf{T}}\hat{\Delta s}}{n} \\
=& -\frac{1}{2}\left((1-\sigma)\mu + \frac{v^\top\mathbf{1}_n}{n}\right).
\end{aligned} \tag{4.74}$$

[3]From the expression of $\mu(\alpha)$ in Lemma 35(b), note that $\mu(\alpha)$ is twice differentiable w.r.t. α and $\mu''(\alpha) = \frac{2\,\hat{\Delta x}^{\mathsf{T}}\hat{\Delta s}}{n}$ which is positive as $\hat{\Delta x}^{\mathsf{T}}\hat{\Delta s} > 0$.

Now, $\mu(\alpha)$ is convex in $0 \le \alpha \le \alpha_{\min}$, we further have, $\mu(\bar{\alpha}) \le \phi(\bar{\alpha})$. Using this and the values of $p = -\frac{1}{2}\left((1-\sigma)\mu + \frac{v^\top \mathbf{1}_n}{n}\right)$ and $q = \mu$ from the above expressions,

$$\begin{aligned}
\mu(\bar{\alpha}) \le\ & \phi(\bar{\alpha}) = p\,\bar{\alpha} + q = -\frac{1}{2}\left((1-\sigma)\mu + \frac{v^\top \mathbf{1}_n}{n}\right)\bar{\alpha} + \mu \\
=\ & \left(1 - (1-\sigma)\frac{\bar{\alpha}}{2}\right)\mu - \frac{v^\mathsf{T}\mathbf{1}n}{n}\cdot\frac{\bar{\alpha}}{2} \\
\le\ & \left(1 - (1-\sigma)\frac{\bar{\alpha}}{2}\right)\mu + \frac{\sigma\mu}{4}\cdot\frac{\bar{\alpha}}{2} = \left(1 - \frac{\bar{\alpha}}{2}\left(1 - \frac{5}{4}\sigma\right)\right)\mu\,, \qquad (4.75)
\end{aligned}$$

where the inequality is due to eqn. (4.67) and $\gamma < 1$. This concludes the proof.

□

At this point, we have provided a lower bound (4.65) for the allowed values of the step size $\bar{\alpha}$. Next, we will show that this lower bound is bounded away from zero. Observing (4.65), it it sufficient to show that $\|\hat{\Delta x} \circ \hat{\Delta s}\|_\infty$ is bounded.

Lemma 38 (Lemma 3.7 of [137]). *Let (x^0, y^0, s^0) be the starting point with $(x^0, s^0) > 0$ such that $(x^0, s^0) \ge (x^*, s^*)$ for some $(x^*, y^*, s^*) \in \mathcal{S}$, and let $(x, y, s) \in \Gamma$ be such that $r = \eta r^0$ for some $\eta \in [0, 1]$. Then, the search direction $(\hat{\Delta x}, \hat{\Delta y}, \hat{\Delta s})$ produced by Algorithm 6 at each iteration satisfies*

$$\max\{\|D^{-1}\hat{\Delta x}\|_2, \|D\hat{\Delta s}\|_2\} \le \left(1 + \frac{\sigma^2}{1-\gamma} - 2\sigma\right)^{1/2}\sqrt{n} + \frac{6n\mu}{\sqrt{(1-\gamma)}}\sqrt{\mu} + \frac{\gamma\sigma}{4}\sqrt{3\,\mu}. \qquad (4.76)$$

Proof. Equations (4.57a) and (4.57b) and assumption $r = \eta r^0$ imply that

$$\begin{aligned}
A(\hat{\Delta x} + \eta(x^0 - x^*)) &= 0 \\
A^\mathsf{T}(\hat{\Delta y} + \eta(y^0 - y^*)) + (\hat{\Delta s} + \eta(s^0 - s^*)) &= 0
\end{aligned}$$

from which it follows that $(\hat{\Delta x} + \eta(x^0 - x^*))^\mathsf{T}(\hat{\Delta s} + \eta(s^0 - s^*)) = 0$. Multiplying (4.57c) on the left by $(XS)^{-1/2}$, we obtain

$$D^{-1}\hat{\Delta x} + D\hat{\Delta s} = H(\sigma) - (XS)^{-1/2}v\,,$$

where $H(\sigma) := -(XS)^{1/2}\mathbf{1}_n + \sigma\mu(XS)^{-1/2}\mathbf{1}_n$. Equivalently,

$$\begin{aligned}D^{-1}(\hat{\Delta x} + \eta(x^0 - x^*)) &+ D(\hat{\Delta s} + \eta(s^0 - s^*)) \\ &= H(\sigma) + \eta\left(D(s^0 - s^*) + D^{-1}(x^0 - x^*)\right) - (XS)^{-1/2}v. \qquad (4.77)\end{aligned}$$

Notice that the two terms on the left hand side are orthogonal. Also, recalling the choice of our v in Lemma 29 and in the proof of Lemma 30, taking $Z = I_n$ yields

$$\begin{aligned}\|(XS)^{-1/2}v\|_2 &= \|W(ADW)^\dagger(AD^2A^\mathsf{T}\hat{\Delta y} - p)\|_2 \\ &\leq \|W(ADW)^\dagger Q^{1/2}\|_2\|\tilde{f}^{(t)}\|_2 = \|W\widehat{V}\widehat{V}^\mathsf{T}\|_2\|\tilde{f}^{(t)}\|_2 \\ &\leq \sqrt{3n}\|\tilde{f}^{(t)}\|\,, \qquad (4.78)\end{aligned}$$

where $\widehat{V}$ the matrix of the right singular vectors in the thin SVD representation of ADW. the last inequality is follows from the fact that $\widehat{V}\widehat{V}^\mathsf{T}$ is a projection matrix and the proof of Lemma 30 with $Z = I_n$.

Now, it is easy to verify that if a and b are any two orthogonal vectors, then $\max\{\|a\|_2, \|b\|_2\} \leq \|a + b\|_2$. Combining this with eqn. (4.77) and applying triangle inequality, we therefore have

$$\begin{aligned}\max&\left\{\|D^{-1}(\hat{\Delta x} + \eta(x^0 - x^*))\|_2, \|D(\hat{\Delta s} + \eta(s^0 - s^*))\|_2\right\} \\ &\leq \|D^{-1}(\hat{\Delta x} + \eta(x^0 - x^*)) + D(\hat{\Delta s} + \eta(s^0 - s^*))\|_2 \\ &\leq \|H(\sigma)\|_2 + \eta\left(\|D(s^0 - s^*)\|_2 + \|D^{-1}(x^0 - x^*)\|_2\right) + \|(XS)^{-1/2}v\|_2 \\ &\leq \|H(\sigma)\|_2 + \eta\left(\|D(s^0 - s^*)\|_2 + \|D^{-1}(x^0 - x^*)\|_2\right) + \sqrt{3n}\,\|\tilde{f}^{(t)}\|_2\,, \qquad (4.79)\end{aligned}$$

where the last inequality follows from eqn. (4.78).

