The Dissertation Committee for Cho-Jui Hsieh
certifies that this is the approved version of the following dissertation:

Exploiting Structure in Large-scale Optimization for Machine Learning

Committee:

Inderjit S. Dhillon, Supervisor

Pradeep Ravikumar

Keshav Pingali

Ambuj Tewari

Stephen J. Wright

Exploiting Structure in Large-scale Optimization for Machine Learning

by

Cho-Jui Hsieh, B.S. EE; B.S. Math; M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2015

Acknowledgments

I would like to thank my supervisor Inderjit Dhillon for sharing his knowledge, pushing me to work hard and constantly try to improve my work, and giving me the freedom to explore a diverse set of projects. I would like to thank other machine learning faculty at the University of Texas at Austin, especially Pradeep Ravikumar, for the substantial collaboration and advising on my research. I gratefully acknowledge the members for my Ph.D. committee Keshav Pingali, Ambuj Tewari, and Stephen Wright for their time and valuable feedback on a preliminary version of this thesis. I would like to acknowledge my other co-authors for their help in letting me be more productive than I would have been able to on my own: Si Si, Hsiang-Fu Yu, Kai-Yang Chiang, S.V.N. Vishwanathan, Peder Olsen, Hyokun Yun, Matyas Sustik, Russell Poldrack, Stephen Becker, Arindam Banerjee, Huahua Wang, Ian En-Hsu Yen, Nagarajan, Natarajan, Ambuj Tewari, Mitul Tiwari, Sam Shah, Deepack Argawal, Chih-Jen Lin, Kai-Wei Chang, Guo-Xun Yuan, Yin-Wen Chang, Ron-En Fan, Sathiya Keerthi, S. Sundararajan, and Michael Ringgaard. The National Science Foundation and IBM provided funding for part of my work. Finally, and most importantly, I'd like to dedicate this thesis to my Fiancée Si Si, and my parents Shu-Miao Lin and Lin-Fen Hsieh.

Exploiting Structure in Large-scale Optimization for Machine Learning

Publication No. _____

Cho-Jui Hsieh, Ph.D.
The University of Texas at Austin, 2015

Supervisor: Inderjit S. Dhillon

With an immense growth of data, there is a great need for solving large-scale machine learning problems. Classical optimization algorithms usually cannot scale up due to huge amount of data and/or model parameters. In this thesis, we will show that the scalability issues can often be resolved by exploiting three types of structure in machine learning problems: problem structure, model structure, and data distribution. This central idea can be applied to many machine learning problems. In this thesis, we will describe in detail how to exploit structure for kernel classification and regression, matrix factorization for recommender systems, and structure learning for graphical models. We further provide comprehensive theoretical analysis for the proposed algorithms to show both local and global convergent rate for a family of in-exact first-order and second-order optimization methods.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Recently data is being generated at a tremendous rate in modern applications, including recommender systems, social network analysis, computer vision, and bio-informatics. As a result, there is a great need for developing scalable and efficient solvers for large-scale machine learning problems, where the input data size can be very large (big data), and the model can be very high dimensional (big models).

To solve large-scale machine learning problems, usually one cannot directly apply classical optimization solvers. For example, in nonlinear SVM problems, the size of the kernel matrix is growing quadratically with the number of samples, which usually leads to the computer running out of memory. In sparse inverse covariance estimation problems, the time complexity grows cubically with the number of random variables, so it is hard to solve problems with size larger than tens of thousands. For matrix factorization problems, the size of the input matrix can be very large when solving web-scale problems. For instance, the adjacency matrix of the LinkedIn social network has more than 300 million users. The above problems are extremely challenging if we directly apply classical techniques such as Newton methods or interior point

methods.

In this thesis, we demonstrate that it is very important to exploit the structure in large-scale optimization problems in order to develop fast and scalable algorithms. We consider the following three types of structure in machine learning problems:

1. **Problem Structure:** When facing large-scale machine learning problems, researchers and practitioners usually apply algorithms with simple update rules, such as stochastic gradient descent or coordinate descent methods. More advanced techniques, such as greedy coordinate descent or second order methods, albeit achieving much faster convergence speed, are rarely used in solving large-scale problems due to their high computational cost per iteration. In this thesis, we show that by carefully analyzing and exploiting the structure of objective functions, those optimization techniques can also be implemented very efficiently — in many applications they can have the same computational complexity as the gradient descent method. In particular, we show that the structure of Hessian matrix is very important for developing efficient optimization algorithms. Therefore we will analyze the structure of the Hessian matrix for several machine learning problems, and develop efficient algorithms by exploiting the structure.

2. **Model Structure:** Many modern machine learning applications lead to high dimensional optimization problems. To avoid over-fitting, a com-

mon way is to add a regularization term to enforce the solution to have a low intrinsic dimension (e.g., sparsity, low-rank, ...) In this thesis, we demonstrate several ways to speed up the optimization procedure by detecting the low intrinsic dimensional space of the solution and reducing the problem size by variable selection or elimination. We formally state the procedure for problems with decomposable-norm regularizers, which includes sparse, low-rank, or group sparse regularizations. We apply this technique to many machine learning applications, and formally provide convergence guarantee of the proposed algorithms.

3. **Data Distribution:** Real datasets usually have a local (clustering) structure. In classification problems, data points are usually generated non-uniformly from several hidden topics. For the problem of learning the relationship between stock prices, there are some natural grouping structure (e.g., industrial groups) of stock symbols, which lead to the structure that within-group correlations is usually stronger than between-group correlations.

   With the above observation, we develop a family of *divide-and-conquer algorithms* to explore data distribution for speeding up optimization algorithms. The main idea is to detect the clustering structure by a fast preprocessing step, and then divide the whole problem into several smaller subproblems. Each sub-problem can then be efficiently solved, and then combined together to give a solution to the original problem.

In Chapter 3, we use the sparse inverse covariance estimation problem as a running example to develop a series of fast and scalable optimization solvers by exploiting structures of problem, model, and data distribution. By analyzing the problem structure, we develop an efficient proximal Newton method for sparse inverse covariance estimation (QUIC), which achieves super-linear convergence rate and has the same time complexity as first order methods (proximal gradient or ADMM). By exploiting the model structure, we overcome the difficulty that the number of parameters quadratically grow with number of random variables, and significantly reduce the time complexity of the proximal Newton method. By exploiting the data structure, we combine the idea of divide-and-conquer and block-coordinate descent updates to arrive at the BigQUIC algorithm, which can solve problems with one million random variables in one day using a single machine.

In Chapter 4, we discuss more machine learning problems that can be efficiently solved by exploiting structure. In Section 4.1, we discuss another example, Nonnegative Matrix Factorization (NMF), where structure of the problem allows the "importance" of each parameter to be maintained efficiently. As a result, the *greedy coordinate descent algorithm* can be efficiently implemented. Kernel Support Vector Machine (kernel SVM) is another important machine learning problem where greedy coordinate descent can be efficiently applied. By further exploring the structure of data distribution, we develop an efficient Divide-and-Conquer solver for kernel SVM (DC-SVM) in Section 4.2. For many machine learning problems it is non-trivial to maintain

the importance efficiently, thus the importance of each variable has to be re-computed periodically during the optimization procedure. In Section 4.3 we focus on the class of *dirty statistical models* for high dimensional problems, where the objective function is composed with a loss function with a general super-position structured regularizations. We show that the Hessian matrix has a block structure, thus a block coordinate descent algorithm can be applied efficiently to compute the Newton direction. We further propose an *active subspace selection* approach for selecting an important subspace of the model for any decomposable norm regularizations. There are many applications of the proposed method, including matrix completion, principal component analysis, graphical model structure learning, and multi-task learning problems.

In Chapter 5, we conclude the proposal by generalizing our algorithms to some general machine learning problems. In Section 5.1, we discuss the structure of problem for two machine learning problems: Empirical Risk Minimization (ERM) and matrix optimization with simple matrix functions (such as covariance selection, matrix completion, PCA, ...). We show the structure can be explored by applying in-exact proximal Newton method. In Section 5.2, we summarized two ways to exploit model structure: coordinate descent with priority and active subspace selection. In Section 5.3, we discuss a divide-and-conquer parallel proximal Newton method for any twice differentiable functions. The algorithm is based on partition the variables into blocks, and each processor focuses on updating its own block of variables. By exploiting the data distribution, we can obtain a better partition than random, and as a

result the algorithm converges to the optimal solution quickly.

Finally, in Chapter 6 we show the theoretical gaurantee for the techniques discussed in this thesis. We discuss a general framework for in-exact proximal gradient and Newton methods, and show the local and global convergence rate. Using this framework, we can prove the convergence rate for in-exact proximal Newton method, active variable selection, and parallel divide-and-conquer proximal Newton method.

Chapter 2

Structure in Machine Learning Problems

We focus on large-scale machine learning problems, where there is a large volume of data or the problem to be solved is very high dimensional. We tackle these challenges in the context of the Regularized Risk Minimization (RLM) problem for machine learning—the model is estimated by solving the following optimization problem:

$$\min_{\boldsymbol{\theta}} g(\boldsymbol{\theta}, X) + h(\boldsymbol{\theta}) \equiv f(\boldsymbol{\theta}), \tag{2.1}$$

where g is the loss function measuring the goodness of the model $\boldsymbol{\theta}$ based on training data X, and h is the regularization term to impose prior knowledge of the model. This framework covers a large number of modern machine learning problems such as SVM, logistic regression, matrix completion, and most of the statistical estimators. There are many classical solvers that can be applied to solve (2.1); however, directly applying those general solvers usually yields very poor performance on large-scale machine learning problems. In order to develop an efficient algorithm, we have to carefully study the structure of problem, model and data distribution for machine learning problems, and speed up the optimization algorithms by exploiting the structure.

2.1 Structure of the Problem

The efficiency of an optimization method depends on two factors – the convergence rate and the computational complexity (per iteration). Developing an efficient solver usually leads to the trade-off problem between these two factors. Most of the first order methods, including coordinate descent or gradient descent, have slower convergence rate but very quick updates. On the other hand, second order methods often have a much faster (quadratic) convergence rate while the computational complexity usually grows quadratically with dimensionality.

In this thesis, the problem structure indicates the structure of the loss function in the regularized risk minimization framework (2.1). The problem structure has been used in developing scalable optimization methods in literature, but most of the previous work focuses on *first order methods*. For example, in empirical risk minimization problems, the loss function $g(\boldsymbol{w}, X)$ can be written as the summation of individual loss components defined on data points:

$$g(\boldsymbol{\theta}, X) = \sum_{j=1}^{n} \ell(h(\boldsymbol{x}_i; \boldsymbol{\theta}), y_i), \tag{2.2}$$

where each $\boldsymbol{x}_i$ is a training sample, y_i is the corresponding target, $h(\boldsymbol{x}_i; \boldsymbol{\theta})$ is the prediction function, and $\ell(\hat{y}, y)$ is a non-negative real-valued loss function which measures how different the prediction $\hat{y}$ is from the true outcome y. The structure of (2.2) suggests that the gradient can be written as the summation of n individual terms. As a result, first order methods including gradient descent

or stochastic gradient descent can be efficiently applied. Another interesting example is the ADMM algorithm [5], where they focus on problems with the structure of $\sum_{i=1}^{k} v_i(\boldsymbol{\theta}) + r(\boldsymbol{\theta})$. When each subproblem $v_i(\cdot)$ can be easily solved, the ADMM algorithm can be applied to solve the combined problem.

In this thesis, We develop several efficient algorithms, including second order methods and greedy coordinate descent, by exploiting the structure of the Hessian matrix. In Section 3.2, we show that the Hessian matrix for the inverse covariance matrix estimation problem can be written as the Kronecker product of two matrices, and an efficient proximal Newton method can be developed based on exploiting this structure. We will also discuss the structure of other problems in Section 4. In Section 5.1, we will discuss the Hessian structure of a broad class of linear models and matrix functions.

2.2 Structure of the Model

Modern machine learning applications usually lead to high-dimensional problems. Statistically, there has been considerable research addressing the sample complexity problem. In many cases, it has been shown that if the underlying parameters have a much lower intrinsic dimensionality, the problem using the appropriate regularizer can recover the solution using samples that scale with the intrinsic dimensionality. For example, the low rank regularization for matrix completion in [6], and the Lasso regularization for recovering sparse models [72].

However, in terms of computational complexity, there is limited work

on speeding up algorithms by exploring the low intrinsic dimensionality of the solution. In many cases, due to the difficulty of dealing with non-smooth regularizations, the resulting problem is usually considered harder to solve, and most existing work focused on applying first order methods including proximal gradient descent or coordinate descent methods. In this thesis, we want to tackle the following problem: *can we utilize the low intrinsic dimensionality of the solution to develop efficient algorithms that scale with the intrinsic dimensionality?*

We develop a family of algorithms that scale with the intrinsic dimensionality of the model. The main idea is to detect the low intrinsic dimensionality of solutions in the early stage of optimization procedure, which helps to significantly reduce the problem size. In Section 3.3, we consider the sparse inverse covariance estimation problem, where the ℓ_1 regularization is used to promote the sparsity. We develop an effective *variable selection scheme*, where the time complexity of the proximal Newton method can be significantly reduced according to the sparsity of the solution. In Section 4.1, we show another example of Non-negative matrix factorization problems, where the "importance" of each variable can be efficiently maintained during the optimization process, thus the *greedy coordinate descent method* can be developed to explore the model sparsity structure. The same trick can be applied for solving the kernel SVM problem as discussed in Section 4.2. In Section 4.3, we will then generalize the *variable selection scheme* to the *active subspace selection scheme*, which can be applied to solve a general decomposable norm regular-

ization. Finally, in Section 5.2, we are going to discuss the general principal for exploiting the model structure.

2.3 Data Distribution

Real datasets usually have a local (clustering) structure. In stock datasets, stock prices within each industrial category are more correlated to each other; in document datasets, samples are usually generated from several hidden topics. We propose a family of divide-and-conquer algorithms to speed up optimization methods by exploiting the data distribution, in particular the clustering structure of data. In this framework, we only requires a very rough estimator of the local (clustering) distribution, which can be computed by different kinds of inexact clustering algorithms and will not add too much overhead to the overall procedure. Our divide-and-conquer algorithms has two steps. In the "divide" step, the large-scale problem is decomposed into several smaller sub-problems. Each sub-problem is defined only on a subset of data and can be efficiently solved. In the "conquer" step, solutions to the sub-problems are then combined to give a solution to the original problem.

We will discuss the divide-and-conquer algorithm for sparse inverse covariance estimation in Section 3.4, and further show the divide-and-conquer algorithm for solving the kernel SVM problem in Section 4.2. Finally we will develop a general divide-and-conquer proximal Newton framework for developing distributed optimization algorithms in Section 5.3.

Chapter 3

Exploiting Structure for Sparse Inverse Covariance Estimation

In this chapter, we use the sparse inverse covariance estimation problem as a running example to demonstrate the idea of exploiting structures of problem, model, and data distribution. Let $\mathbf{y}$ be a p-variate Gaussian random vector, with distribution $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$. Given n independently drawn samples $\{\mathbf{y_1}, \ldots, \mathbf{y_n}\}$ of this random vector, the sample covariance matrix can be written as

$$S = \frac{1}{n-1} \sum_{k=1}^{n} (\mathbf{y_k} - \hat{\boldsymbol{\mu}})(\mathbf{y_k} - \hat{\boldsymbol{\mu}})^T, \text{ where } \hat{\boldsymbol{\mu}} = \frac{1}{n} \sum_{k=1}^{n} \mathbf{y_k}. \tag{3.1}$$

Given a regularization penalty $\lambda > 0$, the ℓ_1-regularized Gaussian MLE for the inverse covariance matrix can be written as the solution of the following regularized *log-determinant* program:

$$\arg\min_{X \succ 0} \left\{ -\log\det X + \operatorname{tr}(SX) + \lambda \sum_{i,j=1}^{p} |X_{ij}| \right\}. \tag{3.2}$$

We will describe the proximal Newton method in Section 3.1, which is the second order method for composite functions. The direct implementation

[0]The material in this chapter has been published in [31, 27, 33, 32].

requires computing and computing p^2 by p^2 Hessian matrix, which is impractical for large-scale datasets. Therefore, we show in Section 3.2 that the time complexity of proximal Newton can be significantly reduced by exploiting the **problem structure**. We then show how to speed up the algorithm by a variable selection scheme that explores the **model structure** in Section 3.3. Finally we show how to scale the algorithm to ultra-high dimensional problems in Section 3.4 and 3.5 by exploiting the **data distribution**.

3.1 Background

Many problems in machine learning, signal processing, and bioinformatics can be formulated as the following composite function minimization problem:

$$\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = g(\boldsymbol{\theta}) + h(\boldsymbol{\theta}), \tag{3.3}$$

where g is a convex twice differentiable function, and h is a convex regularization function. Most of the RLM problems (2.1) with twice-differentiable loss functions fall in this framework. The sparse inverse covariance estimation problem (3.2) is a special case of (6.1).

Proximal Newton method is a second order iterative method to solve (6.1). Let $\boldsymbol{\theta}_t$ denotes the current solution, we build a quadratic approximation around $\boldsymbol{\theta}_t$ by the second-order Taylor expansion of the smooth component $g(\boldsymbol{\theta})$:

$$\bar{g}_{\boldsymbol{\theta}_t}(\boldsymbol{\Delta}) \equiv g(\boldsymbol{\theta}_t) + \nabla g(\boldsymbol{\theta}_t)^T \boldsymbol{\Delta} + \frac{1}{2} \boldsymbol{\Delta}^T \nabla^2 g(\boldsymbol{\theta}_t) \boldsymbol{\Delta}. \tag{3.4}$$

The Newton direction $\boldsymbol{d}_t^*$ for the entire objective can then be written as the solution of the regularized quadratic program:

$$\boldsymbol{d}_t^* = \arg\min_{\boldsymbol{\Delta}} \big\{ \bar{g}_{\boldsymbol{\theta}_t}(\boldsymbol{\Delta}) + h(\boldsymbol{\theta}_t + \boldsymbol{\Delta}) \big\}. \tag{3.5}$$

We use this Newton direction to compute a sequence of solutions $\{\boldsymbol{\theta}_t\}_{t=1}^{\infty}$ of the optimization problem (6.1). This variant of Newton method for such composite objectives is also referred to as a "proximal Newton-type method." We note that a key caveat to applying such second-order methods in high-dimensional settings is that the computation of the Newton direction appears to have a large time complexity. As a result, first-order methods have been so popular for minimizing the high-dimensional composite functions. However, we will show that many efficient solvers can be developed for solving (3.5) by exploiting the structure of the Hessian matrix $\nabla^2 g(\boldsymbol{\theta}_t)$.

Following the computation of the Newton direction $\boldsymbol{d}_t^*$, we need to find a step size $\alpha \in (0, 1]$ that ensures positive definiteness of the next iterate $\boldsymbol{\theta}_t + \alpha \boldsymbol{d}_t^*$ and leads to a sufficient decrease of the objective function. We adopt Armijo's rule [2, 79] and try step-sizes $\alpha \in \{\beta^0, \beta^1, \beta^2, \dots\}$ with a constant decrease rate $0 < \beta < 1$ (typically $\beta = 0.5$), until we find the smallest $k \in \mathbb{N}$ with $\alpha = \beta^k$ such that $\boldsymbol{\theta}_t + \alpha \boldsymbol{d}_t^*$ satisfies the following sufficient decrease condition:

$$f(\boldsymbol{\theta}_t + \alpha \boldsymbol{d}_t^*) \leq f(\boldsymbol{\theta}_t) + \alpha \sigma \delta_t, \quad \delta_t = \nabla g(\boldsymbol{\theta}_t)^T \boldsymbol{d}_t^* + h(\boldsymbol{\theta}_t + \boldsymbol{d}_t^*) - h(\boldsymbol{\theta}_t), \tag{3.6}$$

where $0 < \sigma < 0.5$. The basic version of proximal Newton method can be summarized in Algorithm 1.