Again, using triangle inequality, we have

$$\begin{aligned}\|D^{-1}\hat{\Delta x}\|_2 =&\ \|D^{-1}(\hat{\Delta x} + \eta(x^0 - x^*)) - \eta D^{-1}(x^0 - x^*)\|_2 \\ \leq&\ \|D^{-1}(\hat{\Delta x} + \eta(x^0 - x^*))\|_2 + \eta\|D^{-1}(x^0 - x^*)\|_2. \qquad (4.80)\end{aligned}$$

Similarly, we also have

$$\begin{aligned}\|D\hat{\Delta s}\|_2 =&\ \|D(\hat{\Delta s} + \eta(s^0 - s^*)) - \eta D(s^0 - s^*)\|_2 \\ \leq&\ \|D(\hat{\Delta s} + \eta(s^0 - s^*))\|_2 + \eta\|D(s^0 - s^*)\|_2. \qquad (4.81)\end{aligned}$$

Combining eqns. (4.80) and (4.81), we have

$$\begin{aligned}
\max\ &\Big\{\|D^{-1}\hat{\Delta x}\|_2, \|D\hat{\Delta s}\|_2\Big\} &(4.82)\\
&\leq \max\Big\{\|D^{-1}(\hat{\Delta x}+\eta(x^0-x^*))\|_2, \|D(\hat{\Delta s}+\eta(s^0-s^*))\|_2\Big\}\\
&\qquad\qquad + \eta\left(\|D^{-1}(x^0-x^*)\|_2 + \|D(s^0-s^*)\|_2\right)\\
&\leq \|H(\sigma)\|_2 + \eta\left(\|D(s^0-s^*)\|_2 + \|D^{-1}(x^0-x^*)\|_2\right) + \sqrt{\frac{3}{2}}\,\|\tilde{f}^{(t)}\|_2\\
&\qquad\qquad + \eta\left(\|D^{-1}(x^0-x^*)\|_2 + \|D(s^0-s^*)\|_2\right)\\
&\leq \|H(\sigma)\|_2 + 2\eta\ \left(\|D(s^0-s^*)\|_2 + \|D^{-1}(x^0-x^*)\|_2\right) + \sqrt{3n}\cdot\frac{\gamma\sigma}{4\sqrt{n}}\sqrt{\mu}\,, &(4.83)
\end{aligned}$$

where the second inequality is due to eqn. (4.79) and the last inequality follows from Lemma (34).

Next, we bound $\|H(\sigma)\|_2$ as follows:

$$\begin{aligned}
\|H(\sigma)\|_2^2 =&\ \left\|-(XS)^{1/2}\mathbf{1}_n + \sigma\mu(XS)^{-1/2}\mathbf{1}_n\right\|_2^2\\
=&\ \sum_{i=1}^n x_i\,s_i + \sigma^2\mu^2\sum_{i=1}^n\frac{1}{x_i\,s_i} - 2n\sigma\mu\\
\leq&\ n\,\mu + \sigma^2\mu^2\sum_{i=1}^n\frac{1}{(1-\gamma)\mu} - 2n\sigma\mu = \left(1+\frac{\sigma^2}{1-\gamma}-2\sigma\right)n\,\mu\\
\Rightarrow\ \|H(\sigma)\|_2 \leq&\ \left(1+\frac{\sigma^2}{1-\gamma}-2\sigma\right)^{1/2}\sqrt{n\,\mu}\,, &(4.84)
\end{aligned}$$

where the inequality follows from the fact that $(x,y,s)\in\mathcal{N}(\gamma)$.

Now, using the definition of D and submultplicativity, we bound the following expression:

$$\begin{aligned}
&\left(\|D(s^0-s^*)\|_2 + \|D^{-1}(x^0-x^*)\|_2\right)\\
&\quad\leq\ \|(XS)^{-1/2}\|_2\left(\|X(s^0-s^*)\|_2 + \|S(x^0-x^*)\|_2\right)\\
&\quad\leq\ \left(\max_i\frac{1}{\sqrt{x_i\,s_i}}\right)\left(x^\mathsf{T}s^0 + s^\mathsf{T}x^0\right)\\
&\quad\leq\ \frac{1}{\sqrt{(1-\gamma)\,\mu}}\left(x^\mathsf{T}s^0 + s^\mathsf{T}x^0\right)\,, &(4.85)
\end{aligned}$$

where the second inequality follows from the facts that $\|S(x^0-x^*)\|_2\leq\ s^\mathsf{T}x^0$ and $\|X(s^0-s^*)\|_2\leq\ x^\mathsf{T}s^0$ (see (ii) and (iii) of Lemma 32) and the last inequality holds due to $(x,y,s)\in\mathcal{N}(\gamma)$.

Finally, combining eqns. (4.83), (4.84) and (4.85), we conclude

$$
\begin{aligned}
&\max\left\{\|D^{-1}\hat{\Delta x}\|_2, \|D\hat{\Delta s}\|_2\right\} \\
&\qquad \le \left(1+\frac{\sigma^2}{1-\gamma}-2\sigma\right)^{1/2}\sqrt{n\,\mu}+\frac{2\eta}{\sqrt{(1-\gamma)\,\mu}}\left(x^{\mathsf{T}}s^0+s^{\mathsf{T}}x^0\right)+\frac{\gamma\sigma}{4}\sqrt{3\,\mu} \\
&\qquad \le \left(1+\frac{\sigma^2}{1-\gamma}-2\sigma\right)^{1/2}\sqrt{n\,\mu}+\frac{6n\mu}{\sqrt{(1-\gamma)\,\mu}}+\frac{\gamma\sigma}{4}\sqrt{3\,\mu} \\
&\qquad = \left(1+\frac{\sigma^2}{1-\gamma}-2\sigma\right)^{1/2}\sqrt{n\mu}+\frac{6n}{\sqrt{(1-\gamma)}}\sqrt{\mu}+\frac{\gamma\sigma}{4}\sqrt{3\,\mu}\,,
\end{aligned}
$$

where the last inequality follows from Lemma 32(i).

□

Lemma 39 (Theorem 2.6 of [137]). *Let initial point* (x^0, s^0, y^0) *satisfies* $(x^0, s^0) \ge (x^*, s^*)$ *for some* $(x^*, s^*, y^*) \in \mathcal{S}$. *Then, Algorithm 6 generates an iterate* (x^k, s^k, y^k) *satisfying* $\mu_k \le \epsilon\mu_0$ *and* $\|r^k\|_2 \le \epsilon\|r^0\|_2$ *within* $\mathcal{O}(n^2 \log\frac{1}{\epsilon})$ *iterations.*