Algorithm 1: Basic Proximal Newton Method

Input : Initial iterate $\boldsymbol{\theta}_0$.
Output: Sequence $\{\boldsymbol{\theta}_t\}$ that converges to the optimal solution.

for $t = 0, 1, \ldots$ **do**
- Form the second order approximation $\bar{f}_{\boldsymbol{\theta}_t}(\boldsymbol{\Delta}) := \bar{g}_{\boldsymbol{\theta}_t}(\boldsymbol{\Delta}) + h(\boldsymbol{\theta}_t + \boldsymbol{\Delta})$ to $f(\boldsymbol{\theta}_t + \boldsymbol{\Delta})$.
- Compute the Newton direction $\boldsymbol{d}_t^* = \arg\min_{\Delta} \bar{f}_{\boldsymbol{\theta}_t}(\boldsymbol{\theta}_t + \boldsymbol{\Delta})$
- Use an *Armijo*-rule based step-size selection to get α such that $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \boldsymbol{d}_t^*$ sufficiently decrease the objective function value (see (3.6)).

3.2 Exploiting Problem Structure—Fast Coordinate Descent Solver for Computing Newton Direction

Now we focus on solving the sparse inverse covariance estimation problem (3.2) by the second order method (Algorithm 1). In order to compute the Newton direction, we have to solve (3.5), which is a Lasso regression problem when $h(\cdot)$ is the ℓ_1 regularization. In [20], the authors show that coordinate descent methods are very efficient for solving the Lasso-typed problems. An obvious way to update each element of Δ (to solve (3.5)) requires $O(p^2)$ floating point operations since the Hessian matrix is a $p^2 \times p^2$ matrix, thus yielding an $O(p^4)$ procedure for computing the Newton direction. As we show below, our implementation reduces the cost of updating one variable to $O(p)$ by exploiting the structure of the Hessian matrix.

The gradient and Hessian for $g(X) = -\log\det X + \mathrm{tr}(SX)$ are (see, for instance, [4, Chapter A.4.3])

$$\nabla g(X) = S - X^{-1} \text{ and } \nabla^2 g(X) = X^{-1} \otimes X^{-1}. \tag{3.7}$$

In order to formulate our problem accordingly, we can verify that for a symmetric matrix Δ we have $\mathrm{tr}(X_t^{-1}\Delta X_t^{-1}\Delta) = \mathrm{vec}(\Delta)^T(X_t^{-1} \otimes X_t^{-1})\,\mathrm{vec}(\Delta)$, so that $\bar{g}_{X_t}(\Delta)$ in (3.5) can be rewritten as

$$\bar{g}_{X_t}(\Delta) = -\log\det X_t + \mathrm{tr}(SX_t) + \mathrm{tr}((S - W_t)^T\Delta) + \frac{1}{2}\,\mathrm{tr}(W_t\Delta W_t\Delta), \quad (3.8)$$

where $W_t = X_t^{-1}$.

For notational simplicity, we will omit the iteration index t in the derivations below where we only discuss a single Newton iteration (Hence, the notation for $\bar{g}_{X_t}$ is also simplified to $\bar{g}$.) Furthermore, we omit the use of a separate index for the coordinate descent updates. Thus, we simply use D to denote the current iterate approximating the Newton direction and use D' for the updated direction. Consider the coordinate descent update for the variable X_{ij}, with $i < j$ that preserves symmetry: $D' = D + \mu(\mathbf{e}_i\mathbf{e}_j^T + \mathbf{e}_j\mathbf{e}_i^T)$. The solution of the one-variable problem corresponding to (3.5) is:

$$\arg\min_{\mu} \quad \bar{g}(D + \mu(\mathbf{e}_i\mathbf{e}_j^T + \mathbf{e}_j\mathbf{e}_i^T)) + 2\lambda|X_{ij} + D_{ij} + \mu|. \quad (3.9)$$

We expand the terms appearing in the definition of $\bar{g}$ after substituting $D' = D + \mu(\mathbf{e}_i\mathbf{e}_j^T + \mathbf{e}_j\mathbf{e}_i^T)$ for Δ in (3.8) and omit the terms not dependent on μ. The quadratic term can be rewritten to yield:

$$\mathrm{tr}(WD'WD') = \mathrm{tr}(WDWD) \quad + \quad 4\mu\mathbf{w}_i^T D\mathbf{w}_j + 2\mu^2(W_{ij}^2 + W_{ii}W_{jj}), \quad (3.10)$$

where $\mathbf{w}_i$ refers to the i-th column of W. In order to compute the single variable update we seek the minimum of the following quadratic function of

μ:

$$\frac{1}{2}(W_{ij}^2 + W_{ii}W_{jj})\mu^2 + (S_{ij} - W_{ij} + \mathbf{w}_i^T D\mathbf{w}_j)\mu + \lambda|X_{ij} + D_{ij} + \mu|. \quad (3.11)$$

Letting $a = W_{ij}^2 + W_{ii}W_{jj}$, $b = S_{ij} - W_{ij} + \mathbf{w}_i^T D\mathbf{w}_j$, and $c = X_{ij} + D_{ij}$ the minimum is achieved for:

$$\mu = -c + \mathcal{S}(c - b/a, \lambda/a), \quad (3.12)$$

where

$$\mathcal{S}(z, r) = \text{sign}(z)\max\{|z| - r, 0\} \quad (3.13)$$

is the soft-thresholding function. Since a and c are easy to compute, the main computational cost arises while evaluating $\mathbf{w}_i^T D\mathbf{w}_j$, the third term contributing to coefficient b above. Direct computation requires $O(p^2)$ time. Instead, we maintain a $p \times p$ matrix $U = DW$, and then compute $\mathbf{w}_i^T D\mathbf{w}_j$ by $\mathbf{w}_i^T \mathbf{u}_j$ using $O(p)$ flops, where $\mathbf{u}_j$ is the j-th column of matrix U. In order to maintain the matrix U, we also need to update $2p$ elements, namely two coordinates of each $\mathbf{u}_k$ when D_{ij} is modified. We can compactly write the row updates of U as follows: $\mathbf{u}_{i\cdot} \leftarrow \mathbf{u}_{i\cdot} + \mu\mathbf{w}_{j\cdot}$ and $\mathbf{u}_{j\cdot} \leftarrow \mathbf{u}_{j\cdot} + \mu\mathbf{w}_{i\cdot}$, where $\mathbf{u}_{i\cdot}$ refers to the i-th *row* vector of U. There are totally $O(p^2)$ variables in X, thus the overall complexity for computing the Newton direction is $O(p^3)$.

3.3 Exploiting Model Structure—Fixed and Free Variable Selection

In this section, we are going to further reduce the time complexity by exploiting the *model structure*. Since ℓ_1 regularization is imposed in the

objective function (3.2), the final solution will be sparse. Our goal is to identify the nonzero pattern of X during the optimization steps, and as a result we can significant reduce the number of variables in the coordinate descent updates.

Specifically, we partition the variables into *free* and *fixed* sets based on the value of the gradient at the start of the outer loop that computes the Newton direction. We define the *free* set S_{free} and *fixed* set S_{fixed} as:

$$\begin{aligned} &X_{ij} \in S_{fixed} \ \text{ if } \ |\nabla_{ij} g(X)| \leq \lambda, \ \text{and } X_{ij} = 0, \\ &X_{ij} \in S_{free} \ \text{ otherwise.} \end{aligned} \tag{3.14}$$

Our definition of the *fixed* and *free* sets is clearly motivated by the minimum norm subgradient defined by

$$\text{grad}_{ij}^{S} f(X) = \begin{cases} \nabla_{ij} g(X) + \lambda & \text{if } X_{ij} > 0, \\ \nabla_{ij} g(X) - \lambda & \text{if } X_{ij} < 0, \\ \text{sign}(\nabla_{ij} g(X)) \max(|\nabla_{ij} g(X)| - \lambda, 0) & \text{if } X_{ij} = 0. \end{cases}$$

We can show that $\text{grad}^{S} f(X) = 0$ if and only if X is the global optimum. A variable X_{ij} belongs to the *fixed* set if and only if $X_{ij} = 0$ and $\text{grad}_{ij}^{S} f(X) = 0$. Therefore, we can show that for any X_t and corresponding fixed and free sets S_{fixed} and S_{free} as defined by (3.14), $\Delta^* = 0$ is the solution of the following optimization problem:

$$\arg\min_{\Delta} f(X_t + \Delta) \text{ such that } \Delta_{ij} = 0 \ \ \forall (i,j) \in S_{free}.$$

Based on the above property, if we perform block coordinate descent restricted to the fixed set, then no updates would occur. We then perform the coordinate

descent updates restricted to only the free set to find the Newton direction. Therefore the convergence of our method can be proved by formulated it as a block coordinate descent algorithm.

With this modification, the number of variables over which we perform the coordinate descent update (3.12) can be potentially reduced from p^2 to the number of non-zeros in X_t. But will the size of the free set be small? We initialize X_0 to a diagonal matrix, which is sparse. The following lemma shows that after a *finite* number of iterations, the iterates X_t will have a similar sparsity pattern as the limit X^*.

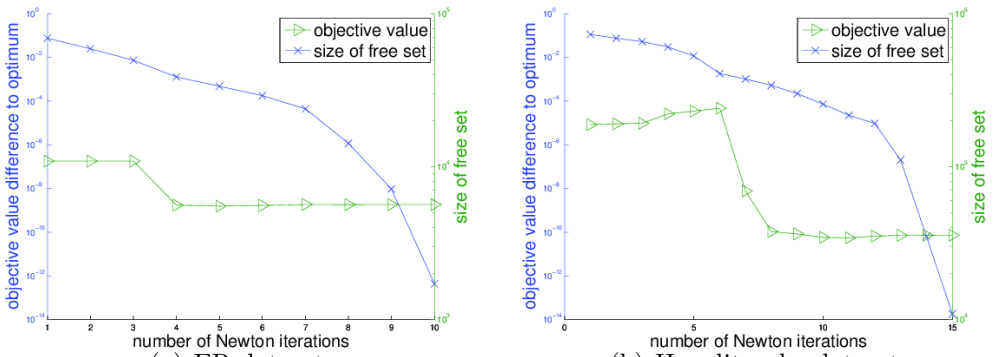
Lemma 1. *Assume that $\{X_t\}$ converges to X^*, the optimal solution of (3.2). If for some index pair (i, j), $|\nabla_{ij} g(X^*)| < \lambda$ (so that $X^*_{ij} = 0$), then there exists a constant $\bar{t} > 0$ such that for all $t > \bar{t}$, the iterates X_t satisfy*

$$|\nabla_{ij} g(X_t)| < \lambda \quad \textit{and} \ (X_t)_{ij} = 0. \tag{3.15}$$

Note that $|\nabla_{ij} g(X^*)| < \lambda$ implies $X^*_{ij} = 0$ from the optimality condition of (3.2). This theorem shows that after $\bar{t}$-th iteration we can ignore all the indexes that satisfies (3.15), and in practice we can use (3.15) as a criterion for identifying the *fixed* set.

To further demonstrate the power of *fixed/free* set selection, we use Hereditarybc dataset as an example. In Figure 3.1, we plot the size of the free set versus the number of Newton iterations. Starting from a total of $1869^2 = 3,493,161$ variables, the size of the free set progressively drops, in

fact to less than 120,000 in the very first iteration. We can see the super-linear convergence of QUIC even more clearly when we plot it against the number of iterations. A summary of the QUIC algorithm is presented in Algorithm 2.



(a) ER dataset (b) Hereditarybc dataset

Figure 3.1: Size of free sets and objective value versus iterations. For both datasets, the sizes of free sets are always less than $6\|X^*\|_0$ when running QUIC algorithm.

3.4 Exploiting Data Distribution—Divide-and-Conquer QUIC

In this section, we discuss a divide-and-conquer procedure for solving the sparse inverse covariance estimation problem by exploiting the **clustering structure of data distribution**. As we discussed in Section 3.2, solving this problem requires $O(p^3)$ computational time and $O(p^2)$ memory, so we aim to apply the following divide-and-conquer approach: in the divide step, we partition random variables into k clusters. Let $\mathcal{V} = \{1, \ldots, p\}$ denote the node set (random variables), given a partition $\{\mathcal{V}_c\}_{c=1}^k$ of $\mathcal{V}$, our divide and

Algorithm 2: QUadratic approximation for sparse Inverse Covariance estimation (QUIC overview)

Input : Empirical covariance matrix S (positive semi-definite, $p \times p$), regularization parameter matrix Λ, initial iterate $X_0 \succ 0$.

Output: Sequence $\{X_t\}$ that converges to $\arg\min_{X \succ 0} f(X)$, where $f(X) = g(X) + h(X)$, where $g(X) = -\log\det X + \mathrm{tr}(SX), h(X) = \|X\|_{1,\Lambda}$.

for $t = 0, 1, \ldots$ **do**

- Compute $W_t = X_t^{-1}$.
- Form the second order approximation $\bar{f}_{X_t}(\Delta) := \bar{g}_{X_t}(\Delta) + h(X_t + \Delta)$ to $f(X_t + \Delta)$.
- Partition the variables into free and fixed sets based on the gradient, see Section 3.3.
- Use coordinate descent to find the Newton direction $D_t^* = \arg\min_\Delta \bar{f}_{X_t}(X_t + \Delta)$ over the set of free variables, see (3.9) and (3.12) Section 3.2. (A *Lasso* problem.)
- Use an *Armijo*-rule based step-size selection to get α such that $X_{t+1} = X_t + \alpha D_t^*$ is positive definite and there is sufficient decrease in the objective function.

conquer algorithm first solves GMRF for all node partitions to get the inverse covariance matrices $\{X^{(c)}\}_{c=1}^k$, and then uses the following matrix

$$\bar{X} = \begin{bmatrix} X^{(1)} & 0 & \ldots & 0 \\ 0 & X^{(2)} & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & X^{(k)} \end{bmatrix}, \tag{3.16}$$

to initialize the solver for the whole GMRF. The skeleton of the divide and conquer framework is quite simple and is summarized in Algorithm 3.

If the partition is balanced, each subproblem only has $O(p/k)$ random variables, so only requires $O(p^2/k^2)$ space complexity and $O(p^3/k^3)$ time com-

plexity, which is significant faster than solving the global problem. In the conquer step, we gather all the results and use the combined solution to initialize the global solver. In order that Algorithm 3 be efficient, we require that $\bar{X}$ defined in (3.16) should be close to the optimal solution of the original problem X^*. In the following, we will derive a bound for $\|X^* - \bar{X}\|_F$. Based on this bound, we propose a spectral clustering algorithm to find an effective partitioning of the nodes.

Algorithm 3: Divide and Conquer method for Sparse Inverse Covariance Estimation

Input : Empirical covariance matrix S, scalar λ
Output: X^*, the solution of (3.2)

1 Obtain a partition of the nodes $\{\mathcal{V}_c\}_{c=1}^k$;
2 **for** $c = 1, \ldots, k$ **do**
3 Solve (4.17) on $S^{(c)}$ and subset of variables in $\mathcal{V}_c$ to get $X^{(c)}$;
4 Form $\bar{X}$ by $X^{(1)}, X^{(2)}, \ldots, X^{(k)}$ as in (3.16) ;
5 Use $\bar{X}$ as an initial point to solve the whole problem (3.2) ;

Bounding the distance between X^* and $\bar{X}$

Recently, [58] showed that when all the between cluster elements in S have absolute values smaller than λ, then X^* will have a block-diagonal structure and $X^* = \bar{X}$. However, in most real examples, a perfect partitioning does not exist. In the following we bound the distance between X^* and $\bar{X}$. Given the partition (clusters) $\{\mathcal{V}_c\}_{c=1}^k$, we define E as the following matrix:

$$E_{ij} = \begin{cases} 0 & \text{if } i, j \text{ are in the same cluster,} \\ \max(|S_{ij}| - \lambda, 0) & \text{otherwise.} \end{cases} \tag{3.17}$$

If $E = 0$, all the off-diagonal elements are below the threshold λ, so $X^* = \bar{X}$. In the following we consider a more interesting case where $E \neq 0$. In this case $\|E\|_F$ measures how much the off-diagonal elements exceed the threshold λ, and a good clustering algorithm should be able to find a partition to minimize $\|E\|_F$. In the following theorem we show that $\|X^* - \bar{X}\|_F$ can be bounded by $\|E\|_F$:

Theorem 1. *If there exists a* $\gamma > 0$ *such that* $\|E\|_2 \leq (1-\gamma)\frac{1}{\|\sigma_{min}(X^*)\|_2}$, *then*

$$\|X^* - \bar{X}\|_F \leq \frac{p \max(\sigma_{max}(\bar{X}), \sigma_{max}(X^*))^2 \sigma_{max}(\bar{X})}{\gamma \min(\sigma_{min}(X^*), \sigma_{min}(\bar{X}))} \|E\|_F,$$

where $\sigma_{min}(\cdot), \sigma_{max}(\cdot)$ *denote the minimum/maximum singular values.*

Clustering algorithm

In order to obtain computational savings, the clustering algorithm for the divide-and-conquer algorithm (Algorithm 3) should satisfy three conditions: (1) minimize the distance between the approximate and the true solution $\|\bar{X} - X^*\|_F$, (2) be cheap to compute, and (3) partition the nodes into balanced clusters.

To find a partition minimizing $\|E\|_F$, we want to find a partition $\{\mathcal{V}_c\}_{c=1}^k$ such that the sum of off-diagonal block entries of S^λ is minimized, where S^λ is defined as

$$(S^\lambda)_{ij} = \max(|S_{ij}| - \lambda, 0)^2 \ \forall \ i \neq j \ \text{ and } \ S^\lambda_{ij} = 0 \ \forall i = j. \tag{3.18}$$

At the same time, we want to have balanced clusters. Therefore, we minimize

the following normalized cut objective value [76]:

$$NCut(S^{\lambda}, \{\mathcal{V}_c\}_{c=1}^{k}) = \sum_{c=1}^{k} \frac{\sum_{i \in \mathcal{V}_c, j \notin \mathcal{V}_c} S_{ij}^{\lambda}}{d(\mathcal{V}_c)} \text{ where } d(\mathcal{V}_c) = \sum_{i \in \mathcal{V}_c} \sum_{j=1}^{p} S_{ij}^{\lambda}. \quad (3.19)$$

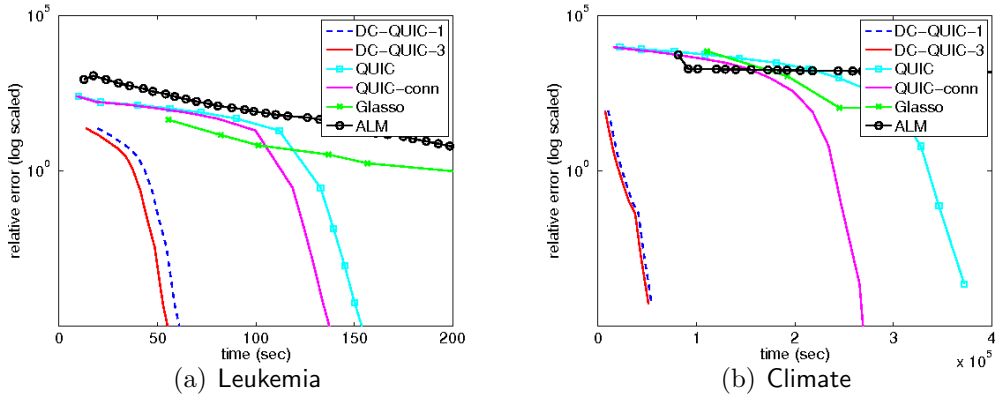
The time complexity of normalized cut on S^{λ} is mainly from computing the leading k eigenvectors of the Laplacian $D^{-1/2} S^{\lambda} D^{-1/2}$, which is at most $O(p^3)$. If S^{λ} is sparse, as is common in real situations, we could speed up the clustering phase by using the Graclus multilevel algorithm, which is a faster heuristic to minimize normalized cut [15].

We demonstrate that our algorithm outperforms other approaches in Figure 3.2. We use two datasets: the gene expression data Leukemia with $p = 1,255$ is provided by [51], and the Climate dataset with $p = 10,512$ generated from NCEP/NCAR Reanalysis data[1], with focus on the daily temperature at several grid points on earth. Figure 3.2 demonstrates that the Divide-and-Conquer algorithms – DC-QUIC-1 and DC-QUIC-3 (three levels of hierarchical clustering) are faster than other approaches.

3.5 Scaling Beyond Memory Capacity – BigQUIC

In the previous sections (Section 3.2, 3.3 and 3.4) we have developed a divide-and-conquer method for sparse inverse covariance estimation. However, the number of parameters in the optimization problem quadratically grows with number of random variables, so all the state-of-the-art methods cannot

[1] www.esrl.noaa.gov/psd/data/gridded/data.ncep.reanalysis.surface.html



(a) Leukemia (b) Climate

Figure 3.2: Comparison of algorithms on real datasets. The results show that DC-QUIC is much faster than other state-of-the-art solvers.

scale to problems with more than 20,000 nodes. In this section, we develop a new algorithm, BigQUIC, based on the proximal Newton method described in Section 3.2, and show that the algorithm can solve a one million dimensional problem in one day using a single machine. As we discussed below, one of the main building block of our algorithm is the block coordinate descent method, and we successfully speed it up by exploiting the clustering structure of the solutions X_t.

Again we want to solve the sparse inverse covariance estimation problem (3.2) . where the dimensionality p can be larger than 1 million. To begin, we list the difficulties of scaling QUIC (Algorithm 2) to million dimensional data:

1. **Difficulty in Approximating the Newton Direction.** In step 5 of Algorithm 2, we have to compute the Newton direction by solving the

optimization problem

$$D_t = \arg\min_D \{\bar{g}_{\Theta_t}(D) + h(X_t + D)\}, \tag{3.20}$$

where $\bar{g}_{\Theta_t}(\cdot)$ is the quadratic approximation of the smooth part. In QUIC, we apply a coordinate descent method to solve (3.20), where each coordinate update rule can be written as (3.12). The **key computational bottleneck** here is in computing the terms $\mathbf{w}_i^T D \mathbf{w}_j$, which take $O(p^2)$ time when implemented naively. To address this in QUIC (Section 3.2), we proposed to store and maintain $U = DW$, which reduced the cost to $O(p)$ flops per update. However, this is not a strategy we can use when dealing with very large data sets: storing the p by p dense matrices U and W in memory would be prohibitive. The straightforward approach is to compute (and recompute when necessary) the elements of W on demand, resulting in $O(p^2)$ time complexity.

2. **Difficulty in the Line Search Procedure.** After finding the generalized Newton direction D_t, QUIC then descends using this direction after a line-search via Armijo's rule. Specifically, it selects the largest step size $\alpha \in \{\beta^0, \beta^1, \dots\}$ such that $X + \alpha D_t$ is (a) positive definite, and (b) satisfies the following sufficient decrease condition (3.6). The **key computational bottleneck** is checking positive definiteness (typically by computing the smallest eigenvalue), and the computation of the determinant of a sparse matrix with dimension that can reach a million. The time and space complexity of classical sparse Cholesky decomposi-

tion generally grows quadratically to dimensionality even when fixing the number of nonzero elements in the matrix, so it is nontrivial to address this problem.

To address the two computational problems above, we propose a BigQUIC algorithm, where we develop an efficient **block coordinate descent** algorithm to solve the Newton direction subproblem using limited memory, and we also propose an efficient procedure for checking (a) and (b) in the line search procedure usinc Schur complement and sparse linear equation solving.

Block Coordinate Descent Method

The most expensive step during the coordinate descent update for D_{ij} is the computation of $\mathbf{w}_i^T D \mathbf{w}_j$, where w_i is the i-th column of $W = X^{-1}$; see (3.12). It is not possible to compute $W = X^{-1}$ with Cholesky factorization as was done in QUIC, nor can it be stored in memory. Note that $\mathbf{w}_i$ is the solution of the linear system $X\mathbf{w}_i = \mathbf{e}_i$. We thus use the conjugate gradient method (CG) to compute $\mathbf{w}_i$, leveraging the fact that X is a positive definite matrix. This solver requires only matrix vector products, which can be efficiently implemented for the sparse matrix X. CG has time complexity $O(mT)$, where T is the number of iterations required to achieve the desired accuracy, and $m = \|X\|_0$ is the number of nonzero elements in X. In the following analysis we use s to denote the current size of free set.

Vanilla Coordinate Descent

A single step of coordinate descent requires the solution of two linear systems $X\mathbf{w}_i = \mathbf{e}_i$ and $X\mathbf{w}_j = \mathbf{e}_j$ which yield the vectors $\mathbf{w}_i$, $\mathbf{w}_j$, and we can then compute $\mathbf{w}_i^T D\mathbf{w}_j$. The time complexity for each update would require $O(mT + s)$ operations, and the overall complexity will be $O(msT + s^2)$ for one full sweep through the entire matrix. Even when the matrix is sparse, the quadratic dependence on nonzero elements is expensive.

Our Approach: Block Coordinate Descent with memory cache scheme

In the following we present a block coordinate descent scheme that can accelerate the update procedure by storing and reusing more results of the intermediate computations. The resulting increased memory use and speedup is controlled by the number of blocks employed, that we denote by k.

Assume that only some columns of W are stored in memory. In order to update D_{ij}, we need both $\mathbf{w}_i$ and $\mathbf{w}_j$; if either one is not directly available, we have to recompute it by CG and we call this a "cache miss". A good update sequence can minimize the cache miss rate. While it is hard to find the optimal sequence in general, we successfully applied a *block by block* update sequence with a careful clustering scheme, where the number of cache misses is sufficiently small.

Assume we pick k such that we can store p/k columns of W (p^2/k elements) in memory. Suppose we are given a partition of $\mathcal{N} = \{1, \ldots, p\}$ into k blocks, $S_1, \ldots, S_k$. We divide matrix D into $k \times k$ blocks accordingly.

Within each block we run T_{inner} sweeps over variables within that block, and in the outer iteration we sweep through all the blocks T_{outer} times. We use the notation W_{S_q} to denote a p by $|S_q|$ matrix containing columns of W that corresponds to the subset S_q.

Coordinate descent within a block

To update the variables in the block (S_z, S_q) of D, we first compute W_{S_z} and W_{S_q} by CG and store it in memory, meaning that there is no cache miss during the within-block coordinate updates. With $U_{s_q} = DW_{S_q}$ maintained, the update for D_{ij} can be computed by $\mathbf{w}_i^T \mathbf{u}_j$ when $i \in S_z$ and $j \in S_q$. After updating each D_{ij} to $D_{ij} + \mu$, we can maintain U_{S_q} by

$$U_{it} \leftarrow U_{it} + \mu W_{jt}, \quad U_{jt} \leftarrow U_{jt} + \mu W_{it}, \quad \forall t \in S_q.$$

The above coordinate update computations cost only $O(p/k)$ operations because we only update a subset of the columns. Observe that U_{rt} never changes when $r \notin \{S_z \cup S_q\}$.

Therefore, we can use the following arrangement to further reduce the time complexity. Before running coordinate descent for the block we compute and store $P_{ij} = (\mathbf{w}_i)_{S_{\bar{z}\bar{q}}}^T (\mathbf{u}_j)_{S_{\bar{z}\bar{q}}}$ for all (i, j) in the free set of the current block, where $S_{\bar{z}\bar{q}} = \{i \mid i \notin S_z \text{ and } i \notin S_q\}$. The term $\mathbf{w}_i^T \mathbf{u}_j$ for updating D_{ij} can then be computed by $\mathbf{w}_i^T \mathbf{u}_j = P_{ij} + \mathbf{w}_{S_z}^T \mathbf{u}_{S_z} + \mathbf{w}_{S_q}^T \mathbf{u}_{S_q}$. With this trick, each coordinate descent step within the block only takes $O(p/k)$ time, and we only need to store U_{S_z,S_q}, which only requires $O(p^2/k^2)$ memory. Computing P_{ij}

takes $O(p)$ time for each i, j, so if we update each coordinate T_{inner} times within a block, the time complexity is $O(p + T_{\text{inner}}p/k)$ and the amortized cost per coordinate update is only $O(p/T_{\text{inner}} + p/k)$. This time complexity suggests that we should run more iterations within each block.

Sweeping through all the blocks

To go through all the blocks, each time we select a $z \in \{1, \ldots, k\}$ and updates blocks $(S_z, S_1), \ldots, (S_z, S_k)$. Since all of them share $\{\mathbf{w}_i \mid i \in S_z\}$, we first compute them and store in memory. When updating an off-diagonal block (S_z, S_q), if the free sets are dense, we need to compute and store $\{\mathbf{w}_i \mid i \in S_q\}$. So totally each block of W will be computed k times. The total time complexity becomes $O(kpmT)$, where m is number of nonzeros in X and T is number of conjugate gradient iterations. Assume the nonzeros in X is close to the size of free set ($m \approx s$), then each coordinate update costs $O(kpT)$ flops.

Selecting the blocks using clustering. We now show that a careful selection of the blocks using a clustering scheme can lead to dramatic speedup for block coordinate descent. When updating variables in the block (S_z, S_q), we would need the column $\mathbf{w}_j$ only if some variable in $\{D_{ij} \mid i \in S_z\}$ lies in the free set. Leveraging this key observation, given two partitions S_z and S_q, we define the set of *boundary nodes* as: $B(S_z, S_q) \equiv \{j \mid j \in S_q \text{ and } \exists i \in S_z \text{ s.t. } F_{ij} = 1\}$, where the matrix F is an indicator of the free set.

The number of columns to be computed in one sweep is then given by $p + \sum_{z \neq q} |B(S_z, S_q)|$. Therefore, we would like to find a partition $\{S_1, \ldots, S_k\}$

for which $\sum_{z \neq q} |B(S_z, S_q)|$ is minimal. It appears to be hard to find the partitioning that minimizes the number of boundary nodes. However, we note that the number in question is bounded by the number of cross cluster edges: $B(S_z, S_q) < \sum_{i \in S_z, j \in S_q} F_{ij}$. This suggests the use of graph clustering algorithms, such as METIS [38] or Graclus [15] which minimize the right hand side. Assuming that the ratio of between-cluster edges to the number of total edges is r, we observe a reduced time complexity of $O((p + rm)T)$ when computing elements of W, and r is very small in real datasets. In real datasets, when we converge to very sparse solutions, more than 95% of edges are in the diagonal blocks. In case of the fMRI dataset with $p = 228483$, we used 20 blocks, and the total number of boundary nodes were only $|B| = 8697$. Compared to block coordinate descent with random partition, which generally needs to compute 228483×20 columns, the clustering resulted in the computation of $228483 + 8697$ columns, thus achieved an almost 20 times speedup.

Line Search

The line search step requires an efficient and scalable procedure that computes $\log\det(A)$ and checks the positive definiteness of a sparse matrix A. We present a procedure that has complexity of at most $O(mpT)$ where T is the number of iterations used by the sparse linear solver. We note that computing $\log\det(A)$ for a large sparse matrix A for which we only have a matrix-vector multiplication subroutine available is an interesting subproblem on its own and we expect that numerous other applications may benefit from the approach

presented below. The following lemma can be proved by induction on p:

Lemma 2. *If* $A = \begin{pmatrix} a & \mathbf{b}^T \\ \mathbf{b} & C, \end{pmatrix}$ *is a partitioning of an arbitrary* $p \times p$ *matrix, where* a *is a scalar and* $\mathbf{b}$ *is a* $p-1$ *dimensional vector then* $\det(A) = \det(C)(a - \mathbf{b}^T C^{-1}\mathbf{b})$. *Moreover,* A *is positive definite if and only if* C *is positive definite and* $(a - \mathbf{b}^T C^{-1}\mathbf{b}) > 0$.

The above lemma allows us to compute the determinant by reducing it to solving linear systems; and also allows us to check positive-definiteness. Applying Lemma 2 recursively, we get

$$\log\det A = \sum_{i=1}^{p} \log(A_{ii} - A^T_{(i+1):p,i} A^{-1}_{(i+1):p,(i+1):p} A_{(i+1):p,i}), \tag{3.21}$$

where each $A_{i_1:i_2,j_1:j_2}$ denotes a submatrix of A with row indexes $i_1, \ldots, i_2$ and column indexes $j_1, \ldots, j_2$. Each $A^{-1}_{(i+1):p,(i+1):p} A_{(i+1):p,i}$ in the above formula can be computed as the solution of a linear system and hence we can avoid the storage of the (dense) inverse matrix. By Lemma 2, we can check the positive definiteness by verifying that all the terms in (3.21) are positive definite. Notice that we have to compute (3.21) in a reverse order ($i = p, \ldots, 1$) to avoid the case that $A_{(i+1):p,(i+1):p}$ is non positive definite.

Summary of the algorithm

We present BigQUIC as Algorithm 4. In summary, the time needed to compute the columns of W in block coordinate descent, $O((p+|B|)mTT_{\text{outer}})$, dominates the time complexity, which underscores the importance of minimizing the number of boundary nodes $|B|$ via our clustering scheme.

Algorithm 4: BigQUIC algorithm

Input : Samples Y, regularization parameter λ, initial iterate X_0

Output: Sequence $\{X_t\}$ that converges to X^*.

for $t = 0, 1, \ldots$ **do**
  Compute $W_t = X_t^{-1}$ column by column, partition the variables into free and fixed sets.
  Run graph clustering algorithm based on absolute values on free set.
  for $sweep = 1, \ldots, T_{outer}$ **do**
    for $s = 1, \ldots, k$ **do**
      Compute W_{S_s} by CG.
      for $q = 1, \ldots, k$ **do**
        Identify boundary nodes $B_{sq} := B(S_s, S_q) \subset S_q$ (only need if $s \neq q$)
        Compute $W_{B_{sq}}$ for boundary nodes (only need if $s \neq q$).
        Compute $U_{B_{sq}}$, and P_{ij} for all (i, j) the current block.
        Conduct coordinate updates for updating $D_{s,q}$.
  Use an Armijo-rule based step-size selection to get α such that $X_{t+1} = X_t + \alpha D_t$ is positive definite and there is sufficient decrease in the objective function.

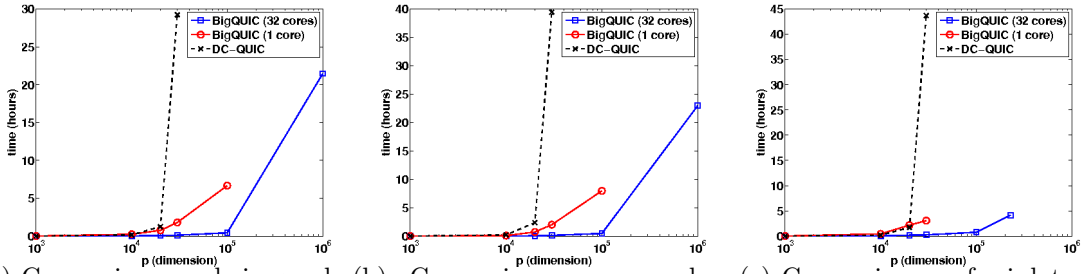
Experimental Results

Scalability of BigQUIC on high-dimensional datasets. In the first set of experiments, we show BigQUIC can scale to extremely high dimensional datasets. We conduct experiments on the following synthetic and real datasets:

(1) Chain graphs: the ground truth precision matrix is set to be $\Sigma^{-1}_{i,i-1} = -0.5$ and $\Sigma^{-1}_{i,i} = 1.25$.

(2) Graphs with random pattern: we use the procedure mentioned in Example 1 in [51] to generate random pattern. When generating the graph, we assume there are 500 clusters, and 90% of the edges are within clusters. We fix the average degree to be 10.
(3) FMRI data: The original dataset has dimensionality $p = 228,483$ and $n = 518$. For scalability experiments, we subsample various number of random variables from the whole dataset.

We use $\lambda = 0.5$ for chain and random Graph so that number of recovered edges is close to the ground truth, and set number of samples $n = 100$. We use $\lambda = 0.6$ for the fMRI dataset, which recovers a graph with average degree 20. We set the stopping condition to be $\text{grad}^S(X_t) < 0.01\|X_t\|_1$. In all of our experiments, number of nonzeros during the optimization phase do not exceed $5\|X^*\|_0$ in intermediate steps, therefore we can always store the sparse representation of X_t in memory. For BigQUIC, we set blocks k to be the smallest number such that p/k columns of W can fit into 32G memory. For both QUIC and BigQUIC, we apply the divide and conquer method proposed in [27] with 10-clusters to get a better initial point. The results are shown in Figure 3.3. We can see that BigQUIC can solve one million dimensional chain graphs and random graphs in one day, and handle the full fMRI dataset in about 5 hours. Finally, we show the output of BigQUIC on fMRI datasets in Figure 3.4. The results show that the output of our algorithm is consistent with some biological domain knowledge.



(a) Comparison on chain graph. (b) Comparison on random graph. (c) Comparison on fmri data.

Figure 3.3: The comparison of scalability on three types of graph structures. In all the experiments, BigQUIC can solve larger problems than QUIC even with a single core, and using 32 cores BigQUIC can solve million dimensional data in one day.

3.6 Summary of the Contribution

The QUIC algorithm mentioned in Section 3.1, 3.2 and 3.3 is published in [31, 32], and the code can be downloaded from `http://www.cs.utexas.edu/~sustik/QUIC/`. The DC-QUIC algorithm mentioned in Section 3.4 is published in [27]. The BigQUIC algorithm mentioned in Section 3.5 is published in [33], and the code can be downloaded from `http://www.cs.utexas.edu/~cjhsieh/BigQUIC-1.21.tgz`.