Proof. Let $(\hat{\Delta x}^k, \hat{\Delta s}^k, \hat{\Delta y}^k)$ denote that the search direction, r^k be the residual and μ_k denotes the duality measure at the k-th iteration of Algorithm 6. Then, $(x^k, s^k, y^k) \in \Gamma$ and the using Lemma 35, it is easy to see that $r^k = \eta r^0$ for some $\eta \in (0, 1)$. Next, we bound $\|\hat{\Delta x}^k \circ \hat{\Delta s}^k\|_\infty$ as the following:

$$
\begin{aligned}
\|\hat{\Delta x}^k \circ \hat{\Delta s}^k\|_\infty &\le \|\hat{\Delta x}^k \circ \hat{\Delta s}^k\|_2 = \|(D^k)^{-1}\hat{\Delta x}^k \circ D^k\hat{\Delta s}^k\|_2 \\
&\le \|(D^k)^{-1}\hat{\Delta x}^k\|_2\,\|D^k\hat{\Delta s}^k\|_2 \\
&\le \max\left\{\|(D^k)^{-1}\hat{\Delta x}^k\|_2^2,\, \|D^k\hat{\Delta s}^k\|_2^2\right\}, \qquad (4.86)
\end{aligned}
$$

where the second inequality holds as $\|a \circ b\|_2 \le \|a\|_2 \cdot \|b\|_2$ for any two vectors a and b with same dimensions. This can be verified applying Cauchy-Schwartz inequality as follows:

$$
\begin{aligned}
\|a \circ b\|_2 = &\sqrt{\sum (a_i\, b_i)^2} = \sqrt{(a \circ a)^{\mathsf{T}}(b \circ b)} \\
\le &\sqrt{\|a \circ a\|_2 \cdot \|b \circ b\|_2} = \sqrt{\sqrt{\sum a_i^4} \cdot \sqrt{\sum b_i^4}} \\
\le &\sqrt{\sqrt{(\sum a_i^2)^2} \cdot \sqrt{(\sum b_i^2)^2}} = \sqrt{\sum a_i^2 \cdot \sum b_i^2} = \|a\|_2 \cdot \|b\|_2.
\end{aligned}
$$

Now, using Lemma 38 and from the right hand side of eqn. (4.86), we see that $\|\hat{\Delta x}^k \circ \hat{\Delta s}^k\|_\infty = \mathcal{O}(n^2)\mu_k$. From Lemma 37, we see that for some constant $\beta > 0$,

$$\mu_{k+1} \leq \left(1 - \frac{\beta}{n^2}\right)\mu_k, \forall k \geq 0. \tag{4.87}$$

Using the above inequality recursively with the fact that $\log(1+x) < x$ for all $x > -1$, we have $\mu_k \leq \epsilon\mu_0$ for some accuracy parameter $\epsilon > 0$. Then using this with the fact that $\|r^k\|_2/\|r^0\|_2 \leq \mu_k/\mu_0$ for all $k \geq 0$, the second conclusion of the Lemma follows. □

4.9.3 Proof of Theorem 27

Finally, our Theorem 27 follows from Lemma 34 and Lemma 39.

BIBLIOGRAPHY

[1] Dimitris Achlioptas, Themis Gouleakis, and Fotis Iliopoulos. "Local Computation Algorithms for the Lovász Local Lemma". In: *CoRR* abs/1809.07910 (2018). arXiv: 1809.07910. URL: http://arxiv.org/abs/1809.07910.

[2] Ajit Agrawal, Philip Klein, and R. Ravi. "When Trees Collide: An Approximation Algorithm for the Generalized Steiner Problem on Networks". In: *SIAM J. on Comp.* 24.3 (1995), pp. 440–456.

[3] Shipra Agrawal and Nikhil R Devanur. "Fast algorithms for online stochastic convex programming". In: *ACM-SIAM Symp. on Discrete Algo.* SIAM. 2014, pp. 1405–1424.

[4] Shipra Agrawal, Zizhuo Wang, and Yinyu Ye. "A Dynamic Near-Optimal Algorithm for Online Linear Programming". In: *Oper. Res.* 62.4 (2014), pp. 876–890.

[5] A. Ahmed et al. "Scalable inference in latent variable models". In: *Proc. of the 5th ACM WSDM.* 2012, pp. 123–132.

[6] Zeyuan Allen-Zhu and Lorenzo Orecchia. "Using Optimization to Break the Epsilon Barrier: A Faster and Simpler Width-independent Algorithm for Solving Positive Linear Programs in Parallel". In: *Proc. of SODA.* 2015, pp. 1439–1456.

[7] N. Alon et al. "Space-Efficient Local Computation Algorithms". In: *Proc. 22ndACM-SIAM Symposium on Discrete Algorithms (SODA).* 2012, pp. 1132–1139.

[8] Ganesh Ananthanarayanan et al. "GRASS: Trimming Stragglers in Approximation Analytics". In: *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14).* 2014.

[9] Erling D. Andersen and Knud. D. Andersen. *The Mosek Interior Point Optimizer for Linear Programming: An Implementation of the Homogeneous Algorithm.* Springer US, 2000, pp. 197–232.

[10] R. Andersen et al. "Local Computation of PageRank Contributions". In: *Internet Mathematics* 5(1–2) (2008), pp. 23–45.

[11] Haim Avron, Petar Maymounkov, and Sivan Toledo. "Blendenpik: Supercharging LAPACK's Least-Squares Solver". In: *SIAM Journal on Scientific Computing* 32.3 (2010), pp. 1217–1236. DOI: 10.1137/090767911.

[12] Baruch Awerbuch and Rohit Khandekar. "Stateless distributed gradient descent for positive linear programs". In: *Proc. of STOC.* 2008, pp. 691–700.

[13] Owe Axelsson and Vincent Allan Barker. *Finite element solution of boundary value problems: theory and computation*. Vol. 35. Siam, 1984.

[14] Y. Azar et al. “Online algorithms for covering and packing problems with convex objectives”. In: *Proc. of the IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*. 2016, pp. 148–157.

[15] A. Balakrishnan, T. L. Magnanti, and R. T. Wong. “A Dual-Ascent Procedure for Large-Scale Uncapacitated Network Design”. In: *Oper. Res.* 37.5 (1989), pp. 716–740.

[16] R Bar-Yehuda and S Even. “A linear-time approximation algorithm for the weighted vertex cover problem”. In: *J. of Algs.* 2.2 (1981), pp. 198–203.

[17] C. Barnhart et al. “Branch-and-price: Column generation for solving huge integer programs”. In: *Operations Research* 48.3 (2000), pp. 318–326.

[18] Cynthia Barnhart et al. *Handbook in Transportation Science. Airline crew scheduling*. Springer, 2002, pp. 517–560.

[19] Yair Bartal, John W. Byers, and Danny Raz. “Fast Distributed Approximation Algorithms for Positive Linear Programming with Applications to Flow Control”. In: *SIAM J. on Comp.* 33.6 (2004), pp. 1261–1279.

[20] Amrit Singh Bedi and Ketan Rajawat. “Asynchronous Incremental Stochastic Dual Descent Algorithm for Network Resource Allocation”. In: *IEEE Transactions on Signal Processing* 66 (2017), pp. 2229–2244.

[21] Diego Bello and German Riano. “Linear Programming solvers for Markov Decision Processes”. In: *Systems and Information Engineering Design Symposium*. Vol. 28. May 2006, pp. 90–95.

[22] J. F. Benders. “Partitioning procedures for solving mixed-variables programming problems”. In: *Numerische mathematik* 4.1 (1962), pp. 238–252.

[23] D. P. Bertsekas. *Nonlinear programming*. Athena Scientific, 1999.

[24] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, 1989.

[25] Dimitris Bertsimas and Rakesh Vohra. “Rounding algorithms for covering problems”. In: *Math. Prog.* 80.1 (1998), pp. 63–89.

[26] D. Bienstock and G. Iyengar. “Approximating Fractional Packings and Coverings in O(1/epsilon) Iterations”. In: *SIAM J. Comp.* 35.4 (2006), pp. 825–854.