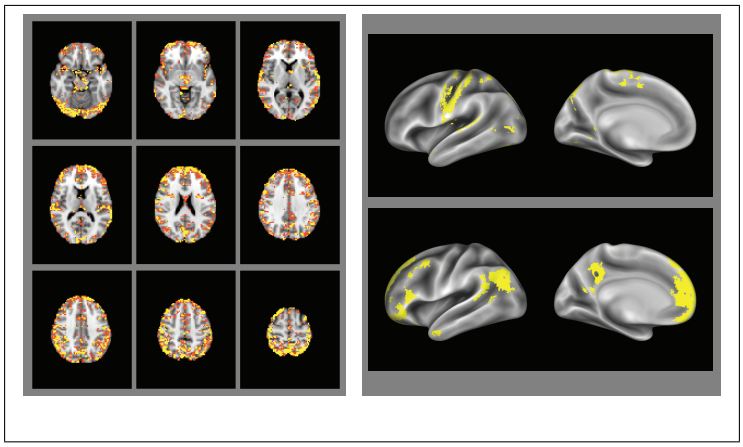


Figure 3.4: (Best viewed in color) Results from BigQUIC analyses of resting-state fMRI data. Left panel: Map of degree distribution across voxels, thresholded at degree=20. Regions showing high degree were generally found in the gray matter (as expected for truly connected functional regions), with very few high-degree voxels found in the white matter. Right panel: Left-hemisphere surface renderings of two network modules obtained through graph clustering. Top panel shows a sensorimotor network, bottom panel shows medial prefrontal, posterior cingulate, and lateral temporoparietal regions characteristic of the "default mode" generally observed during the resting state. Both of these are commonly observed in analyses of resting state fMRI data.

Chapter 4

Exploiting Structure for other Machine Learning Problems

In this section, we develop efficient algorithms for other problems by exploiting structure of problem, model, and data distribution.

4.1 Greedy Coordinate Descent for NMF

Non-negative matrix factorization (NMF) ([67, 48]) is a popular matrix decomposition method for finding non-negative representations of data. Given a matrix $V \in \mathbb{R}^{m \times n}$, $V \geq 0$, and a specified positive integer k, NMF seeks to approximate V by the product of two non-negative matrices $W \in \mathbb{R}^{m \times k}$ and $H \in \mathbb{R}^{k \times n}$, and usually $k \ll m, n$. Suppose each column of V is an input data vector, the main idea behind NMF is to approximate these input vectors by nonnegative linear combinations of nonnegative "basis" vectors (columns of W) with the coefficients stored in columns of H. The distance between V and WH can be measured by various distortion functions. The most commonly used one is the Frobenius norm, which leads to the following minimization

[0]The material in this chapter has been published in [26, 30, 29, 28].

problem:

$$\min_{W,H\geq 0} f(W,H) \equiv \frac{1}{2}\|V - WH\|_F^2 = \frac{1}{2}\sum_{i,j}(V_{ij} - (WH)_{ij})^2. \tag{4.1}$$

To achieve better sparsity, researchers ([25, 69]) have proposed adding regularization terms, on W and H, to (4.1). For example, an L1-norm penalty on W and H can achieve a more sparse solution:

$$\min_{W,H\geq 0} \frac{1}{2}\|V - WH\|_F^2 + \rho_1 \sum_{i,r} W_{ir} + \rho_2 \sum_{r,j} H_{rj}. \tag{4.2}$$

Many algorithms ([49, 23, 1, 90, 52, 43, 42]) have been proposed for this purpose. A cyclic coordinate descent method, called FastHals [12], has been proposed to solve the least squares NMF problem (4.1). Despite being a state-of-the-art method, FastHals has an inefficiency in that it uses a cyclic coordinate descent scheme and thus, may perform unneeded descent steps on unimportant variables. In this section we show that the greedy coordinate descent method can be applied to solve the NMF problem to focus on updating important variables, and the algorithm has the same time complexity to cyclic coordinate descent by exploring the problem structure.

4.1.1 Exploiting Problem Structure

The objective function (4.1) is not convex. However, when we fix one of the W, H and update the other, the problem will become convex. Therefore, most of existing algorithms fall into the alternating minimization framework, which switches between W and H in the outer iterations:

$$(W^0, H^0) \to (W^1, H^0) \to (W^1, H^1) \to \cdots \tag{4.3}$$

We will apply this alternating minimization scheme as well.

Now we analyze the problem structure. When we fix H and update W, the objective function of (4.1) is a quadratic problem with the following Hessian matrix:

$$\nabla^2_W f(W,H) = \begin{bmatrix} HH^T & 0 & \cdots & 0 \\ 0 & HH^T & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & HH^T \end{bmatrix},$$

where each HH^T is a k by k matrix. Since the Hessian is a block-diagonal matrix, each row-subproblem of W is independent, $\min_W f(W,H)$ is equivalent to solve the following m independent quadratic problems:

$$\begin{aligned} W_{i\cdot} &\leftarrow \underset{\boldsymbol{w}\geq 0}{\operatorname{argmin}} \frac{1}{2}\|V_{i\cdot}^T - H^T\boldsymbol{w}\|_2^2 \\ &= \underset{\boldsymbol{w}\geq 0}{\operatorname{argmin}} \frac{1}{2}\boldsymbol{w}^T HH^T\boldsymbol{w} - V_{i\cdot}H^T\boldsymbol{w} + \text{constant} \equiv g_i(\boldsymbol{w}). \end{aligned} \tag{4.4}$$

where we use $W_{i\cdot}$ to denote the i-th row of W. We can develop an efficient greedy coordinate descent algorithm based on this structure.

4.1.2 Exploiting Model Structure—Greedy Coordinate Descent

In the NMF problem, due to the bounded constraints $W \geq 0, H \geq 0$, there will be many zeros in W and H. The sparsity will be even increased when the ℓ_1 regularization is added in the objective function, such as in (4.2). Therefore, we want to develop algorithms for solving (4.4) that can focus on "important" variables. In Section 3.3, we compute the free/fixed sets in the beginning of each Newton iteration to conduct variable selection. This

approach has a drawback that the variable selection is only done periodically and cannot be updated on the fly. We will show that a greedy coordinate descent method can be applied to solve the NMF problem, which dynamically maintain the "importance" of each variable without having too much overhead.

Now we apply a coordinate descent algorithm for solving (4.1) for the i-th row. Assume $\bar{\boldsymbol{w}}$ is the current solution, then the current gradient has the form $\nabla g_i(\bar{\boldsymbol{w}}) = HH^T\bar{\boldsymbol{w}} - HV_{i\cdot}^T$, and the Hessian has the form $\nabla^2 g_i(\bar{\boldsymbol{w}}) = HH^T$. To update the r-th element of $\boldsymbol{w}$, the optimal update can be written as

$$w_i \leftarrow w_i + \delta \quad \text{where} \quad \delta = \max(0, w_i - \nabla_r g_i(\bar{\boldsymbol{w}})/(HH^T)_{rr}) - w_i, \tag{4.5}$$

and the change of objective function value is reduced by

$$g_i(\bar{\boldsymbol{w}}) - g_i(\bar{\boldsymbol{w}} + \delta \boldsymbol{e}_r) = \frac{\nabla_r g_i(\bar{\boldsymbol{w}})}{2(HH^T)_{rr}}.$$

Therefore, we can maintain two vectors $\boldsymbol{g}, \boldsymbol{d} \in \mathbb{R}^k$ when applying the greedy coordinate descent algorithm, where

$$g_r = \nabla_r g_i(\bar{\boldsymbol{w}}) \quad \text{and} \quad d_r = g_r/(HH^T)_{rr}.$$

Since $HH^T \in \mathbb{R}^{k\times k}$ is shared by all the subproblems, it is relatively cheap to pre-compute HH^T and store it in memory. The greedy coordinate descent update then has the following steps:

1. Choose $r = \text{argmax}_r\, d_r$.
2. Compute the update δ by (4.5).

3. Update $w_r \leftarrow w_r + \delta$.

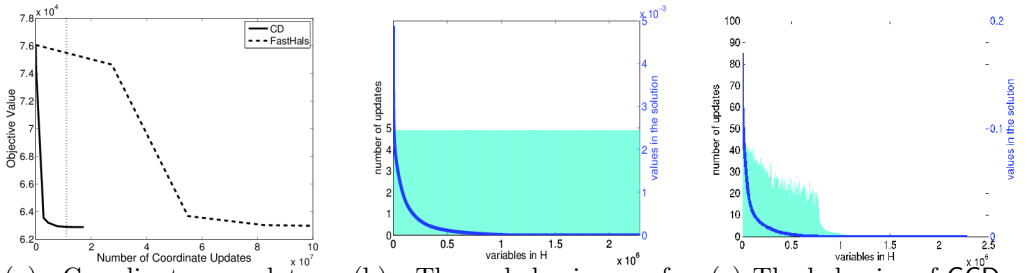
4. Update $g_s \leftarrow g_s + \delta(HH^T)_{s,r}$, $d_s \leftarrow g_s/(HH^T)_{ss}$ for all $s = 1, \ldots, k$.

Each greedy coordinate descent update only takes $O(k)$ time.

Note that FastHals applied a cyclic coordinate descent method to solve the NMF problem, which apply the update rule (4.5) cyclically. In our greedy coordinate descent algorithm, we can always select most important variable to update, and the time complexity is exactly the same with the cyclic coordinate descent method FastHals.

In Figures 4.1(b) and 4.1(c) the variables of the final solution H are listed on the X-axis — note that the solution is sparse as most of the variables are 0. Figure 4.1(b) shows the behavior of FastHals, which clearly shows that each variable is chosen uniformly. In contrast, as shown in Figure 4.1(c), by applying our new coordinate descent method, the number of updates for the variable is roughly proportional to their final values. For most variables with final value 0, our algorithm will never pick them to update. Therefore our new method focuses on nonzero variables and reduces the objective value more efficiently. Figure 4.1(a) shows that we can attain a 10-fold speedup by applying our variable selection scheme.

In the following we use $G \in \mathbb{R}^{m \times k}$ to denote the matrix where each row is its gradient $\boldsymbol{g}$, and $D \in \mathbb{R}^{m \times k}$ to denote the matrix where each row is the corresponding $\boldsymbol{d}$.



(a) Coordinate updates versus objective value (b) The behavior of FastHals (c) The behavior of GCD

Figure 4.1: Illustration of our variable selection scheme. Figure 4.1(a) shows that our method GCD reduces the objective value more quickly than FastHals. With the same number of coordinate updates (as specified by the vertical dotted line in Figure 4.1(a)), we further compare the distribution of their coordinate updates. In Figure 4.1(b) and 4.1(c), the X-axis is the variables of H listed by descending order of their final values. The solid line gives their final values, and the light blue bars indicate the number of times they are chosen. The figures indicate that FastHals updates all variables uniformly, while the number of updates for GCD is proportional to their final values, which helps GCD to converge faster.

Overall Greedy Coordinate Descent algorithm for NMF

We use the alternating minimization scheme: at each time we fix one of the W, H and update the other matrix. We run greedy coordinate descent algorithm for solving W (or H), and a stopping condition is needed for a sequence of updates. At the beginning of updates to W, we can store

$$p^{\text{init}} = \max_{i,j} D_{ij}^{W}. \tag{4.6}$$

Our algorithm then iteratively chooses variables to update until the following stopping condition is met:

$$\max_{i,j} D_{ij}^{W} < \epsilon p^{\text{init}}, \tag{4.7}$$

where ϵ is a small positive constant. Note that (4.7) will be satisfied in a finite number of iterations as $f(W, H)$ is lower bounded, and so the minimum for $f(W, H)$ with fixed H is achievable. A small ϵ value indicates each sub-problem is solved to high accuracy, while a larger ϵ value means our algorithm switches more often between W and H. We choose $\epsilon = 0.001$ in our experiments.

More interestingly, with this setting of stopping condition, we can prove that our algorithm GCD converges to a stationary point:

Theorem 2. *For least squares NMF, if a sequence* $\{(W_i, H_i)\}$ *is generated by* GCD*, then every limit point of this sequence is a stationary point.*

This convergence result holds for any inner stopping condition $\epsilon < 1$, thus it is different from the proof for exact methods, which assumes that each sub-problem is solved exactly. It is easy to extend the convergence result for GCD to regularized least squares NMF. Note that our proof is the first convergence gaurantee for alternative NMF solvers which does not assume subproblems ($\min_{W \geq 0} f(W, H)$ and $\min_{H \geq 0} f(W, H)$) are not solved exactly.

4.1.3 Experimental Comparisons

For least squares NMF, we compare GCD with three other state-of-the-art solvers:

1. ProjGrad: the projected gradient method in [52]. We use the MATLAB source code at `http://www.csie.ntu.edu.tw/~cjlin/nmf/`.

Table 4.1: The comparisons for least squares NMF solvers on dense datasets. For each method we present time/FLOPs (number of floating point operations) cost to achieve the specified relative error. The method with the shortest running time is boldfaced. The results indicate that GCD is most efficient both in time and FLOPs.

dataset	m	n	k	relative error	Time (in seconds)/FLOPs			
					GCD	FastHals	ProjGrad	BlockPivot
Synth03	500	1,000	10	10^{-4}	**0.6/0.7G**	2.3/2.9G	2.1/1.4G	1.7/1.1G
			30	10^{-4}	**4.0/5.0G**	9.3/16.1G	26.6/23.5G	12.4/8.7G
Synth08	500	1,000	10	10^{-4}	**0.21/0.11G**	0.43/0.38G	0.53/0.41G	0.56/0.35G
			30	10^{-4}	**0.43/0.46G**	0.77/1.71G	2.54/2.70G	2.86/1.43G
CBCL	361	2,429	49	0.0410	**2.3/2.3G**	4.0/10.2G	13.5/14.4G	10.6/8.1G
				0.0376	**8.9/8.8G**	18.0/46.8G	45.6/49.4G	30.9/29.8G
				0.0373	**14.6/14.5G**	29.0/75.7G	84.6/91.2G	51.5/53.8G
ORL	10,304	400	25	0.0365	**1.8/2.7G**	6.5/14.5G	9.0/9.1G	7.4/5.4G
				0.0335	**14.1/20.1G**	30.3/66.9G	98.6/67.7G	33.9/38.2G
				0.0332	**33.0/51.5G**	63.3/139.0G	256.8/193.5G	76.5/82.4G

2. BlockPivot: the block-pivot method in [44]. We use the MATLAB source code at `http://www.cc.gatech.edu/~hpark/nmfsoftware.php`.
3. FastHals: Cyclic coordinate descent method in [12]. We implemented the algorithm in MATLAB.

We test the performance on dense image datasets. We construct two synthetic datasets, Synth03 and Synth08, where the suffix numbers indicate 30% or 80% variables in the groundtruth W, H are zeros. We also test the algorithms on two image datasets CBCL and ORL. The results are summarized in Table 4.1. Table 4.1 compares the CPU time for each solver to achieve the specified relative error defined by $\|V - WH\|_F^2/\|V\|_F^2$, and we can conclude that GCD is two to three times faster than BlockPivot and FastHals on dense image data.

4.2 kernel Support Vector Machine

The support vector machine (SVM) [14] is probably the most widely used classifier in varied machine learning applications. For problems that are not linearly separable, kernel SVM uses a "kernel trick" to implicitly map samples from input space to a high-dimensional feature space, where samples become linearly separable. Due to its importance, optimization methods for kernel SVM have been widely studied [70, 37], and efficient libraries such as LIBSVM [10] and SVMLight [37] are well developed. However, the kernel SVM is still hard to scale up when the sample size reaches more than one million instances. The bottleneck stems from the high computational cost and memory requirements of computing and storing the kernel matrix, which in general is not sparse. Many previous exact or inexact solvers have been proposed to speed up the SVM training speed, including decomposition methods [66, 70], chunking and shrinking [68, 37], cascade SVM [24], kernel low rank approximations [85, 18, 17, 92, 91], random feature approaches [71, 46], AESVM [61], and online SVMs [3, 16].

Given a set of instance-label pairs $(\boldsymbol{x}_i, y_i), i = 1, \ldots, n, \boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \{1, -1\}$, the main task in training the kernel SVM is to solve the following quadratic optimization problem:

$$\min_{\boldsymbol{\alpha}} f(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} - \boldsymbol{e}^T \boldsymbol{\alpha}, \quad \text{s.t. } 0 \leq \boldsymbol{\alpha} \leq C, \tag{4.8}$$

where $\boldsymbol{e}$ is the vector of all ones; C is the balancing parameter between loss and regularization in the SVM primal problem; $\boldsymbol{\alpha} \in \mathbb{R}^n$ is the vector of dual

variables; and Q is an $n \times n$ matrix with $Q_{ij} = y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, where $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is the kernel function. Letting $\boldsymbol{\alpha}^*$ denote the optimal solution of (4.8), the decision value for a test data $\boldsymbol{x}$ can be computed by

$$\sum_{i=1}^{n} \alpha_i^* y_i K(\boldsymbol{x}, \boldsymbol{x}_i). \tag{4.9}$$

4.2.1 Exploiting Problem and Model Structure—Greedy Coordinate Descent Updates

The kernel SVM problem (4.8) is a quadratic minimization problem with bounded constraint, and the matrix Q is an n by n dense matrix. With the bounded constraint $\boldsymbol{\alpha} \geq 0$, the model usually has a *sparse structure* – there will be many zero elements in α, and the SVM prediction (4.9) is only related to the set of support vectors: $\{\alpha_i : \alpha_i > 0\}$. Therefore, we apply a greedy coordinate descent algorithm to solve the problem, which is able to identify the support vectors quickly and ignore non support vectors. This algorithm is originally proposed as the decomposition method [66, 70] for solving the kernel SVM problem with the bias term, where they update two variables at a time due to the additional constraint $\sum_i y_i \alpha_i = 0$ in the dual problem.

Similar to the case discussed in Section 4.1, by exploiting the problem structure, the greedy coordinate descent method has the same time complexity with cyclic or random coordinate descent algorithms. Assume α_i is updated by $\alpha_i \leftarrow \alpha_i + \delta_i$ at a coordinate descent step, the optimal δ_i has a closed form solution:

$$\delta_i \leftarrow \min(\max(\alpha_i - \boldsymbol{e}_i^T Q \boldsymbol{\alpha} / Q_{ii}, 0), C) - \alpha_i,$$

which requires $O(n)$ computation if the i-th row of Q is stored in memory. Now we analyze the time complexity for the following greedy coordinate descent method. During the optimization process, we maintain the gradient $\boldsymbol{g} = Q\boldsymbol{\alpha} - \boldsymbol{e}$ in memory, and δ_i can be computed by

$$\delta_i \leftarrow \min(\max(\alpha_i - g_i/Q_{ii}, 0), C) - \alpha_i,$$

which requires only $O(1)$ operation. We then need to maintain $\boldsymbol{g}$ by

$$\boldsymbol{g} \leftarrow \boldsymbol{g} + \delta_i(Q\boldsymbol{e}_i),$$

which requires $O(n)$ operations. Therefore the time complexity of greedy coordinate descent is *exactly the same* with the cyclic coordinate descent algorithm. We will use this algorithm as a base solver for the kernel SVM problem.

4.2.2 Exploring data distribution—Divide and Conquer kernel SVM

Next we develop a divide-and-conquer procedure for solving kernel SVM by exploiting the clustering structure of data points. Clustering structure appears in many real world applications. In classification problems, data points are usually generated non-uniformly from the input domain, and it is often the case that the sample points are dense in some areas, resulting in a local clustering structure.

We begin by describing the single-level version of our proposed algorithm. The main idea behind our divide and conquer SVM solver (DC-SVM) is to divide the data into smaller subsets, where each subset can be handled

efficiently and independently. The subproblem solutions are then used to initialize a coordinate descent solver for the whole problem. To do this, we first partition the dual variables into k subsets $\{\mathcal{V}_1, \ldots, \mathcal{V}_k\}$, and then solve the respective subproblems independently

$$\min_{\boldsymbol{\alpha}_{(c)}} \frac{1}{2}(\boldsymbol{\alpha}_{(c)})^T Q_{(c,c)} \boldsymbol{\alpha}_{(c)} - \boldsymbol{e}^T \boldsymbol{\alpha}_{(c)}, \text{ s.t. } 0 \leq \boldsymbol{\alpha}_{(c)} \leq C, \tag{4.10}$$

where $c = 1, \ldots, k$, $\boldsymbol{\alpha}_{(c)}$ denotes the subvector $\{\alpha_i \mid i \in \mathcal{V}_c\}$ and $Q_{(c,c)}$ is the submatrix of Q with row and column indexes $\mathcal{V}_c$.

The quadratic programming problem (4.8) has n variables, and takes at least $O(n^2)$ time to solve in practice (as shown in [59]). By dividing it into k subproblems (4.10) with equal sizes, the time complexity for solving the subproblems can be reduced to $O(k \cdot (\frac{n}{k})^2) = O(n^2/k)$. Moreover, the space requirement is also reduced from $O(n^2)$ to $O(n^2/k^2)$.

After computing all the subproblem solutions, we concatenate them to form an approximate solution for the whole problem $\bar{\boldsymbol{\alpha}} = [\bar{\boldsymbol{\alpha}}_{(1)}, \ldots, \bar{\boldsymbol{\alpha}}_{(k)}]$, where $\bar{\boldsymbol{\alpha}}_{(c)}$ is the optimal solution for the c-th subproblem. In the conquer step, $\bar{\boldsymbol{\alpha}}$ is used to initialize the solver for the whole problem. We show that this procedure achieves faster convergence since $\bar{\boldsymbol{\alpha}}$ is close to the optimal solution for the whole problem $\{\boldsymbol{\alpha}\}^*$.

Divide Step. We now discuss in detail how to divide problem (4.8) into subproblems. In order for our proposed method to be efficient, we require $\bar{\boldsymbol{\alpha}}$ to be close to the optimal solution of the original problem $\boldsymbol{\alpha}^*$. In the following,

we derive a bound on $\|\bar{\boldsymbol{\alpha}} - \boldsymbol{\alpha}^*\|_2$ by first showing that $\bar{\boldsymbol{\alpha}}$ is the optimal solution of (4.8) with an approximate kernel.

Lemma 3. *$\bar{\boldsymbol{\alpha}}$ is the optimal solution of* (4.8) *with kernel function $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ replaced by*

$$\bar{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = I(\pi(\boldsymbol{x}_i), \pi(\boldsymbol{x}_j))K(\boldsymbol{x}_i, \boldsymbol{x}_j), \tag{4.11}$$

where $\pi(\boldsymbol{x}_i)$ is the cluster that $\boldsymbol{x}_i$ belongs to; $I(a, b) = 1$ iff $a = b$ and $I(a, b) = 0$ otherwise.

Based on the above lemma, we are able to bound $\|\boldsymbol{\alpha}^* - \bar{\boldsymbol{\alpha}}\|_2$ by the sum of between-cluster kernel values:

Theorem 3. *Given data points $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ with labels $y_i \in \{1, -1\}$ and a partition indicator $\{\pi(\boldsymbol{x}_1), \ldots, \pi(\boldsymbol{x}_n)\}$,*

$$0 \leq f(\bar{\boldsymbol{\alpha}}) - f(\boldsymbol{\alpha}^*) \leq (1/2)C^2 D(\pi), \tag{4.12}$$

where $f(\boldsymbol{\alpha})$ is the objective function in (4.8) *and $D(\pi) = \sum_{i,j:\pi(\boldsymbol{x}_i)\neq\pi(\boldsymbol{x}_j)} |K(\boldsymbol{x}_i, \boldsymbol{x}_j)|$. Furthermore, $\|\boldsymbol{\alpha}^* - \bar{\boldsymbol{\alpha}}\|_2^2 \leq C^2 D(\pi)/\sigma_n$ where σ_n is the smallest eigenvalue of the kernel matrix.*

In order to minimize $\|\boldsymbol{\alpha}^* - \bar{\boldsymbol{\alpha}}\|$, we want to find a partition with small $D(\pi)$. Moreover, a balanced partition is preferred to achieve faster training speed. This can be done by the kernel kmeans algorithm, which aims to minimize the off-diagonal values of the kernel matrix with a balancing normalization. However, kernel kmeans has $O(n^2 d)$ time complexity, which is too

expensive for large-scale problems. Therefore we consider a simple two-step kernel kmeans approach as in [22]. The two-step kernel kmeans algorithm first runs kernel kmeans on m randomly sampled data points ($m \ll n$) to construct cluster centers in the kernel space. Based on these centers, each data point computes its distance to cluster centers and decides which cluster it belongs to. The algorithm has time complexity $O(nmd)$ and space complexity $O(m^2)$. In our implementation we just use random initialization for kernel kmeans, and observe good performance in practice.

Conquer Step. After computing $\bar{\boldsymbol{\alpha}}$ from the subproblems, we use $\bar{\boldsymbol{\alpha}}$ to initialize the solver for the whole problem. In principle, we can use any SVM solver in our divide and conquer framework, but we focus on using the coordinate descent method as in LIBSVM to solve the whole problem.

Divide and Conquer SVM with multiple levels. There is a trade-off in choosing the number of clusters k for a single-level DC-SVM with only one divide and conquer step. When k is small, the subproblems have similar sizes as the original problem, so we will not gain much speedup. On the other hand, when we increase k, time complexity for solving subproblems can be reduced, but the resulting $\bar{\boldsymbol{\alpha}}$ can be quite different from $\boldsymbol{\alpha}^*$ according to Theorem 3, so the conquer step will be slow. Therefore, we propose to run DC-SVM with multiple levels to further reduce the time for solving the subproblems, and meanwhile still obtain $\bar{\boldsymbol{\alpha}}$ values that are close to $\boldsymbol{\alpha}^*$.

In multilevel DC-SVM, at the l-th level, we partition the whole dataset

into k^l clusters $\{\mathcal{V}_1^{(l)}, \ldots, \mathcal{V}_{k^l}^{(l)}\}$, and solve those k^l subproblems independently to get $\bar{\boldsymbol{\alpha}}^{(l)}$. In order to solve each subproblem efficiently, we use the solutions from the lower level $\bar{\boldsymbol{\alpha}}^{(l+1)}$ to initialize the solver at the l-th level, so each level requires very few iterations. This allows us to use small values of k, for example, we use $k = 4$ for all the experiments.

Early prediction strategy. Computing the exact kernel SVM solution can be quite time consuming, so it is important to obtain a good model using limited time and memory. We now propose a way to efficiently predict the label of unknown instances using the lower-level models $\bar{\boldsymbol{\alpha}}^l$. We will see in the experiments that prediction using $\bar{\boldsymbol{\alpha}}^l$ from a lower level l already can achieve near-optimal testing performance.

When the l-th level solution $\bar{\boldsymbol{\alpha}}^l$ is computed, we propose the following early prediction strategy. From Lemma 3, $\bar{\boldsymbol{\alpha}}$ is the optimal solution to the SVM dual problem (4.8) on the whole dataset with the approximated kernel $\bar{K}$ defined in (4.11). Therefore, we propose to use the same kernel function $\bar{K}$ in the testing phase, which leads to the prediction

$$\sum_{c=1}^{k} \sum_{i \in \mathcal{V}_c} y_i \alpha_i \bar{K}(\boldsymbol{x}_i, \boldsymbol{x}) = \sum_{i \in \mathcal{V}_{\pi(\boldsymbol{x})}} y_i \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}), \tag{4.13}$$

where $\pi(\boldsymbol{x})$ can be computed by finding the nearest cluster center. Therefore, the testing procedure for early prediction is: (1) find the nearest cluster that x belongs to, and then (2) use the model trained by data within that cluster to compute the decision value.

Table 4.2: Comparison on real datasets using the RBF kernel.

	ijcnn1		cifar		census		covtype	
	$C = 32, \gamma = 2$		$C = 8, \gamma = 2^{-22}$		$C = 512, \gamma = 2^{-9}$		$c = 32, \gamma = 32$	
	time(s)	acc(%)	time(s)	acc(%)	time(s)	acc(%)	time(s)	acc(%)
DC-SVM (early)	12	98.35	**1977**	87.02	**261**	**94.9**	**672**	96.12
DC-SVM	41	**98.69**	16314	**89.50**	1051	94.2	11414	**96.15**
LIBSVM	115	**98.69**	42688	**89.50**	2920	94.2	83631	**96.15**
LIBSVM (subsapmle)	**6**	98.24	2410	85.71	641	93.2	5330	92.46
LaSVM	251	98.57	57204	88.19	3514	93.2	102603	94.39
CascadeSVM	17.1	98.08	6148	86.8	849	93.0	5600	89.51
LLSVM	38	98.23	9745	86.5	1212	92.8	4451	84.21
FastFood	87	95.95	3357	80.3	851	91.6	8550	80.1
SpSVM	20	94.92	21335	85.6	3121	90.4	15113	83.37
LTPU	248	96.64	17418	85.3	1695	92.0	11532	83.25
BudgetedSVM	24	96.88	5722	87.62	430	91.8	3839	87.83
AESVM	10	93.10	9519	87.83	335	93.61	3821	87.03

4.2.3 Experimental Results

We include the exact kernel SVM solvers (LIBSVM [10], CascadeSVM [24]), approximate SVM solvers (SpSVM [39], LLSVM [91], FastFood [46], LTPU [60], AESVM [61], BudgetedSVM [16]), and online SVM (LaSVM [3]) in our comparison. The results are shown in Table 4.2. Experimental results show that the early prediction approach in DC-SVM (stopped at the level with 64 clusters, denoted by DC-SVM(early)) achieves near-optimal test performance. By going to the top level (handling the whole problem), DC-SVM achieves better test performance but needs more time. Both DC-SVM and DC-SVM(early) are much faster than other approaches.

4.3 Proximal Newton method for Dirty Statistical Models

In this section, our goal is to extend the proximal Newton method with variable selection to handle the following broader class of problems. Consider a general superposition-structured parameter $\bar{\boldsymbol{\theta}} := \sum_{r=1}^{k} \boldsymbol{\theta}^{(r)}$, where $\{\boldsymbol{\theta}^{(r)}\}_{r=1}^{k}$ are the parameter-components, each with their own structure. Let $\{\mathcal{R}^{(r)}(\cdot)\}_{r=1}^{k}$ be regularization functions suited to the respective parameter components, and let $\mathcal{L}(\cdot)$ be a loss function that measures the goodness of fit of the superposition-structured parameter $\bar{\boldsymbol{\theta}}$ to the data. We consider a popular class of M-estimators studied in the papers above for these superposition-structured models:

$$\min_{\{\boldsymbol{\theta}^{(r)}\}_{r=1}^{k}} \mathcal{L}\left(\sum_{r} \boldsymbol{\theta}^{(r)}\right) + \sum_{r} \lambda_r \mathcal{R}^{(r)}(\boldsymbol{\theta}^{(r)}) := F(\boldsymbol{\theta}), \tag{4.14}$$

where $\{\lambda_r\}_{r=1}^{k}$ are regularization penalties. Note that in (4.14), the overall regularization contribution is separable in the individual parameter components, but the loss function term itself is not, and depends on the sum $\bar{\boldsymbol{\theta}} := \sum_{r=1}^{k} \boldsymbol{\theta}^{(r)}$. Throughout this section, we will use $\bar{\boldsymbol{\theta}} = \sum_{r=1}^{k} \boldsymbol{\theta}^{(r)}$ to denote the overall superposition-structured parameter, and $\boldsymbol{\theta} = [\boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(k)}]$ to denote the concatenation of all the parameters. This kind of estimators are used in many machine learning problems [83, 34], and most of the state-of-the-art solvers use proximal gradient descent approach or the ADMM method [5, 56, 73].

Decomposable norms. We consider the case where all the regularizers $\{\mathcal{R}^{(r)}\}_{r=1}^{k}$ are decomposable norms $\|\cdot\|_{\mathcal{A}_r}$. A norm $\|\cdot\|$ is decomposable at $\boldsymbol{x}$ if there is a subspace $\mathcal{T}$ and a vector $\boldsymbol{e} \in \mathcal{T}$ such that the sub differential at $\boldsymbol{x}$ has the following form:

$$\partial\|\boldsymbol{x}\|_r = \{\boldsymbol{\rho} \in \mathbb{R}^n \mid \Pi_{\mathcal{T}}(\boldsymbol{\rho}) = \boldsymbol{e} \text{ and } \|\Pi_{\mathcal{T}^\perp}(\boldsymbol{\rho})\|_{\mathcal{A}_r}^* \leq 1\}, \tag{4.15}$$

where $\Pi_{\mathcal{T}}(\cdot)$ is the orthogonal projection onto $\mathcal{T}$, and $\|\boldsymbol{x}\|^* := \sup_{\|\boldsymbol{a}\|\leq 1}\langle \boldsymbol{x}, \boldsymbol{a}\rangle$ is the dual norm of $\|\cdot\|$. The decomposable norm was defined in [63, 6], and many interesting regularizers belong to this category, including:

- Sparse vectors: for the ℓ_1 regularizer, $\mathcal{T}$ is the span of all points with the same support as $\boldsymbol{x}$.

- Group sparse vectors: suppose that the index set can be partitioned into a set of $N_{\mathcal{G}}$ disjoint groups, say $\mathcal{G} = \{G_1, \ldots, G_{N_{\mathcal{G}}}\}$, and define the (1,$\alpha$)-group norm by $\|\boldsymbol{x}\|_{\mathcal{G},\alpha} := \sum_{t=1}^{N_{\mathcal{G}}} \|\boldsymbol{x}_{G_t}\|_\alpha$. If S_G denotes the subset of groups where $\boldsymbol{x}_{G_t} \neq 0$, then the subgradient has the following form:

$$\partial\|\boldsymbol{x}\|_{1,\alpha} := \{\boldsymbol{\rho} \mid \boldsymbol{\rho} = \sum_{t\in S_G} \boldsymbol{x}_{G_t}/\|\boldsymbol{x}_{G_t}\|_\alpha^* + \sum_{t\notin S_G} \boldsymbol{m}_t\},$$

where $\|\boldsymbol{m}_t\|_\alpha^* \leq 1$ for all $t \notin S_G$. Therefore, the group sparse norm is also decomposable with

$$\mathcal{T} := \{\boldsymbol{x} \mid \boldsymbol{x}_{G_t} = 0 \text{ for all } t \notin S_G\}. \tag{4.16}$$

- Low-rank matrices: for the nuclear norm regularizer $\|\cdot\|_*$, which is defined to be the sum of singular values, the subgradient can be written as

$$\partial\|X\|_* = \{UV^T + W \mid U^T W = 0, WV = 0, \|W\|_2 \leq 1\},$$

where $\|\cdot\|_2$ is the matrix 2 norm and U, V are the left/right singular vectors of X corresponding to *non-zero* singular values. The above subgradient can also be written in the decomposable form (4.15), where $\mathcal{T}$ is defined to be $\text{span}(\{\boldsymbol{u}_i \boldsymbol{v}_j^T\}_{i,j=1}^k)$ where $\{\boldsymbol{u}_i\}_{i=1}^k, \{\boldsymbol{v}_i\}_{i=1}^k$ are the columns of U and V.

Applications. Next we discuss some widely used applications of superposition-structured models, and the corresponding instances of the class of M-estimators in (4.14).

• Gaussian graphical model with latent variables: let Θ denote the precision matrix with corresponding covariance matrix $\Sigma = \Theta^{-1}$. [8] showed that the precision matrix will have a low rank + sparse structure when some random variables are hidden, thus $\Theta = S - L$ can be estimated by solving the following regularized MLE problem:

$$\min_{S,L:L\succeq 0, S-L\succ 0} -\log\det(S-L) + \langle S-L, \Sigma\rangle + \lambda_S\|S\|_1 + \lambda_L \operatorname{tr}(L). \qquad (4.17)$$

• Multi-task learning: given k tasks, each with sample matrix $X^{(r)} \in \mathbb{R}^{n_r \times d}$ (n_r samples in the r-th task) and labels $y^{(r)}$, [36] proposes minimizing the following objective:

$$\sum_{r=1}^{k} \ell(y^{(r)}, X^{(r)}(S^{(r)} + B^{(r)})) + \lambda_S\|S\|_1 + \lambda_B\|B\|_{1,\infty}, \qquad (4.18)$$

where $\ell(\cdot)$ is the loss function and $S^{(r)}$ is the r-th column of S.

• Noisy PCA: to recover a covariance matrix corrupted with sparse noise, a widely used technique is to solve the matrix decomposition problem [9]. In

contrast to the squared loss above, an exponential PCA problem [13] would use a Bregman divergence for the loss function.

4.3.1 Exploiting Problem Structure – Block Coordinate Descent for Computing Newton direction

Given k sets of variables $\boldsymbol{\theta} = [\boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(k)}]$, and each $\boldsymbol{\theta}^{(r)} \in \mathbb{R}^n$, let $\boldsymbol{\Delta}^{(r)}$ denote perturbation of $\boldsymbol{\theta}^{(r)}$, and $\boldsymbol{\Delta} = [\boldsymbol{\Delta}^{(1)}, \ldots, \boldsymbol{\Delta}^{(k)}]$. We define $g(\boldsymbol{\theta}) := \mathcal{L}(\sum_{r=1}^{k} \boldsymbol{\theta}^{(r)}) = \mathcal{L}(\bar{\boldsymbol{\theta}})$ to be the loss function, and $h(\boldsymbol{\theta}) := \sum_{r=1}^{k} \mathcal{R}^{(r)}(\boldsymbol{\theta}^{(r)})$ to be the regularization. To apply the proximal Newton method (as described in Section 3.1), given the current estimate $\boldsymbol{\theta}$, we form the quadratic approximation of the smooth loss function:

$$\bar{g}(\boldsymbol{\theta} + \boldsymbol{\Delta}) = g(\boldsymbol{\theta}) + \sum_{r=1}^{k} \langle \boldsymbol{\Delta}^{(t)}, G \rangle + \frac{1}{2} \boldsymbol{\Delta}^T \mathcal{H} \boldsymbol{\Delta}, \tag{4.19}$$

where $G = \nabla \mathcal{L}(\bar{\boldsymbol{\theta}})$ is the gradient of $\mathcal{L}$ and $\mathcal{H}$ is the Hessian matrix of $g(\boldsymbol{\theta})$. Note that $\nabla_{\bar{\boldsymbol{\theta}}} \mathcal{L}(\bar{\boldsymbol{\theta}}) = \nabla_{\boldsymbol{\theta}^{(r)}} \mathcal{L}(\bar{\boldsymbol{\theta}})$ for all r so we simply write ∇ and refer to the gradient at $\bar{\boldsymbol{\theta}}$ as G (and similarly for ∇^2). By the chain rule, we can show that the Hessian Matrix has the following structure:

$$\mathcal{H} := \nabla^2 g(\boldsymbol{\theta}) = \begin{bmatrix} H & \cdots & H \\ \vdots & \ddots & \vdots \\ H & \cdots & H \end{bmatrix}, \quad H := \nabla^2 \mathcal{L}(\bar{\boldsymbol{\theta}}). \tag{4.20}$$

The Newton direction $\boldsymbol{d}$ is defined to be:

$$[\boldsymbol{d}^{(1)}, \ldots, \boldsymbol{d}^{(k)}] = \underset{\boldsymbol{\Delta}^{(1)}, \ldots, \boldsymbol{\Delta}^{(k)}}{\operatorname{argmin}} \bar{g}(\boldsymbol{\theta} + \boldsymbol{\Delta}) + \sum_{r=1}^{k} \lambda_r \|\boldsymbol{\theta}^{(r)} + \boldsymbol{\Delta}^{(r)}\|_{\mathcal{A}_r} := Q_{\mathcal{H}}(\boldsymbol{\Delta}; \boldsymbol{\theta}). \tag{4.21}$$

Based on the structure of Hessian in (4.20), we propose a block coordinate descent (or alternating minimization) method to solve (4.21). At each iteration, we pick a variable set $\boldsymbol{\Delta}^{(r)}$ where $r \in \{1, 2, \ldots, k\}$ by a cyclic (or random) order, and update the parameter set $\boldsymbol{\Delta}^{(r)}$ while keeping other parameters fixed. Assume $\boldsymbol{\Delta}$ is the current solution (for all the variable sets), then the subproblem with respect to $\boldsymbol{\Delta}^{(r)}$ can be written as

$$\boldsymbol{\Delta}^{(r)} \leftarrow \operatorname*{argmin}_{\boldsymbol{d} \in \mathbb{R}^n} \frac{1}{2} \boldsymbol{d}^T H \boldsymbol{d} + \langle \boldsymbol{d}, G + \sum_{t:r \neq t} H \boldsymbol{\Delta}^{(t)} \rangle + \lambda_r \|\boldsymbol{\theta}^{(r)} + \boldsymbol{d}\|_{\mathcal{A}_r}. \tag{4.22}$$

The subproblem (4.22) is just a typical quadratic problem with a specific regularizer, so there already exist efficient algorithms for solving it for different choices of $\|\cdot\|_{\mathcal{A}}$.