[27] V. Blondel et al. “Convergence in multiagent coordination, consensus, and flocking”. In: *Proc. of IEEE Conference on Decision and Control*. 2005, pp. 2996–3000.

[28] Sem Borst, Varun Gupta, and Anwar Walid. "Distributed caching algorithms for content distribution networks". In: *Proceedings of IEEE INFOCOM*. 2010, pp. 1–9.

[29] Christos Boutsidis, Petros Drineas, and Malik Magdon-Ismail. "Near-Optimal Column-Based Matrix Reconstruction". In: *The SIAM Journal on Computing* 43.2 (2014), pp. 687–717.

[30] R Bouyouli et al. "New results on the convergence of the conjugate gradient method". In: *Numerical Linear Algebra with Applications* 16.3 (2009), pp. 223–236.

[31] S. Boyd et al. "Distributed optimization and statistical learning via the alternating direction method of multipliers". In: *Found. and Trends in Machine Learning* 3.1 (2011), pp. 1–122.

[32] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[33] Stephen Boyd et al. "Distributed optimization and statistical learning via the alternating direction method of multipliers". In: *Found. and Trends in M. L.* 3 (2011), pp. 1–122.

[34] N. Buchbinder and J. Naor. "The Design of Competitive Online Algorithms via a Primal-Dual Approach". In: *Foundations and Trends in Theoretical Computer Science* 3.2-3 (2009), pp. 93–263.

[35] Niv Buchbinder and Joseph Naor. "The Design of Competitive Online Algorithms via a Primal-Dual Approach". In: *Found. and Trends in Theoretical Computer Science* 3.2-3 (2009), pp. 93–263.

[36] M. Burger et al. "A distributed simplex algorithm for degenerate linear programs and multi-agent assignment". In: *Automatica* 48.9 (2012), pp. 2298–2304.

[37] John Byers and Gabriel Nasser. "Utility-based decision-making in wireless sensor networks". In: *Mobile and Ad Hoc Networking and Comp.* 2000, pp. 143–144.

[38] Emmanuel Candes and Yaniv Plan. "Tight oracle inequalities for low-rank matrix recovery from a minimal number of noisy random measurements". In: *IEEE Trans. Info. Theory* 57.4 (2011), pp. 2342–2359.

[39] Emmanuel Candes, Justin Romberg, and Terence Tao. "Robust Uncertainty Principles: Exact Signal Reconstruction from Highly Incomplete Frequency Information". In: *IEEE Trans. Inform. Theory* 52.2 (2006), pp. 489–509.

[40] Volkan Cevher, Stephen Becker, and Mark Schmidt. "Convex Optimization for Big Data: Scalable, randomized, and parallel algorithms for big data analytics". In: *IEEE Signal Proc. Mag.* 31.5 (2014), pp. 32–43.

[41] T. Chang et al. “Asynchronous Distributed ADMM for Large-Scale Optimization—Part I: Algorithm and Convergence Analysis”. In: *IEEE Transactions on Signal Processing* 64.12 (2016), pp. 3118–3130.

[42] N. Chatzipanagiotis, D. Dentcheva, and M. M. Zavlanos. “An Augmented Lagrangian Method for Distributed Optimization”. In: *Math. Program.* 152.1-2 (2015), pp. 405–434.

[43] Yudong Chen, Lili Su, and Jiaming Xu. “Distributed Statistical Machine Learning in Adversarial Settings: Byzantine Gradient Descent”. In: *Proc. ACM Meas. Anal. Comput. Syst. SIGMETRICS* 1.2 (2017).

[44] Mung Chiang et al. “Layering as optimization decomposition: A mathematical theory of network architectures”. In: *Proceedings of the IEEE* 95.1 (2007), pp. 255–312.

[45] Agniva Chowdhury, Jiasen Yang, and Petros Drineas. “An Iterative, Sketching-based Framework for Ridge Regression”. In: *Proceedings of the 35th International Conference on Machine Learning.* Vol. 80. 2018, pp. 988–997.

[46] Agniva Chowdhury et al. “Speeding up Linear Programming using Randomized Linear Algebra”. In: *arXiv:2003.08072* (2020).

[47] Kenneth L. Clarkson and David P. Woodruff. “Low Rank Approximation and Regression in Input Sparsity Time”. In: *Proceedings of the 45th annual ACM symposium on Theory of Computing.* 2013, pp. 81–90.

[48] Kenneth L Clarkson and David P Woodruff. “Low-rank approximation and regression in input sparsity time”. In: *Journal of the ACM (JACM)* 63.6 (2017), p. 54.

[49] Michael B Cohen. “Nearly tight oblivious subspace embeddings by trace inequalities”. In: *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms.* SIAM. 2016, pp. 278–287.

[50] Michael B. Cohen, Jelani Nelson, and David P. Woodruff. “Optimal Approximate Matrix Product in Terms of Stable Rank”. In: *43rd International Colloquium on Automata, Languages, and Programming.* 2016, 11:1–11:14.

[51] Michael B Cohen, Yin Tat Lee, and Zhao Song. “Solving linear programs in the current matrix multiplication time”. In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing.* ACM. 2019, pp. 938–942.

[52] P. Combettes and V. Wajs. “Signal recovery by proximal forward-backward splitting”. In: *Multiscale Modeling & Simulation* 4.4 (2005), pp. 1168–1200.

[53] C. Cortes and V. Vapnik. “Support-vector networks”. In: *Machine Learning* 20.3 (1995), pp. 273–297.

[54] Yiran Cui et al. “Implementation of Interior-point Methods for LP based on Krylov Subspace Iterative Solvers with Inner-iteration Preconditioning”. In: *arXiv preprint arXiv:1604.07491* (2016).

[55] Samuel I Daitch and Daniel A Spielman. “Faster approximate lossy generalized flow via interior point algorithms”. In: *Proceedings of the fortieth annual ACM symposium on Theory of computing*. ACM. 2008, pp. 451–460.

[56] G. Dantzig and P. Wolfe. “Decomposition principle for linear programs”. In: *Oper. Res.* 8.1 (1960), pp. 101–111.

[57] George Dantzig. *Linear programming and extensions*. Princeton university press, 2016.

[58] G. Desaulniers, J. Desrosiers, and M.M. Solomon. *Column Generation*. Springer, 2005.

[59] Nikhil R. Devanur and Thomas P. Hayes. “The adwords problem: online keyword matching with budgeted bidders under random permutations”. In: *Proc. of EC*. 2009, pp. 71–78.

[60] Steven Diamond and Stephen Boyd. “CVXPY: A Python-Embedded Modeling Language for Convex Optimization”. In: *J. of Machine Learning Res.* 17.83 (2016), pp. 1–5.

[61] Katerine Diaz-Chito, Aura Hernández-Sabaté, and Antonio M López. “A reduced feature set for driver head pose estimation”. In: *Applied Soft Computing* 45 (2016), pp. 98–107.

[62] Alexander Domahidi, Eric Chu, and Stephen Boyd. “ECOS: An SOCP solver for embedded systems”. In: *Euro. Control Conf.* 2013, pp. 3071–3076.

[63] David L. Donoho. “Compressed sensing”. In: *IEEE Trans. Inform. Theory* 52 (2006), pp. 1289–1306.