4.3.2 Exploiting Model Structure – Active Subspace Selection

Since the quadratic subproblem (4.21) contains a large number of variables, directly applying the above quadratic approximation framework is not efficient. In this subsection, we provide a general *active subspace selection* technique, which dramatically reduces the size of variables by exploiting the structure of regularizers.

Given the current $\boldsymbol{\theta}$, our subspace selection approach partitions each $\boldsymbol{\theta}^{(r)}$ into $\mathbb{S}^{(r)}_{fixed}$ and $\mathbb{S}^{(r)}_{free} = (\mathbb{S}^{(r)}_{fixed})^{\perp}$, and then restricts the search space of the Newton direction in (4.21) within $\mathbb{S}_{free}$, which yields the following quadratic approximation problem:

$$[\boldsymbol{d}^{(1)}, \ldots, \boldsymbol{d}^{(k)}] = \operatorname*{argmin}_{\boldsymbol{\Delta}^{(1)} \in \mathbb{S}^{(1)}_{free}, \ldots, \boldsymbol{\Delta}^{(k)} \in \mathbb{S}^{(k)}_{free}} \bar{g}(\boldsymbol{\theta} + \boldsymbol{\Delta}) + \sum_{r=1}^{k} \lambda_r \|\boldsymbol{\theta}^{(r)} + \boldsymbol{\Delta}^{(r)}\|_{\mathcal{A}_r}. \tag{4.23}$$

Each group of parameter has its own fixed/free subspace, so we now focus on a single parameter component $\boldsymbol{\theta}^{(r)}$. An ideal subspace selection procedure would satisfy:

> **Property (I).** Given the current iterate $\boldsymbol{\theta}$, any updates along directions in the fixed set, for instance as $\boldsymbol{\theta}^{(r)} \leftarrow \boldsymbol{\theta}^{(r)} + \boldsymbol{a},\ \boldsymbol{a} \in \mathbb{S}^{(r)}_{\text{fixed}}$, does not improve the objective function value.
>
> **Property (II).** The subspace $\mathbb{S}_{free}$ converges to the support of the final solution in a finite number of iterations.

Suppose given the current iterate, we first do updates along directions in the fixed set, and then do updates along directions in the free set. Property (I) ensures that this is equivalent to ignoring updates along directions in the fixed set in this current iteration, and focusing on updates along the free set. As we will show in the next section, this property would suffice to ensure global convergence of our procedure. Property (II) will be used to derive the asymptotic quadratic convergence rate.

We will now discuss our active subspace selection strategy which will satisfy both properties above. Consider the parameter component $\boldsymbol{\theta}^{(r)}$, and its corresponding regularizer $\|\cdot\|_{\mathcal{A}_r}$. Based on the definition of decomposable norm in (4.15), there exists a subspace $\mathcal{T}_r$ where $\Pi_{\mathcal{T}_r}(\boldsymbol{\rho})$ is a fixed vector for any subgradient of $\|\cdot\|_{\mathcal{A}_r}$. The following proposition explores some properties of the sub-differential of the overall objective $F(\boldsymbol{\theta})$ in (4.14).

Proposition 4. *Consider any unit-norm vector* $\boldsymbol{a}$*, with* $\|\boldsymbol{a}\|_{\mathcal{A}_r} = 1$*, such that* $\boldsymbol{a} \in \mathcal{T}_r^{\perp}$*.*

(a) The inner-product of the sub-differential $\partial_{\boldsymbol{\theta}^{(r)}} F(\boldsymbol{\theta})$ *with* $\boldsymbol{a}$ *satisfies:*

$$\langle \boldsymbol{a}, \partial_{\boldsymbol{\theta}^{(r)}} F(\boldsymbol{\theta}) \rangle \in [\langle \boldsymbol{a}, G \rangle - \lambda_r, \langle \boldsymbol{a}, G \rangle + \lambda_r]. \tag{4.24}$$

(b) Suppose $|\langle \boldsymbol{a}, G \rangle| \leq \lambda_r$*. Then,* $0 \in \operatorname{argmin}_{\sigma} F(\boldsymbol{\theta} + \sigma \boldsymbol{a})$*.*

The proposition thus implies that if $|\langle \boldsymbol{a}, G \rangle| \leq \lambda_r$ and $\mathcal{S}_{\text{fixed}}^{(r)} \subset \mathcal{T}_r^{\perp}$ then Property (I) immediately follows. The difficulty is that the set $\{\boldsymbol{a} \mid |\langle \boldsymbol{a}, G \rangle| \leq \lambda_r\}$ is possibly hard to characterize, and even if we could characterize this set, it may not be amenable enough for the optimization solvers to leverage in order to provide a speedup. Therefore, we propose an alternative characterization of the fixed subspace:

Definition 5. *Let* $\boldsymbol{\theta}^{(r)}$ *be the current iterate,* $\operatorname{prox}_{\lambda}^{(r)}$ *be the proximal operator defined by*

$$\operatorname{prox}_{\lambda}^{(r)}(\boldsymbol{x}) = \operatorname*{argmin}_{\boldsymbol{y}} \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{x}\|^2 + \lambda \|\boldsymbol{y}\|_{\mathcal{A}_r},$$

and $\mathcal{T}_r(\boldsymbol{x})$ *be the subspace for the decomposable norm* (4.15) $\|\cdot\|_{\mathcal{A}_r}$ *at point* $\boldsymbol{x}$*. We can define the fixed/free subset at* $\boldsymbol{\theta}^{(r)}$ *as:*

$$\mathcal{S}_{\mathit{fixed}}^{(r)} := [\mathcal{T}(\boldsymbol{\theta}^{(r)})]^{\perp} \cap [\mathcal{T}(\operatorname{prox}_{\lambda_r}^{(r)}(G))]^{\perp}, \quad \mathcal{S}_{\mathit{free}}^{(r)} = {\mathcal{S}_{\mathit{fixed}}^{(r)}}^{\perp}. \tag{4.25}$$

It can be shown that from the definition of the proximal operator, and Definition 5, it holds that $|\langle \boldsymbol{a}, G\rangle| < \lambda_r$, so that we would have local optimality in the direction $\boldsymbol{a}$ as before. We have the following proposition:

Proposition 6. *Let $\mathcal{S}^{(r)}_{fixed}$ be the fixed subspace defined in Definition 5. We then have:*

$$0 = \underset{\boldsymbol{\Delta}^{(r)} \in \mathcal{S}^{(r)}_{fixed}}{\operatorname{argmin}} \bar{Q}_{\mathcal{H}}([\mathbf{0}, \ldots, \mathbf{0}, \boldsymbol{\Delta}^{(r)}, \mathbf{0}, \ldots, \mathbf{0}]; \boldsymbol{\theta}).$$

We are also able to prove that $\mathcal{S}_{\text{free}}$ as defined above converges to the final support, as required in Property (II) above. We will now detail some examples of the fixed/free subsets defined above.

- For ℓ_1 regularization: $\mathcal{S}_{\text{fixed}}$ is $\text{span}\{e_i \mid \theta_i = 0 \text{ and } |\nabla_i \mathcal{L}(\hat{\boldsymbol{\theta}})| \leq \lambda\}$ where e_i is the i^{th} canonical vector.

- For nuclear norm regularization: the selection scheme can be written as

$$\mathcal{S}_{\text{free}} = \{U_A D V_A^T \mid D \in \mathbb{R}^{k \times k}\}, \tag{4.26}$$

where $U_A = \text{span}(U, U_g), V_A = \text{span}(V, V_g)$, with $\Theta = U\Sigma V^T$ is the thin SVD of Θ and U_g, V_g are the left and right singular vectors of $\text{prox}_\lambda(\Theta - \nabla\mathcal{L}(\Theta))$. The proximal operator $\text{prox}_\lambda(\cdot)$ in this case corresponds to singular-value soft-thresholding, and can be computed by the iterative QR algorithm or the Lanczos algorithm.

• For group sparse regularization: in the $(1,2)$-group norm case, let S_G be the nonzero groups, then the fixed groups F_G can be defined by $F_G := \{i \mid i \notin S_G \text{ and } \|G_{G_i}\| \leq \lambda\}$, and the free subspace will be

$$\mathcal{S}_{\text{free}} = \{\boldsymbol{\theta} \mid \boldsymbol{\theta}_i = 0 \ \forall i \in F_G\}. \tag{4.27}$$

Algorithm 5: QUIC & DIRTY: Quadratic Approximation Framework for Dirty Statistical Models

Input : Loss function $\mathcal{L}(\cdot)$, regularizers $\lambda_r \|\cdot\|_{\mathcal{A}_r}$ for $r = 1, \ldots, k$, and initial iterate $\boldsymbol{\theta}_0$.
Output: Sequence $\{\boldsymbol{\theta}_t\}$ such that $\{\bar{\boldsymbol{\theta}}_t\}$ converges to $\bar{\boldsymbol{\theta}}^\star$.

```
for t = 0, 1, ... do
    Compute θ̄_t ← Σ_{r=1}^k θ_t^(r).
    Compute ∇L(θ̄_t).
    Compute S_free by (4.25).
    for sweep = 1, ..., T_outer do
        for r = 1, ..., k do
            Solve the subproblem (4.22) within S_free^(t).
            Update Σ_{r=1}^k ∇²L(θ̄_t) Δ^(r).
    Find the step size α.
    θ^(r) ← θ^(r) + α Δ^(r) for all r = 1, ..., k.
```

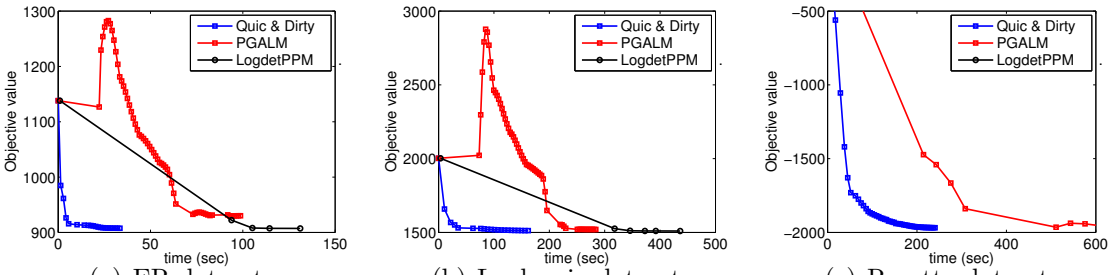
4.3.3 Experimental Results

We demonstrate that our algorithm is extremely efficient for two applications: Gaussian Markov Random Fields (GMRF) with latent variables (with sparse + low rank structure) and multi-task learning problems (with sparse + group sparse structure).

GMRF with Latent Variables Due to the importance of the latent variable Gaussian Markov Random Fields (GMRF) model, several software packages have been recently developed that solve the corresponding superposition-structured M-estimator in eq (4.17). We compare our algorithm with two state-of-the-art software packages. The LogdetPPA algorithm was proposed in [83] and used in [8] to solve (4.17). The PGALM algorithm was proposed in [56]. We run our algorithm on three gene expression datasets: the ER dataset ($p = 692$), the Leukemia dataset ($p = 1255$), and a subset of the Rosetta dataset ($p = 2000$)[1] For the parameters, we use $\lambda_S = 0.5, \lambda_L = 50$ for the ER and Leukemia datasets, which give us low-rank and sparse results. For the Rosetta dataset, we use the parameters suggested in LogdetPPA, with $\lambda_S = 0.0313, \lambda_L = 0.1565$. The results in Figure 4.2 shows that our algorithm is more than 10 times faster than other algorithms. Note that in the beginning PGALM tends to produce infeasible solutions (L or $S - L$ is not positive definite), which is not plotted in the figures.

Multiple-task learning with superposition-structured regularizers. We follow [36] and transform multi-class problems into multi-task problems. For a multiclass dataset with k classes and n samples, for each $r = 1, \ldots, k$, we generate $\boldsymbol{y}^r \in \{0, 1\}^n$ to be the vector such that $y_i^{(k)} = 1$ if and only if the i-th sample is in class r. Our first dataset is the USPS dataset which was first collected in [81] and subsequently widely used in multi-task papers. On

[1]The full dataset has $p = 6316$ but the other methods cannot solve this size problem.



(a) ER dataset (b) Leukemia dataset (c) Rosetta dataset

Figure 4.2: Comparison of algorithms on the latent feature GMRF problem using gene expression datasets. Our algorithm is much faster than PGALM and LogdetPPA.

this dataset, the use of several regularizers is crucial for good performance. For example, [36] demonstrates that on USPS, using lasso and group lasso regularizations together outperforms models with a single regularizer. However, they only consider the squared loss in their paper, whereas we consider a logistic loss which leads to better performance. For example, we get 7.47% error rate using 100 samples in USPS dataset, while using the squared loss the error rate is 10.8% [36]. Our second dataset is a larger document dataset RCV1 downloaded from LIBSVM Data, which has 53 classes and 47,236 features. We show that our algorithm is much faster than other algorithms on both datasets, especially on RCV1 where we are more than 20 times faster than proximal gradient descent. Here our subspace selection techniques works well because we expect that the active subspace at the true solution is small.

Table 4.3: The comparisons on multi-task problems.

dataset	number of training data	relative error	Dirty Models (sparse + group sparse)			Other Models	
			QUIC & DIRTY	proximal gradient	ADMM	Lasso	Group Lasso
USPS	100	10^{-1}	8.3% / **0.42s**	8.5% / 1.8s	8.3% / 1.3	10.27%	8.36%
	100	10^{-4}	7.47% / **0.75s**	7.49% / 10.8s	7.47% / 4.5s		
	400	10^{-1}	2.92% / **1.01s**	2.9% / 9.4s	3.0% / 3.6s	4.87%	2.93%
	400	10^{-4}	2.5% / **1.55s**	2.5% / 35.8	2.5% / 11.0s		
RCV1	1000	10^{-1}	18.91% / **10.5s**	18.5%/47s	18.9% / 23.8s	22.67%	20.8%
	1000	10^{-4}	18.45% / **23.1s**	18.49% / 430.8s	18.5% / 259s		
	5000	10^{-1}	10.54% / **42s**	10.8% / 541s	10.6% / 281s	13.67%	12.25%
	5000	10^{-4}	10.27% / **87s**	10.27% / 2254s	10.27% / 1191s		

4.4 Summary of the Contribution

The greedy coordinate descent algorithm for NMF is published in [26], and the code can be downloaded at `http://www.cs.utexas.edu/~cjhsieh/nmf`. The active subspace selection approach for nuclear norm has been published in [29] and the code can be downloaded at `http://www.cs.utexas.edu/~cjhsieh/nuclear_active_1.1.zip`, and in [28] we extend the algorithm to solve dirty statistical models. The divide-and-conquer algorithm for Kernel SVM presented in Section 4.2 is published in [30], and the code can be downloaded from `http://www.cs.utexas.edu/~cjhsieh/dcsvm`.

Chapter 5

Exploiting Structure for General Problems

In this section, we discuss how to explore the structure of problem, model, and data distribution for a wide class of optimization problems, and we further develop a novel divide-and-conquer framework on distributed systems.

5.1 Exploiting Problem Structure—Efficient Proximal Newton Methods for General Functions

We have provided several examples in this proposal showing that exploiting problem structure is very important for developing efficient optimization algorithms, especially second order methods. To conclude the thesis, we discuss two classes of problems where *proximal Newton methods* can be efficiently applied by exploiting structure of the Hessian matrix.

Linear Empirical Risk Minimization Problems

We first discuss problems that aim to learn a linear model under the Empirical Risk Minimization framework. In those cases, the objective function can be written as

$$\min_{\boldsymbol{w}\in\mathbb{R}^d} g(\boldsymbol{w}) + h(\boldsymbol{w}) \quad \text{where} \quad g(\boldsymbol{w}) = \sum_{i=1,\ldots,n} \ell(\boldsymbol{w}^T\boldsymbol{x}_i, y_i),$$

where $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$ is the training dataset, $\boldsymbol{w}$ is the model and $h(\cdot)$ is the regularizer. The Hessian of the loss function has the following form:

$$H = \nabla^2 g(\boldsymbol{w}) = X^T D X, \tag{5.1}$$

where $X \in \mathbb{R}^{n \times d}$ is the data matrix, and D is a diagonal matrix with

$$D_{ii} = \frac{\partial^2 \ell(z, y_i)}{\partial z^2}\Big|_{z=\boldsymbol{w}^T \boldsymbol{x}_i}$$

Therefore, we can efficiently conduct the following two operations when solving the quadratic approximation subproblem (3.5) for proximal Newton method:

- **Hessian Vector Product:** Without exploiting the structure of the Hessian matrix, Hessian vector product has to be computed in $O(d^2)$ time where d is the dimensionality. This is very time consuming for high-dimensional (sparse) problems. Using the structure of Hessian in (5.1), the Hessian-vector product computation can be improved from $O(d^2)$ to $O(\text{nnz}(X))$ time, and this is very efficient when the input data X is a sparse matrix. The approach has been used in [53] for solving the ℓ_2-regularized logistic regression problem, where they use a trust region Newton method to solve the quadratic approximation problem (3.5).

- **Coordinate descent update:** If we apply the coordinate descent algorithm to solve the quadratic subproblem (3.5), each coordinate descent update mainly involves the computation of $\boldsymbol{e}_i^T H \boldsymbol{w}$ where $\boldsymbol{w} \in \mathbb{R}^d$ is the current solution. Without exploiting the structure of Hessian, we have to

compute $\boldsymbol{e}_i^T H\boldsymbol{x}$ in $O(d)$ time. Furthermore, the $O(d^2)$ space complexity is needed for storing H in memory.