[64] David L. Donoho and Jared Tanner. “Sparse Nonnegative Solution of Underdetermined Linear Equations by Linear Programming”. In: *Proc. of the National Academy of Sciences of the USA*. 2005, pp. 9446–9451.

[65] Petros Drineas and Michael W Mahoney. “Lectures on Randomized Numerical Linear Algebra”. In: vol. 25. The Mathematics of Data, IAS/Park City Mathematics Series. 2018, pp. 1–45.

[66] Petros Drineas and Michael W. Mahoney. “RandNLA: Randomized Numerical Linear Algebra”. In: *Communications of the ACM* 59.6 (2016), pp. 80–90.

[67] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[68] R. Eghbali and M. Fazel. “Designing smoothing functions for improved worst-case competitive ratio in online optimization”. In: *Neural Information Processing Systems*. 2016, pp. 3287–3295.

[69] Reza Eghbali, Jon Swenson, and Maryam Fazel. “Exponentiated Subgradient Algorithm for Online Optimization under the Random Permutation Model”. In: *arXiv preprint arXiv:1410.7171* (2014).

[70] Donald Erlenkotter. “A Dual-Based Procedure for Uncapacitated Facility Location”. In: *Oper. Res.* 26.6 (1978), pp. 992–1009.

[71] T. Erseghe. “Distributed optimal power flow using ADMM”. In: *IEEE Trans. on Power Sys.* 29.5 (2014), pp. 2370–2380.

[72] Tomaso Erseghe et al. “Fast Consensus by the Alternating Direction Multipliers Method”. In: *IEEE Trans. on Sig. Proces.* 59 (2011), pp. 5523–5537.

[73] Ernie Esser, Xiaoqun Zhang, and Tony Chan. “A general framework for a class of first order primal-dual algorithms for convex optimization in imaging science”. In: *SIAM J. on Imaging Sciences* 3.4 (2010), pp. 1015–1046.

[74] H. III. Everett. “Generalized Lagrange multiplier method for solving problems of optimum allocation of resources”. In: *Operations research* 11.3 (1963), pp. 399–417.

[75] Uriel Feige, Boaz Patt-Shamir, and Shai Vardi. “On the Probe Complexity of Local Computation Algorithms”. In: *45th International Colloquium on Automata, Languages, and Programming, (ICALP)*. 2018, 50:1–50:14.

[76] Pedro A Forero, Alfonso Cano, and Georgios B. Giannakis. “Consensus-Based Distributed Support Vector Machines”. In: *J. Mach. Learn. Res.* 11 (2010), pp. 1663–1707.

[77] D. Gabay and B. Mercier. “A dual algorithm for the solution of nonlinear variational problems via finite element approximation”. In: *Computers & Mathematics with Applications* 2.1 (1976), pp. 17–40.

[78] D. Gabay and B. Mercier. “A dual algorithm for the solution of nonlinear variational problems via finite element approximations”. In: *Comp. and Math. with Appl.* 2.1 (1976), pp. 17–40.

[79] L. Gan, U. Topcu, and S. H. Low. “Optimal decentralized protocol for electric vehicle charging”. In: *IEEE Transactions on Power Systems* 28.2 (2013), pp. 940–951.

[80] Nicolas Gillis and Robert Luce. “Robust Near-Separable Nonnegative Matrix Factorization Using Linear Optimization”. In: *J. Mach. Learn. Res.* 15.1 (2014), pp. 1249–1280. ISSN: 1532-4435.

[81] P Gilmore and Ralph Gomory. "A linear programming approach to the cutting-stock problem". In: *Operations Research* 11.6 (1963), pp. 863–888.

[82] Michel X. Goemans and David P. Williamson. "A General Approximation Technique for Constrained Forest Problems". In: *SIAM J. on Comp.* 24.2 (1995), pp. 296–317.

[83] Tom Goldstein et al. "Unwrapping ADMM: Efficient Distributed Computing via Transpose Reduction". In: *AISTATS*. 2016, pp. 1151–1158.

[84] Gene H Golub and Charles F Van Loan. *Matrix computations*. Vol. 3. JHU press, 2012.

[85] J. Gonzalez et al. "Distributed parallel inference on large factor graphs". In: *Proc. of the 25th Conf. on UIAI*. 2009, pp. 203–212.

[86] B. Gopalakrishnan and E.L. Johnson. "Airline crew scheduling: state-of-the-art." In: *Annals of Operations Research* 140 (2005), pp. 305–337.

[87] C. Guestrin et al. "Distributed regression: an efficient framework for modeling sensor network data". In: *Proceedings of the 3rd ACM IPSN*. 2004, pp. 1–10.

[88] Yi Guo and Lynne E Parker. "A distributed and optimal motion planning approach for multiple mobile robots". In: *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*. Vol. 3. 2002, pp. 2612–2619.

[89] Anupam Gupta and Marco Molinaro. "How experts can solve LPs online". In: *Euro. Symp. on Algo.* Springer. 2014, pp. 517–529.

[90] Anupam Gupta and Marco Molinaro. "How the experts algorithm can help solve LPs online". In: *Math. of Oper. Res.* 41.4 (2016), pp. 1404–1431.

[91] Isabelle Guyon et al. "Result analysis of the NIPS 2003 feature selection challenge". In: *Advances in neural information processing systems*. 2005, pp. 545–552.

[92] N. Halko, P. Martinsson, and J. Tropp. "Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions". In: *SIAM Review* 53.2 (2011), pp. 217–288.

[93] T. Hazan, A. Man, and A. Shashua. "A parallel decomposition solver for svm: Distributed dual ascend using fenchel duality". In: *Proc. of CVPR*. 2008, pp. 1–8.

[94] Jianghai Hu, Yingying Xiao, and Ji Liu. "Distributed Algorithms for Solving Locally Coupled Optimization Problems on Agent Networks". In: *Decision and Control (CDC), 2007 IEEE Annual Conference on*. IEEE. 2018, pp. 2420–2425.

[95] T. Joachims. “Training Linear SVMs in Linear Time”. In: *Proceedings of the 12th ACM SIGKDD.* 2006, pp. 217–226.

[96] M. Johansson and M. Sternad. “Resource allocation under uncertainty using the maximum entropy principle”. In: *IEEE Trans. Info. Theory* 51.12 (2005), pp. 4103–4117.

[97] Narendra Karmarkar. “A new polynomial-time algorithm for linear programming”. In: *Proceedings of the sixteenth annual ACM symposium on Theory of computing.* ACM. 1984, pp. 302–311.

[98] J. Katz and L. Trevisan. “On the efficiency of local decoding procedures for error-correcting codes”. In: *Proc. 32nd Annual ACM Symposium on the Theory of Computing (STOC).* 2000, pp. 80–86.

[99] F. Kelly, A. Maulloo, and D. Tan. “Rate control for communication networks: shadow prices, proportional fairness and stability”. In: *J. of the Operational Research Society* 49.3 (1998), pp. 237–252.

[100] Thomas Kesselheim et al. “Primal beats dual on online packing LPs in the random-order model”. In: *ACM Symp. on Theory of Comp.* 2014, pp. 303–312.

[101] S. Khirirat, M. Johansson, and D. Alistarh. “Gradient compression for communication-limited convex optimization”. In: *2018 IEEE Conference on Decision and Control (CDC).* 2018, pp. 166–171.

[102] Jakub Konecny, Brendan McMahan, and Daniel Ramag. “Federated optimization: Distributed optimization beyond the datacenter”. In: *arXiv preprint arXiv:1511.03575* (2015).

[103] C. Koufogiannakis and N. E. Young. “Distributed algorithms for covering, packing and maximum weighted matching”. In: *Distributed Computing* 24.1 (2011), pp. 45–63.

[104] Yoshiaki Kuwata and Jonathan P How. “Cooperative distributed robust trajectory optimization using receding horizon MILP”. In: *IEEE Transactions on Control Systems Technology* 19.2 (2011), pp. 423–431.

[105] Leon S Lasdon. *Optimization theory for large systems.* Courier Corporation, 1970.

[106] Yin Tat Lee and Aaron Sidford. “Efficient inverse maintenance and faster algorithms for linear programming”. In: *2015 IEEE 56th Annual Symposium on Foundations of Computer Science.* IEEE. 2015, pp. 230–249.

[107] Yin Tat Lee and Aaron Sidford. “Path Finding I: Solving Linear Programs with $\tilde{O}\sqrt{rank}$ Linear System Solves”. In: *arXiv preprint arXiv:1312.6677* (2013).

[108] Yin Tat Lee and Aaron Sidford. "Path Finding II: An\~ O (m sqrt (n)) Algorithm for the Minimum Cost Flow Problem". In: *arXiv preprint arXiv:1312.6713* (2013).

[109] Yin Tat Lee and Aaron Sidford. "Path finding methods for linear programming: Solving linear programs in $O(rank)$ iterations and faster algorithms for maximum flow". In: *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*. IEEE. 2014, pp. 424–433.

[110] Yin Tat Lee and Aaron Sidford. "Solving Linear Programs with Sqrt (rank) Linear System Solves". In: *arXiv preprint arXiv:1910.08033* (2019).

[111] Jure Leskovec and Andrej Krevl. *SNAP Datasets: Stanford Large Network Dataset Collection*. http://snap.stanford.edu/data. June 2014.

[112] R. Levi, R. Rubinfeld, and A. Yodpinyanee. "Brief Announcement: Local Computation Algorithms for Graphs of Non-Constant Degrees". In: *Proc. of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, (SPAA)*. 2015, pp. 59–61.

[113] David D. Lewis et al. "RCV1: A New Benchmark Collection for Text Categorization Research". In: *J. Mach. Learn. Res.* 5 (Dec. 2004), pp. 361–397. ISSN: 1532-4435.

[114] Na Li, Lijun Chen, and Steven H Low. "Optimal demand response based on utility maximization in power networks". In: *IEEE Power and Energy Society General Meeting*. 2011, pp. 1–8.

[115] Y. Liao, H. Qi, and W. Li. "Load-balanced clustering algorithm with distributed self-organization for wireless sensor networks". In: *IEEE Sensors Journal* 13.5 (2013), pp. 1498–1506.

[116] Palma London, Shai Vardi, and Adam Wierman. "Logarithmic Communication for Distributed Optimization in Multi-Agent Systems". In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 3.3 (2019), pp. 1–29.

[117] Palma London et al. "A Parallelizable Acceleration Framework for Packing Linear Programs". In: *Association for the Advancement of Artificial Intelligence*. 2018, pp. 3706–3713.

[118] Palma London et al. "Distributed optimization via local computation algorithms". In: *ACM SIGMETRICS Performance Evaluation Review* 45.2 (2017), pp. 30–32.

[119] S. H. Low and D. E. Lapsley. "Optimization flow control. I. Basic algorithm and convergence". In: *IEEE/ACM Transactions on Networking* 7.6 (Dec. 1999), pp. 861–874. ISSN: 1063-6692. DOI: 10.1109/90.811451.

[120] Steven H Low, Fernando Paganini, and John C Doyle. “Internet congestion control”. In: *IEEE Control Systems* 22.1 (2002), pp. 28–43.

[121] Michael Luby and Noam Nisan. “A Parallel Approximation Algorithm for Positive Linear Programming”. In: *Proc. of STOC*. 1993, pp. 448–457.

[122] David G. Luenberger and Yinyu Ye. *Linear and Nonlinear Programming*. 3rd. Springer Publishing Company, Incorporated, 2008. ISBN: 3319188410.

[123] S. Magnússon et al. “Communication Complexity of Dual Decomposition Methods for Distributed Resource Allocation Optimization”. In: *IEEE Journal of Selected Topics in Signal Processing* 12.4 (2018), pp. 717–732.

[124] S. Magnússon et al. “Convergence of limited communication gradient methods.” In: *IEEE Journal of Selected Topics in Signal Processing* 63.5 (2018), pp. 1356–1371.

[125] Michael W. Mahoney. “Randomized Algorithms for Matrices and Data”. In: *Foundations and Trends in Machine Learning* 3.2 (2011), pp. 123–224.

[126] Y. Mansour et al. “Converting Online Algorithms to Local Computation Algorithms”. In: *Proc. of 39th Intl. Colloq. on Automata, Lang. and Prog. (ICALP)*. 2012, pp. 653–664.

[127] Yishay Mansour et al. “Converting Online Algorithms to Local Computation Algorithms”. In: *Proc. of ICALP*. 2012, pp. 653–664.

[128] Andrea De Martino and Daniele De Martino. “An introduction to the maximum entropy approach and its application to inference problems in biology”. In: *Heliyon* 4.4 (2018).

[129] L. Massoulié and J. Roberts. “Bandwidth sharing: objectives and algorithms”. In: *IEEE INFOCOM’99*. Vol. 3. 1999, pp. 1395–1403.

[130] Brendan McMahan and Daniel Ramage. *Federated learning: Collaborative machine learning without centralized training data*. https://research.googleblog.com/2017/04/federated-learning-collaborative.html. Accessed: 2017-04-10.

[131] Xiangrui Meng and Michael W Mahoney. “Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression”. In: *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. ACM. 2013, pp. 91–100.

[132] Xiangrui Meng, Michael a Saunders, and Michael W Mahoney. “LSRN: A parallel iterative solver for strongly over- or underdetermined systems”. In: *SIAM Journal on Scientific Computing* 36.2 (2014), pp. 95–118.

[133] Ofer Meshi and Amir Globerson. “An alternating direction method for dual MAP LP relaxation”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2011, pp. 470–483.

[134] K. Mohan et al. “Node-Based Learning of Multiple Gaussian Graphical Models”. In: *Journal of Machine Learning Research* 15 (2014), pp. 445–488.

[135] Marco Molinaro and R Ravi. “The geometry of online packing linear programs”. In: *Math. of Oper. Res.* 39.1 (2013), pp. 46–59.

[136] Renato DC Monteiro, Jerome W O’Neal, and Takashi Tsuchiya. “Uniform boundedness of a preconditioned normal matrix used in interior-point methods”. In: *SIAM Journal on Optimization* 15.1 (2004), pp. 96–100.

[137] Renato DC Monteiro and Jerome W O’Neal. “Convergence analysis of a long-step primal-dual infeasible interior-point lp algorithm based on iterative linear solvers”. In: *Georgia Institute of Technology* (2003).