By exploiting the structure of Hessian matrix in (5.1), each coordinate descent update can be conducted efficiently. We can maintain a vector $\boldsymbol{h} = X\boldsymbol{w}$, and at each iteration $\boldsymbol{e}_i^T H\boldsymbol{w}$ can be computed by $\bar{\boldsymbol{x}}_i^T D\boldsymbol{h}$, where $\bar{\boldsymbol{x}}_i$ is the i-th column of X. This only requires $O(d_i)$ time complexity, where d_i is number of nonzero elements in $\bar{\boldsymbol{x}}_i$. After each coordinate update, we have to maintain $\boldsymbol{h}$, which takes only $O(d_i)$ time. This approach has been used in [21, 89] for ℓ_1-regularized logistic regression.

Matrix Functions

In Section 3.2, we have shown that the Hessian of sparse inverse covariance estimation loss function $f(X) = -\log\det(X) + \operatorname{trace}(SX)$ is $X^{-1} \otimes X^{-1}$ where $\otimes$ indicates the Kronecker product. By exploiting this structure, we can improve the time complexity of coordinate descent updates for solving the Newton direction subproblem (3.5). But do we have the similar structure for other functions? Luckily, the answer is yes for most of the matrix functions.

Define a matrix function to be $f : \mathbb{R}^{d\times d} \to \mathbb{R}$, we can define the gradient to be $\nabla f : \mathbb{R}^{d\times d} \to \mathbb{R}^{d\times d}$ where $\nabla_{ij} f(X) = \frac{\partial f(X)}{\partial X_{ij}}$, and the Hessian matrix to be $\nabla^2 f : \mathbb{R}^{d\times d} \to \mathbb{R}^{d^2\times d^2}$, where

$$\frac{\partial f}{\partial X} = \frac{\partial \operatorname{vec}(f^\top)}{\partial \operatorname{vec}^\top(X^\top)} = \begin{pmatrix} \frac{\partial f_{11}}{\partial X_{11}} & \frac{\partial f_{11}}{\partial X_{12}} & \cdots & \frac{\partial f_{11}}{\partial X_{mn}} \\ \frac{\partial f_{12}}{\partial X_{11}} & \frac{\partial f_{12}}{\partial X_{12}} & \cdots & \frac{\partial f_{12}}{\partial X_{mn}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_{kl}}{\partial X_{11}} & \frac{\partial f_{kl}}{\partial X_{12}} & \cdots & \frac{\partial f_{kl}}{\partial X_{mn}} \end{pmatrix}.$$

We then consider the class of *rational matrix-matrix functions*, where $f(\cdot)$ is formed by four operations: $+, -, \times, (\cdot)^{-1}$. [65] prove the following theorem:

Theorem 7. *For any rational matrix-matrix functions, the derivative can be written as*

$$\nabla^2 f(X) = \left(\sum_{i=1}^{k} A_i \otimes B_i \right) \operatorname{vec}(D), \tag{5.2}$$

where A_i, B_i are rational matrix-matrix functions of X.

Given this following two differentiation rules for matrix-vector functions:

$$\begin{aligned} \operatorname{vec}\left(\left(\frac{\partial}{\partial X} \log\det(f(X)) \right)^\top \right) &= \left(\frac{\partial f}{\partial X} \right)^\top \operatorname{vec}\left((f(X))^{-1} \right) \\ \operatorname{vec}\left(\left(\frac{\partial}{\partial X} \operatorname{trace}(f(X)) \right)^\top \right) &= \left(\frac{\partial f}{\partial X} \right)^\top \operatorname{vec}(I) \end{aligned}$$

We can conclude that the Hessian matrix for all the matrix-vector functions composed with $\log\det(\cdot), \operatorname{trace}(\cdot), +, -, \times, (\cdot)^{-1}$ can be written as (5.2). Therefore, if we apply a proximal Newton method to minimize $f(X) + h(X)$ where $h(X)$ is the regularization, the Newton direction subproblem has the following form:

$$\sum_{i=1}^{k} \operatorname{vec}(D)^T A_i \otimes B_i \operatorname{vec}(D) + \langle \nabla f(X), D \rangle + h(X).$$

We are able to efficiently conduct the following two operations when solving this quadratic subproblem.

- **Hessian Vector Product:** Without exploiting the structure of the Hessian matrix, the Hessian vector product has to be computed in $O(d^4)$ time where d is the dimensionality. This is too expensive for high-dimensional problems. Using the structure of Hessian in (5.2), the Hessian-vector product can be computed in $O(d^3)$ time by using the fact that $(A_i \otimes B_i)\,\mathrm{vec}(D) = A_i D B_i$.

- **Coordinate descent update:** If we apply the coordinate descent algorithm for solving the quadratic subproblem (3.5), each coordinate descent update requires $O(d^2)$ time and $O(d^4)$ space to store the Hessian matrix in memory. By exploiting the structure of Hessian matrix as shown in (5.2), each coordinate descent update can be conduct efficiently. We can maintain a set of matrices $W_i = DB_i$ for all $i = 1, \ldots, k$, and then $(\sum_{i=1}^k A_i D B_i)_{ij}$ can be computed efficiently by $\sum_{i=1}^k (A_i W_i)_{ij}$, which only requires $O(d)$ time. This is the strategy we used in the sparse inverse covariance estimation problem (Section 3.2), and here we show this strategy can be generalized to the matrix function.

5.2 Exploiting Model Structure—Coordinate Descent with Priority

In machine learning problems, usually only a subset of variables are crucial and needed to be updated frequently, and thus **variable selection** is a very important technique for speeding the optimization process. We consider minimizing the objective function $f(\boldsymbol{\theta})$. To conduct variable selection, we first

define and measure the "importance" of each variable. This can be mathematically defined as a function $q : \mathbb{R}^d \to \mathbb{R}^d$ that maps the current solution $\bar{\boldsymbol{\theta}}$ to the importance of each variable. This mapping can be defined in multiple ways, and the most straightforward definition is the maximum amount of objective function reduction when updating each variable, which can be written formally as:

$$q_i(\bar{\boldsymbol{\theta}}) := \max_{d \in \mathbb{R}} f(\bar{\boldsymbol{\theta}}) - f(\bar{\boldsymbol{\theta}} + d\boldsymbol{e}_i), \tag{5.3}$$

where $\boldsymbol{e}_i$ is the i-th indicator vector. However, this quantity is sometimes hard to compute. Instead, if we approximate the current function by a quadratic function:

$$f(\boldsymbol{\theta}) \approx \tilde{f}(\boldsymbol{\theta}) := f(\bar{\boldsymbol{\theta}}) + (\boldsymbol{\theta} - \bar{\boldsymbol{\theta}})^T \nabla f(\bar{\boldsymbol{\theta}}) + \frac{\gamma}{2}(\boldsymbol{\theta} - \bar{\boldsymbol{\theta}})^T(\boldsymbol{\theta} - \bar{\boldsymbol{\theta}}), \tag{5.4}$$

then the approximate objective function decrease can be measured by

$$\tilde{q}_i(\bar{\boldsymbol{\theta}}) := \max_{d \in \mathbb{R}} \tilde{f}(\bar{\boldsymbol{\theta}}) - \tilde{f}(\bar{\boldsymbol{\theta}} + d\boldsymbol{e}_i) = \frac{\nabla_i^2 f(\bar{\boldsymbol{\theta}})}{2\gamma}, \tag{5.5}$$

and thus the importance is proportional to the gradient. Therefore it is very common to use the gradient as the importance of each variable.

The natural way to incorporate variable selection into the optimization procedure is to conduct the **greedy coordinate descent** update. In classical coordinate descent algorithms, each time only one variable is updated, and the update sequence can be chosen cyclically [54, 2] or randomly [64]. To explore the importance of variables, a straightforward way is to choose the most important variable to update at each step, resulting a greedy coordinate

descent algorithm. The greedy coordinate descent algorithm first appeared in [19], and have been widely used in machine learning applications [93, 78]. Also, the decomposition method with maximum violating pair selection for solving the kernel SVM problem is very close to greedy coordinate descent, where a pair important variables instead of one single variable are selected at each step. This important technique has been proposed by [41] and implemented in many state-of-the-art SVM solvers including LIBSVM [10] and SVMLight [37].

For most problems, it is non-trivial to maintain the gradient efficiently, so the "importance" has to be recomputed periodically during the optimization procedure. Therefore, in Section 4.3, we further develop an *active subspace selection* approach within this proximal Newton framework to exploit the model structure. In the proximal Newton method proposed in Section 4.3, the "importance" of each variable (or subspace) is measured by 0 (inactive) or 1 (active) using the optimality condition of the problem, and then we solve the reduce-sized subproblem which contains only active variables (or subspace).

In general, for any optimization methods solving the RLM problem with decomposable norm regularization, we can periodically compute the "importance" or gradient for each variable/subspace, and then only solve the reduce-sized problem. The algorithm is guaranteed to converge if the active variable/subspace selection is done by our proposed rule in Section 4.3.

5.3 Exploiting Data Distribution—Distributed Divide-and-Conquer Algorithms

In this section we discuss a parallel proximal Newton framework that can be used for solving composite optimization problems. The idea of the algorithm is to divide the variable set into several disjoint partitions, and each worker conducts updates using the local information. At each synchronization point, the workers communicate the gradient information and coordinating together to find a suitable step size to do the update. We will also show two interesting aspects of the proposed algorithm:

1. The convergence speed of the algorithm highly depends on the quality of the partition of variables. Therefore, to develop an efficient algorithm, we have to exploit the (clustering) structure of data to obtain a better partition which minimizes the correlation between variables.

2. When the smooth part of the objective function is quadratic and the non-smooth part is separable, the line search procedure can be conducted with $O(1)$ communication time.

3. Combining the above two observations, we implement a distributed kernel SVM solver PBM. Our algorithm achieves the state-of-the-art performance.

5.3.1 Related Work

Recently there are some divide-and-conquer approaches for parallel programming, but all of them are based on the *random partition*. Assume the dataset is randomly partitioned into k blocks, [96] proposed to train each partition independently and averaging the results, and [94] further provided the theoretical guarantee for this approach. Recently, [75] proposed an iterative algorithm based on a distributed Alternating Direction Method of Multipliers (ADMM, [5]). Recently, [86, 47, 35] propose parallel block minimization for dual linear SVM, where they all use random partition of dual variables. Instead of using random partition, our proposed work uses clustering algorithm based on the objective.

5.3.2 A Parallel Proximal Newton Framework

To introduce the new framework, we take another view of the divide-and-conquer algorithms—all of them are trying to approximate the Hessian matrix by a block-diagonal matrix.

Consider the composite minimization problem

$$\operatorname*{argmin}_{\boldsymbol{x}} \big\{ g(\boldsymbol{x}) + h(\boldsymbol{x}) \big\} = f(\boldsymbol{x}), \tag{5.6}$$

where $g(\boldsymbol{x})$ is the smooth part (usually corresponds to the loss function), and $h(\boldsymbol{x})$ is a convex function (usually corresponds to the regularization). We partition the variables $\boldsymbol{x}$ into k disjoint index sets $\{S_r\}_{r=1}^k$ such that

$$S_1 \cup S_2 \cup \cdots \cup S_k = \{1, \ldots, n\} \quad \text{and} \quad S_p \cap S_q = \phi \ \forall p \neq q,$$

and we use $\pi(i)$ to denote the cluster indicator that i belongs to. We associate each worker r with a subset of variables $\boldsymbol{x}_{S_r} := \{x_i \mid i \in S_r\}$. We require $h(\boldsymbol{x})$ to be block-separable, i.e., $h(\boldsymbol{x}) = \sum_{r=1}^{k} h_{S_r}(\boldsymbol{x}_{S_r})$. Note that our framework allows any partition, and we will discuss how to obtain a better partition later.

At each iteration, to solve Problem (5.6) we form the following quadratic approximation around the current solution:

$$f(\boldsymbol{x}+\boldsymbol{d}) - f(\boldsymbol{x}) \approx \bar{f}_{\boldsymbol{x}}(\boldsymbol{d}) = \nabla g(\boldsymbol{x})^T \boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^T \bar{H} \boldsymbol{d} + h(\boldsymbol{x}+\boldsymbol{d}) - h(\boldsymbol{x}), \quad (5.7)$$

where the Hessian matrix is replaced by a block-diagonal approximation $\bar{H}$ where

$$\bar{H}_{ij} = \begin{cases} \nabla^2_{ij} g(\boldsymbol{x}) & \text{if } \pi(i) = \pi(j) \\ 0 & \text{otherwise.} \end{cases} \quad (5.8)$$

By solving (5.7), we obtain the descent direction $\boldsymbol{d}^*$:

$$\boldsymbol{d}^* := \operatorname*{argmin}_{\boldsymbol{d}} \bar{f}_{\boldsymbol{x}}(\boldsymbol{d}). \quad (5.9)$$

Since $\bar{H}$ is block-diagonal, problem (5.9) can be decomposed into k independent subproblems based on the partition $\{S_r\}_{r=1}^k$:

$$\boldsymbol{d}_{S_r} = \arg\min_{\boldsymbol{d}_{S_r}} \left\{ \sum_{i \in S_r} \nabla_i g(\boldsymbol{x}) d_i + \frac{1}{2}\boldsymbol{d}_{S_r}^T H_{S_r,S_r} \boldsymbol{d}_{S_r} + h_{S_r}(\boldsymbol{x}_{S_r} + \boldsymbol{d}_{S_r}) \right\} := f_{\boldsymbol{\alpha}}^{(r)}(\boldsymbol{d}_{S_r}), \quad (5.10)$$

The subproblem can be solved by any solver, and it does not need to be solved exactly. In Chapter 6 we will show the theoretical guarantee even when each subproblem is not solved exactly (for example, each subproblem can be solved by a fixed number of coordinate descent updates).

The descent direction $\boldsymbol{d}$ is the concatenation of $\boldsymbol{d}_{S_1}, \ldots \boldsymbol{d}_{S_r}$. Since $f(\boldsymbol{x}+\boldsymbol{d})$ might even increase the objective function value $f(\boldsymbol{x})$, we find the step size β to ensure the following sufficient decrease condition of the objective function value:

$$f(\boldsymbol{x}+\beta\boldsymbol{d}) - f(\boldsymbol{x}) \leq \beta\sigma\Delta, \tag{5.11}$$

where $\Delta = \nabla g(\boldsymbol{x})^T\boldsymbol{d} + h(\boldsymbol{x}+\boldsymbol{d}) - h(\boldsymbol{x})$, and $\sigma \in (0,1)$ is a constant. We then update $\boldsymbol{x} \leftarrow \boldsymbol{x} + \beta\boldsymbol{d}$. The algorithm is summarized in Algorithm 6.

Algorithm 6: Parallel Proximal Newton Method for solving (5.6)

Input : The objective function (5.6), initial $\boldsymbol{x}_0$.
Output: The solution $\boldsymbol{x}^*$.

1 Obtain a disjoint index partition $\{S_r\}_{r=1}^k$.
2 **for** $t = 0, 1, \ldots$ **do**
3 Update the diagonal blocks of the Hessian matrix **in parallel**.
4 Obtain $\boldsymbol{d}_{S_r}$ by solving subproblems (5.10) **in parallel**.
5 Obtain the step size β using line search.
6 $\boldsymbol{x}_{S_r} \leftarrow \boldsymbol{x}_{S_r} + \beta\boldsymbol{x}_{S_r}$.

5.3.3 Quality of the Variable Partition

We will show in Section 6.4 that Algorithm 6 converges to the optimal solution and has a global linear convergence rate. However, it is important to select a good partition in order to achieve faster convergence speed. Note that if $\bar{H} = \nabla^2 g(\boldsymbol{x})$ in subproblem (5.7), then the quadratic subproblem (5.7) is the same with the subproblem in proximal Newton method where the exact Hessian is used. Therefore, to achieve faster convergence, we want to minimize

the difference between $\bar{H}$ and $\nabla^2 g(\boldsymbol{x}) = H$, and this can be solved by finding a partition $\{S_r\}_{r=1}^k$ to minimize error $\|\bar{H} - H\|_F^2 = \sum_{i,j} H_{ij}^2 - \sum_{r=1}^k \sum_{i,j \in S_r} H_{ij}^2$. The minimizer can be obtained by maximizing the second term. However, we also want to have a balanced partition in order to achieve better parallelization speedup. Therefore, a common approach is to apply spectral clustering algorithms [82] on the Hessian matrix H, where the above error is normalized by the partition sizes.

5.3.4 Application to Kernel Machines

In this section, we apply the above parallel proximal Newton framework to solve the following composite optimization problem:

$$\arg\min_{\boldsymbol{\alpha} \in \mathbb{R}^n} \left\{ \boldsymbol{\alpha}^T Q \boldsymbol{\alpha} + \sum_i g_i(\alpha_i) \right\} := f(\boldsymbol{\alpha}) \quad \text{s.t. } \boldsymbol{a} \leq \boldsymbol{\alpha} \leq \boldsymbol{b}, \tag{5.12}$$

where $Q \in \mathcal{R}^{n \times n}$ is positive semi-definite and each g_i is a univariate convex function. Note that we can easily handle the box constraint $\boldsymbol{a} \leq \boldsymbol{\alpha} \leq \boldsymbol{b}$ by setting $g_i(\alpha_i) = \infty$ if $\alpha_i \notin [a_i, b_i]$, so we will omit the constraint in most part of the paper. Since the quadratic term in problem (5.12) is fixed, we do not need to recompute the Hessian matrix at each iteration (step 3 in Algorithm 6), and we will also show that the line search step (step 5 in Algorithm 6) can be computed using only $O(1)$ communication time. The resulting algorithm, called Parallel Block Minimization (PBM), beats state-of-the-art algorithms for solving kernel machines.

An important application of (5.12) in machine learning is that it is the dual problem of ℓ_2-regularized empirical risk minimization. Given a set of

instance-label pairs $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$, we consider the following ℓ_2-regularized empirical risk minimization problem:

$$\arg\min_{\boldsymbol{w}} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum\nolimits_{i=1}^n \ell_i(\boldsymbol{w}^T\Phi(\boldsymbol{x}_i)), \tag{5.13}$$

where ℓ_i is the loss function depending on the label y_i, and $\Phi(\cdot)$ is the feature mapping. For example, $\ell_i(u) = \max(0, 1 - y_i u)$ for SVM with hinge loss, and $\ell_i(u) = \log(1 + \exp(-y_i u))$ for regularized logistic regression. The dual problem of (5.13) can be written as

$$\arg\min_{\boldsymbol{\alpha}\in\mathbb{R}^n} \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} + \sum_{i=1}^n \ell_i^*(-\alpha_i), \tag{5.14}$$

where $Q \in \mathbb{R}^{n\times n}$ in this case is the kernel matrix with $Q_{ij} = \Phi(\boldsymbol{x}_i)^T\Phi(\boldsymbol{x}_j)$. Our proposed approach works in the general setting, but we will discuss in more detail the applications to kernel SVM, where Q is the kernel matrix and $\boldsymbol{\alpha}$ is the vector of dual variables. Note that, as in [27], we ignore the bias term in Eq.(5.13). Indeed, in our experimental results we did not observe improvement in test accuracy by adding the bias term.