[138] D. Mosk-Aoyama, T. Roughgarden, and D. Shah. “Fully distributed algorithms for convex optimization problems”. In: *SIAM J. on Opt.* 20.6 (2010), pp. 3260–3279.

[139] J.F.C. Mota et al. “D-ADMM: A communication-efficient distributed algorithm for separable optimization”. In: *IEEE Trans. on Sig. Proces.* 61.10 (2013), pp. 2718–2723.

[140] A. Nedic and A. Ozdaglar. “On the Rate of Convergence of Distributed Subgradient Methods for Multi-agent Optimization”. In: *Decision and Control (CDC), 2007 IEEE 46th Annual Conference on*. IEEE. 2007, pp. 4711–4716.

[141] A. Nedić and A. Ozdaglar. “Convergence rate for consensus with delays”. In: *Journal of Global Optimization* 47.3 (2010), pp. 437–456.

[142] A. Nedic, A. Ozdaglar, and P. A. Parrilo. “Constrained Consensus and Optimization in Multi-Agent Networks”. In: *IEEE Transactions on Automatic Control* 55.4 (2010).

[143] Angelia Nedic and Asuman Ozdaglar. “Distributed subgradient methods for multi-agent optimization”. In: *IEEE Trans. on Autom. Control* 54.1 (2009), pp. 48–61.

[144] Jelani Nelson and Huy L Nguyên. “OSNAP: Faster numerical linear algebra algorithms via sparser subspace embeddings”. In: *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. IEEE. 2013, pp. 117–126.

[145] George L. Nemhauser. “Column Generation for Linear and Integer Programming”. In: *Documenta Mathematica Extra Volume ISMP* (2012), pp. 65–73.

[146] Yurii Nesterov. “Smooth minimization of non-smooth functions”. In: *Math. Prog.* 103.1 (2005), pp. 127–152.

[147] Nam Ho-Nguyen and Fatma Kilinc-Karzan. “Online First-Order Framework for Robust Convex Optimization”. In: *Operations Research* 66.6 (2018), pp. 1670–1692.

[148] Feng Niu et al. “HOGWILD!: A Lock-free Approach to Parallelizing Stochastic Gradient Descent”. In: *Proceedings of the 24th International Conference on Neural Information Processing Systems*. NIPS’11. 2011.

[149] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[150] G. Notarstefano and F. Bullo. “Distributed abstract optimization via constraints consensus: Theory and applications”. In: *IEEE Trans. Autom. Control* 56.10 (2011), pp. 2247–2261.

[151] Brendan O’Donoghue et al. “Conic Optimization via Operator Splitting and Homogeneous Self-Dual Embedding”. In: *J. Opt. Theory Appl.* 169 (2016), pp. 1042–1068.

[152] R. Olfati-Saber. “Distributed Kalman filtering for sensor networks”. In: *Proc. of IEEE CDC*. 2007, pp. 5492–5498.

[153] Peter Orlik and Hiroaki Terao. *Arrangements of Hyperplanes*. Grundlehren der mathematischen Wissenschaften. Springer-Verlag Berlin Heidelberg, 1992.

[154] V. Padmanabhan et al. “Distributing streaming media content using cooperative networking”. In: *Proceedings of workshop on Network and operating systems support for digital audio and video*. ACM. 2002, pp. 177–186.

[155] Christopher C Paige and Michael A Saunders. “Solution of sparse indefinite systems of linear equations”. In: *SIAM journal on numerical analysis* 12.4 (1975), pp. 617–629.

[156] D. Palomar and M. Chiang. “Alternative distributed algorithms for network utility maximization: Framework and applications”. In: *IEEE Trans. on Autom. Control* 52.12 (2007), pp. 2254–2269.

[157] Xinghao Pan et al. “CYCLADES: Conflict-free Asynchronous Machine Learning”. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS’16. 2016.

[158] M. Paskin, C. Guestrin, and J. McFadden. "A robust architecture for distributed inference in sensor networks". In: *Proceedings of the 4th ACM IPSN*. 2005.

[159] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM Monographs on Discrete Mathematics and Applications, 2000.

[160] Q. Peng and S. Low. "Distributed optimal power flow algorithm for radial networks, I: Balanced single phase case". In: *IEEE Transactions on Smart Grid* (2016).

[161] Pierre Pesnea, Ruslan Sadykov, and François Vanderbeck. "Feasibility Pump Heuristics for Column Generation Approaches". In: *INRIA research report:Lecture Notes in Computer Science inria-00686255* 7276.6 (2012), pp. 332–343.

[162] Mert Pilanci and Martin J Wainwright. "Newton sketch: A near linear-time optimization algorithm with linear-quadratic convergence". In: *SIAM Journal on Optimization* 27.1 (2017), pp. 205–245.

[163] Serge A. Plotkin, David B. Shmoys, and Eva Tardos. "Fast Approximation Algorithms for Fractional Packing and Covering Problems". In: *Math. of Oper. Res.* 20.2 (1995), pp. 257–301.

[164] William H. Press, Saul A. Teukolsky, and Vetterling. "Numerical Recipes 3rd Edition: The Art of Scientific Computing". In: *The Oxford Handbook of Innovation*. 3rd ed. Cambridge University Press, 2007. Chap. 10.

[165] Robin L Raffard, Claire J Tomlin, and Stephen P Boyd. "Distributed optimization for cooperative agents: Application to formation flight". In: *Proc. of IEEE Conference on Decision and Control*. Vol. 3. 2004, pp. 2453–2459.

[166] P. Ravikumar, A. Agarwal, and M. J. Wainwright. "Message passing for graph-structured linear programs: Proximal methods and rounding schemes". In: *JMLR* 11.1 (2010), pp. 1043–1080.

[167] Benjamin Recht, Maryam Fazel, and Pablo A Parrilo. "Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization". In: *SIAM Review* 52.3 (2010), pp. 471–501.

[168] Ben Recht et al. "Factoring nonnegative matrices with linear programs". In: *Advances in Neural Information Processing Systems*. 2012, pp. 1214–1222.

[169] O. Reingold and S. Vardi. "New techniques and tighter bounds for local computation algorithms". In: *Journal of Computer and System Science* 82.7 (2016), pp. 1180–1200.

[170] Mauricio GC Resende and Geraldo Veiga. "An implementation of the dual affine scaling algorithm for minimum-cost flow on bipartite uncapacitated networks". In: *SIAM Journal on Optimization* 3.3 (1993), pp. 516–537.

[171] D. Richert and J. Cortés. "Robust distributed linear programming". In: *Trans. Autom. Control* 60.10 (2015), pp. 2567–2582.

[172] Carlos Riquelme, Ramesh Johari, and Baosen Zhang. "Online Active Linear Regression via Thresholding". In: *Proc. of Association for the Advancement of Artificial Intelligence*. 2017.

[173] R. T. Rockafellar. *Network Flows and Monotropic Optimization*. John Wiley and Sons, New York, 1984.

[174] R. Rubinfeld et al. "Fast Local Computation Algorithms". In: *Proc. 2nd Sym. on Innov. in Computer Science (ICS)*. 2011, pp. 223–238.

[175] Ruslan Sadykov and François Vanderbeck. "Column Generation for Extended Formulations". In: *EURO Journal on Computational Optimization* 1.1-2 (2013), pp. 81–115.

[176] M. Saks and C. Seshadhri. "Local Monotonicity Reconstruction". In: *SIAM J. on Comp.* 39.7 (2010), pp. 2897–2926.

[177] Pedram Samadi et al. "Optimal real-time pricing algorithm based on utility maximization for smart grid". In: *Proc. of IEEE Smart Grid Communications (SmartGridComm)*. 2010, pp. 415–420.

[178] S. Sanghavi, D. Malioutov, and A. S. Willsky. "Linear programming analysis of loopy belief propagation for weighted matching". In: *Proc. of NIPS*. 2008, pp. 1273–1280.

[179] I. Schizas, A. Ribeiro, and G. Giannakis. "Consensus in ad hoc WSNs with noisy links—Part I: Distributed estimation of deterministic signals". In: *IEEE Trans. on Signal Processing* 56.1 (2008), pp. 350–364.

[180] Shai Shalev-Shwartz et al. "Pegasos: primal estimated sub-gradient solver for SVM". In: *Mathematical Programming* 127.1 (2011), pp. 3–30.

[181] Yuanming Shi et al. "Large-scale convex optimization for dense wireless cooperative networks". In: *IEEE Trans. Signal Proc.* 63.18 (2015), pp. 4729–4743.

[182] N. Z. Shor. *Minimization methods for non-differentiable functions*. Vol. 3. Springer Science & Business Media, 2012.

[183] Daniel A Spielman and Shang-Hua Teng. "Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems". In: *SIAM Journal on Matrix Analysis and Applications* 35.3 (2014), pp. 835–885.

[184] Daniel A Spielman and Shang-Hua Teng. “Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems”. In: *Proceedings of the STOC*. Vol. 4. 2004.

[185] Srikrishna Sridhar et al. “An Approximate, Efficient LP Solver for LP Rounding”. In: *Proc. of NIPS*. 2013, pp. 2895–2903.

[186] Rayadurgam Srikant. *The mathematics of Internet congestion control*. Springer Science & Business Media, 2012.

[187] G. Steidl and T. Teuber. “Removing multiplicative noise by Douglas-Rachford splitting methods”. In: *Journal of Math. Imaging and Vision* 36.2 (2010), pp. 168–184.

[188] J. F. Sturm. “Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones”. In: *Optim. Methods Softw.* 1.1-4 (1999), pp. 625–653.

[189] I. Suzuki and M. Yamashita. “Distributed anonymous mobile robots: Formation of geometric patterns”. In: *SIAM Journal on Computing* 28.4 (1999), pp. 1347–1363.

[190] Ben Taskar, Vassil Chatalbashev, and Daphne Koller. “Learning Associative Markov Networks”. In: *Proc. of ICML*. 2004.

[191] H. Terelius, U. Topcu, and R. M. Murray. “Decentralized multi-agent optimization via dual decomposition”. In: *IEEE Transactions on Automatic Control* 44.1 (2011).

[192] *The CAIDA UCSD AS Relationship Dataset*. http://www.caida.org/data/as-relationships/. 2007.

[193] K.-C. Toh, M. J. Todd, and R. H. Tutuncu. “SDPT3 - a MATLAB software package for semidefinite programming, version 1.3”. In: *Optim. Methods Softw.* 11.545-4 (1999), pp. 625–581.

[194] Luca Trevisan. “Parallel Approximation Algorithms by Positive Linear Programming”. In: *Algorithmica* 21.1 (1998), pp. 72–88.

[195] J. N. Tsitsiklis, D. P. Bertsekas, and M. Athans. “Distributed asynchronous deterministic and stochastic gradient optimization algorithms”. In: *IEEE Trans. on Autom. Control* 31 (1986), pp. 803–812.

[196] AW van der Vaart and Jon Wellner. *Weak Convergence and Empirical Processes With Applications to Statistics*. Springer Series in Statistics. Springer-Verlag New York, 1996.

[197] Ky Vu, Pierre-Louis Poirion, and Leo Liberti. “Random Projections for Linear Programming”. In: *Mathematics of Operations Research* 43.4 (2018), pp. 1051–1071. DOI: 10.1287/moor.2017.0894. eprint: https://doi.org/10.1287/moor.2017.0894. URL: https://doi.org/10.1287/moor.2017.0894.

[198] Hansheng Wang, Guodong Li, and Guohua Jiang. “Robust Regression Shrinkage and Consistent Variable Selection Through the LAD-Lasso”. In: *Journal of Business and Economic Statistics* 25 (2007), pp. 347–355.

[199] E. Wei, A. Ozdaglar, and A. Jadbabaie. “A distributed Newton method for network utility maximization: Algorithm”. In: *IEEE Trans. on Autom. Control* 58.9 (2015), pp. 2162–2175.

[200] John N Weinstein et al. “The cancer genome atlas pan-cancer analysis project”. In: *Nature genetics* 45.10 (2013), pp. 1113–1120.

[201] David P. Woodruff. “Sketching as a Tool for Numerical Linear Algebra”. In: *Foundations and Trends in Theoretical Computer Science* 10.1-2 (2014).

[202] Stephen J Wright. *Primal-dual interior-point methods*. Vol. 54. Siam, 1997.

[203] Peng Xu et al. “Sub-sampled newton methods with non-uniform sampling”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 3000–3008.

[204] Junfeng Yang and Yin Zhang. “Alternating Direction Algorithms for ℓ_1-Problems in Compressive Sensing”. In: *SIAM Journal on Scientific Computing* 33.1 (2011), pp. 250–278.

[205] G. Yarmish and R. Slyke. “A distributed, scalable simplex method”. In: *J. of Supercomputing* 49.3 (2009), pp. 373–381.

[206] Y. Yi and M. Chiang. “Stochastic network utility maximization -— a tribute to Kelly’s paper published in this journal a decade ago”. In: *European Transactions on Telecommunications* 19.4 (2008), pp. 421–442.

[207] Neal E. Young. “Sequential and parallel algorithms for mixed packing and covering”. In: *Proc. of FOCS*. 2001, pp. 538–546.

[208] M. Yuan and Y. Lin. “Model selection and estimation in the Gaussian graphical model”. In: *Biometrika* 94.10 (2007a), pp. 19–35.

[209] Ming Yuan. “High dimensional inverse covariance matrix estimation via linear programming”. In: *Journal of Machine Learning Research* 11.Aug (2010), pp. 2261–2286.

[210] Ruiliang Zhang and James T. Kwok. “Asynchronous Distributed ADMM for Consensus Optimization”. In: *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*. ICML’14. 2014, pp. II-1701–II-1709.

[211] Yin Zhang. “On the convergence of a class of infeasible interior-point methods for the horizontal linear complementarity problem”. In: *SIAM Journal on Optimization* 4.1 (1994), pp. 208–227.

[212] Ji Zhu et al. “1-Norm Support Vector Machines”. In: *Proceedings of the 16th International Conference on Neural Information Processing Systems*. 2003, pp. 49–56.

[213] Edo Zurel and Noam Nisan. “An efficient approximate allocation algorithm for combinatorial auctions”. In: *Proc. of EC*. 2001.