Quadratic Subproblems. When the objective function is (5.12), the quadratic subproblem (5.10) can be written as

$$\boldsymbol{d}_{S_r} = \arg\min_{\Delta\boldsymbol{\alpha}_{S_r}} \big\{\frac{1}{2}\Delta\boldsymbol{\alpha}_{S_r}^T Q_{S_r,S_r}\Delta\boldsymbol{\alpha}_{S_r} + \sum_{i\in S_r}\bar{g}_i(\Delta\alpha_i)\big\} := f_{\boldsymbol{\alpha}}^{(r)}(\Delta\boldsymbol{\alpha}_{S_r}), \tag{5.15}$$

where $\bar{g}_i(\Delta\alpha_i) = g_i(\alpha_i+\Delta\alpha_i)+(Q\boldsymbol{\alpha})_i\Delta\alpha_i$. This subproblem has the same form with the original kernel SVM problem, so can be solved (approximately) by any existing method. We use greedy coordinate descent in our implementation.

At each iteration the variable with the largest projected gradient is chosen:

$$\begin{aligned} i^* &:= \operatorname*{argmax}_{i \in S_r} \left| \Pi_{[a_i, b_i]}\big(\alpha_i + \Delta\alpha_i - \nabla_i f_{\boldsymbol{\alpha}}^{(r)}(\Delta\boldsymbol{\alpha}_{S_r})\big) - \alpha_i - \Delta\alpha_i \right| \\ &= \operatorname*{argmax}_{i \in S_r} \left| \Pi_{[a_i, b_i]}\big(\alpha_i + \Delta\alpha_i - (Q_{S_r,S_r}\Delta\boldsymbol{\alpha}_{S_r})_i - \bar{g}_i'(\Delta\alpha_i)\big) - \alpha_i - \Delta\alpha_i \right| \end{aligned} \tag{5.16}$$

where $\Pi_{[a_i,b_i]}$ is the projection to the interval. The selection only requires $O(|S_i|)$ time if $Q_{S_r,S_r}\Delta\boldsymbol{\alpha}_{S_r}$ is maintained in local memory. Variable $\Delta\alpha_i$ is then updated by solving the following one-variable subproblem:

$$\begin{aligned} \Delta\alpha_{i^*} &\leftarrow \operatorname*{argmin}_{\delta : a_{i^*} \le \alpha_i + \Delta\alpha_i + \delta \le b_{i^*}} \frac{1}{2}(\Delta\boldsymbol{\alpha}_{S_r} + \delta\boldsymbol{e}_{i^*})Q_{S_r,S_r}(\Delta\boldsymbol{\alpha}_{S_r} + \delta\boldsymbol{e}_{i^*}) + \bar{g}_{i^*}(\Delta\alpha_{i^*} + \delta) \\ &= \operatorname*{argmin}_{\delta : a_{i^*} \le \alpha_i + \Delta\alpha_i + \delta \le b_{i^*}} \frac{1}{2}\delta^2 + (Q_{S_r,S_r}\Delta\boldsymbol{\alpha}_{S_r})\delta + \bar{g}_{i^*}(\Delta\alpha_{i^*} + \delta) \end{aligned} \tag{5.17}$$

For kernel SVM, the one-variable subproblem (5.17) has a closed form solution, while for logistic regression the subproblem can be solved by Newton's method (see [88]). The bottleneck of both (5.16) and (5.17) is to compute $Q_{S_r,S_r}\Delta\boldsymbol{\alpha}_{S_r}$, which can be maintained after each update using $O(|S_r|)$ time.

Communication Cost. There is no communication needed for solving the subproblems between workers; however, after solving the subproblems and obtaining $\boldsymbol{d}$, each worker needs to obtain the updated $(Q\boldsymbol{d})_{S_r}$ vector for next iteration. Since each worker only has local $\boldsymbol{d}_{S_r}$, we compute $Q_{:,S_r}(\boldsymbol{d}_{S_r})$ in each worker, and use a REDUCE_SCATTER collective communication to obtain updated $(Q\boldsymbol{d})_{S_r}$ for each worker. The communication cost for the collective REDUCE_SCATTER operation for an n-dimensional vector requires

$$\log(k)T_{\text{initial}} + \frac{k-1}{k}nT_{\text{byte}} \tag{5.18}$$

communication time, where T_{initial} is the message startup time and T_{byte} is the transmission time per byte (see Section 6.3 of [7]). When n is large, the second term usually dominates, so we need $O(n)$ communication time and this *does not grow with number of workers.*

Communication-efficient Line Search. After obtaining $(Q\boldsymbol{d})_{S_r}$ for each worker, we propose two communication efficient line search approaches in the following. Earlier work on distributed linear SVM solvers usually set a fixed step size [86, 35], and only recently [47] proposed an efficient line search for distributed linear SVM by synchronizing primal variables. Our algorithm is different from [47] since we focus on kernelized problems where the primal variables cannot be used.

1. **Armijo-rule based step size selection.** For general $g_i(\cdot)$, a commonly used line search approach is to adopt the Armijo-rule based step size selection and try step sizes $\beta \in \{1, \frac{1}{2}, \frac{1}{4}, \dots\}$ until β satisfies the sufficient decrease condition (5.11), and the only cost is to evaluate the objective function value. For each choice of β, $f(\boldsymbol{\alpha} + \beta\boldsymbol{d})$ can be computed as

   $$f(\boldsymbol{\alpha}+\beta\boldsymbol{d}) = f(\boldsymbol{\alpha})+\sum_r \{\beta \boldsymbol{d}_{S_r}^T (Q\boldsymbol{\alpha})_{S_r} + \frac{1}{2}\beta^2 \boldsymbol{d}_{S_r}^T (Q\boldsymbol{d})_{S_r} + \sum_{i\in S_r} g_i(\alpha_i+\beta d_i)\},$$

   so if each worker has the vector $(Q\boldsymbol{d})_{S_i}$, we can compute $f(\boldsymbol{\alpha}+\beta\boldsymbol{d})$ using $O(n/k)$ time and $O(1)$ communication cost.

2. **Optimal step size selection.** If each g_i is a linear function with bounded constraint (such as for the kernel SVM case), the optimal step size can be computed without communication. The optimal step size is defined by

$$\beta_t^* := \arg\min_{\beta} f(\boldsymbol{\alpha} + \beta \boldsymbol{d}) \text{ s.t. } \boldsymbol{a} \leq \boldsymbol{\alpha} + \beta \boldsymbol{d} \leq \boldsymbol{b}. \tag{5.19}$$

If $\sum_i g_i(\alpha_i) = \boldsymbol{p}\boldsymbol{\alpha}$, then $f(\boldsymbol{\alpha} + \beta \boldsymbol{d})$ with respect to β is a univariate quadratic function, and thus β_t^* can be obtained by the following closed form solution:

$$\beta = \min(\bar{\eta}, \max(\underline{\eta}, -\frac{\boldsymbol{\alpha}^T Q \boldsymbol{d} + \boldsymbol{p}^T \boldsymbol{d}}{\boldsymbol{d}^T Q \boldsymbol{d}})), \tag{5.20}$$

where $\bar{\eta} := \min_{i=1}^n (b_i - \alpha_i)$ and $\underline{\eta} := \max_{i=1}^n (a_i - \alpha_i)$. This can also be computed in $O(n/k)$ time and $O(1)$ communication time.

Data Partition. When Q is the kernel matrix, e.g., in kernel SVM, then the problem is equivalent to finding a good block diagonal approximation for the kernel matrix. The same problem has been discussed in [30, 77], and they showed that kmeans algorithm can be used for shift-invariant kernels, and kernel kmeans (on a subset of samples) algorithm can be used for a general kernel.

We observe PBM with kmeans partition converges much faster compared to random partition. In Figure 5.1, we test the PBM algortihm on the kernel SVM problem with Gaussian kernel, and show that the convergence is

much faster when the partition is obtained by kmeans clustering. Note that previous work for parallel linear SVM solvers [47, 35] all use random partition. The oscillatory behavior of PBM-random in Figure 5.1 was also observed in [47] for solving linear SVM problems.

The detail of PBM is in Algorithm 7.

Algorithm 7: PBM: Parallel Block Minimization for solving (5.12)

Input : Initial $\boldsymbol{\alpha}_0$.
Output: The solution $\boldsymbol{\alpha}^*$.
Obtain a disjoint index partition $\{S_r\}_{r=1}^k$.
for $t = 0, 1, \ldots$ **do**
  Obtain $\boldsymbol{d}_{S_r}$ by solving subproblems (6.2) **in parallel**.
  Compute $Q_{:,S_r}\boldsymbol{d}_{S_r}$ **in parallel**.
  Use REDUCE_SCATTER to obtain $(Q\boldsymbol{d})_{S_r}$ in each worker.
  Obtain the step size β using line search.
  $\boldsymbol{\alpha}_{S_r} \leftarrow \boldsymbol{\alpha}_{S_r} + \beta \boldsymbol{d}_{S_r}$ and $(Q\boldsymbol{\alpha})_{S_r} \leftarrow (Q\boldsymbol{\alpha})_{S_r} + \beta(Q\boldsymbol{d})_{S_r}$ **in parallel**.

Experimental Results We conduct experiments on four large-scale datasets listed in Table 5.1. We follow the procedure in [91, 30] to transform cifar and mnist8m into binary classification problems, and Gaussian kernel $K(\boldsymbol{x}_i, \boldsymbol{x}_j) =$

Table 5.1: Dataset statistics for Kernel SVM Experiments.

Dataset	# training samples	# testing samples	Number of features	C	γ
cifar	50,000	10,000	3072	2^3	2^{-22}
covtype	464,810	116,202	54	2^5	2^5
webspam	280,000	70,000	254	2^3	2^5
mnist8m	8,000,000	100,000	784	2^0	2^{-21}

$e^{-\gamma\|\boldsymbol{x}_i-\boldsymbol{x}_j\|^2}$ is used in all the comparisons. We follow the parameter settings in [30], where C and γ are selected by 5-fold cross validation on a grid of parameters. The experiments are conducted on a parallel platform at the Texas Advanced Computing Center, where each machine has an Intel E5-2680 CPU and 256GM memory. We will release our code later.

We first compare our PBM method with the following distributed kernel SVM training algorithms:

1. P-pack SVM [95]: a parallel Stochastic Gradient Descent (SGD) algorithm for kernel SVM training. We set the pack size $r = 100$ according to the original paper.

2. Random Fourier feature with distributed LIBLINEAR: random Fourier feature [71] has become popular for solving kernel SVM. In a distributed system, we can compute random features for each sample in parallel, and then solve the resulting linear SVM problem by distributed dual coordinate descent [47] implemented in MPI LIBLINEAR.

3. Nyström approximation with distributed LIBLINEAR: We implemented the ensemble Nyström approximation [45] in a distributed system and solve the resulting linear SVM problem by MPI LIBLINEAR. The approach is similar to [57], but they use a MapReduce system and we use an MPI implementation.

4. PSVM [11]: a parallel kernel SVM solver by in-complete Cholesky factorization and a parallel interior point method. We test the performance

of PSVM with the rank suggested by the original paper ($n^{0.5}$ or $n^{0.6}$ where n is number of samples).

Comparison with other solvers. We use 32 machines (each with 1 thread) and the best C, γ for all the solvers. The results in Figure 5.2 (a)-(d) indicate that our proposed algorithm is much faster than other approaches. We further test the algorithms with varied number of workers and parameters in Table 5.2. Note that PSVM usually got lower test accuracy since they approximate the kernel function by incomplete Cholesky factorization, so we only show the results in the table. We also compare our algorithm with the state-of-the-art sequential kernel SVM algorithm DC-SVM [30]. The results in Figure 5.3 shows that PBM is much faster by multiple machines.

Scalability of PBM. For the second experiment we varied the number of workers from 8 to 64, and plot the scaling behavior of PBM. In Figure 5.2 (e)-(f), we set y-axis to be the relative error defined by $(f(\boldsymbol{\alpha}_t) - f(\boldsymbol{\alpha}^*))/f(\boldsymbol{\alpha}^*)$ where $\boldsymbol{\alpha}^*$ is the optimal solution, and x-axis to be the total CPU time expended which is given by the number of seconds elapsed multiplied by the number of workers. We plot the convergence curves by setting the # cores=8, 32, 64. The perfect linear speedup is achieved if the curves overlap. This is indeed the case for covtype, and the difference is also small for webspam.

Kernel logistic regression. Finally, we implement the PBM algorithm to solve the kernel logistic regression problem. We use greedy coordinate descent proposed in [40] to solve each subproblem (6.2). The results are also

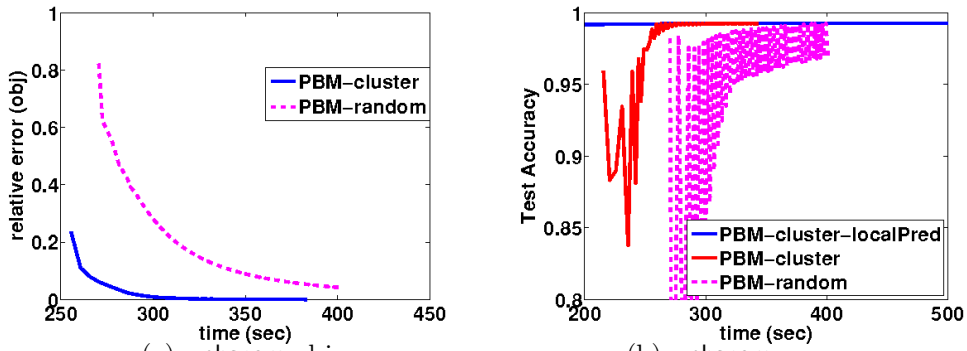
Table 5.2: Comparison on real datasets using 32 machines. The first column shows that PBM achieves good test accuracy after 1 iteration, and the second column shows PBM can achieve an accurate solution (with $\frac{f(\boldsymbol{\alpha})-f(\boldsymbol{\alpha}^*)}{|f(\boldsymbol{\alpha}^*)|} < 10^{-3}$) quickly and obtain even better accuracy. The timing for kernel logistic regression (LR) is much slower because $\boldsymbol{\alpha}$ will always be dense using the logistic loss.

	PBM (first step)		PBM (10^{-3} error)		P-packSGD		PSVM $p = n^{0.5}$		PSVM $p = n^{0.6}$	
	time(s)	acc(%)	time(s)	acc(%)	time(s)	acc(%)	time(s)	acc(%)	time(s)	acc(%)
webspam (SVM)	**16**	99.07	360	**99.26**	1478	98.99	773	75.79	2304	88.68
covtype (SVM)	**14**	96.05	772	**96.13**	1349	92.67	286	76.00	7071	81.53
cifar (SVM)	**15**	85.91	540	**89.72**	1233	88.31	41	79.89	1474	69.73
mnist8m (SVM)	**321**	98.94	8112	**99.45**	2414	98.60	-	-	-	-
webspam (LR)	**1679**	92.01	2131	**99.07**	4417	98.96	-	-	-	-
cifar (LR)	**471**	83.37	758	**88.14**	2115	87.07	-	-	-	-

presented in Table 5.2, showing that our algorithm is faster than distributed SGD algorithm. Note that PSVM cannot be directly applied to kernel logistic regression.

5.4 Summary of the Contribution

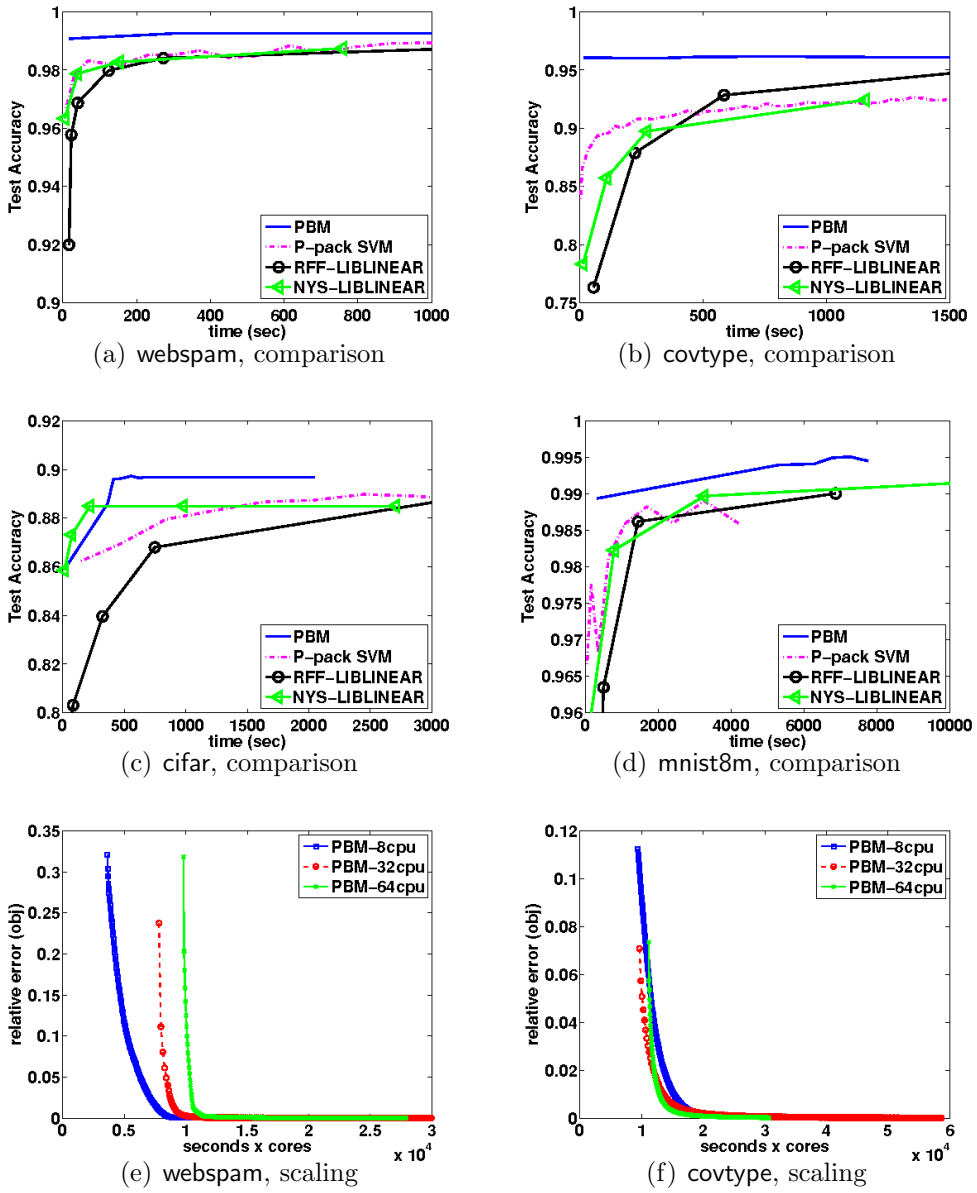
We have developed an automatic differentiation software to compute the gradient and Hessian for matrix functions (as discussed in Section 5.1), and use the special structure of the Hessian to optimize matrix functions. The software can be downloaded at `https://github.com/pkambadu/AMD`. The paper for distributed divide-and-conquer kernel SVM has been submitted to a conference.



(a) webspam obj (b) webspam accuracy

(c) covtype obj (d) covtype accuracy

Figure 5.1: Comparison of different variances of PBM. PBM-random uses random partition of data points, which performs the worst. PBM-cluster use kmeans partitioning and converges much faster than PBM-random. PBM-localPred further applies a local prediction heuristic on top of PBM-cluster to get better prediction accuracy in the early stage.

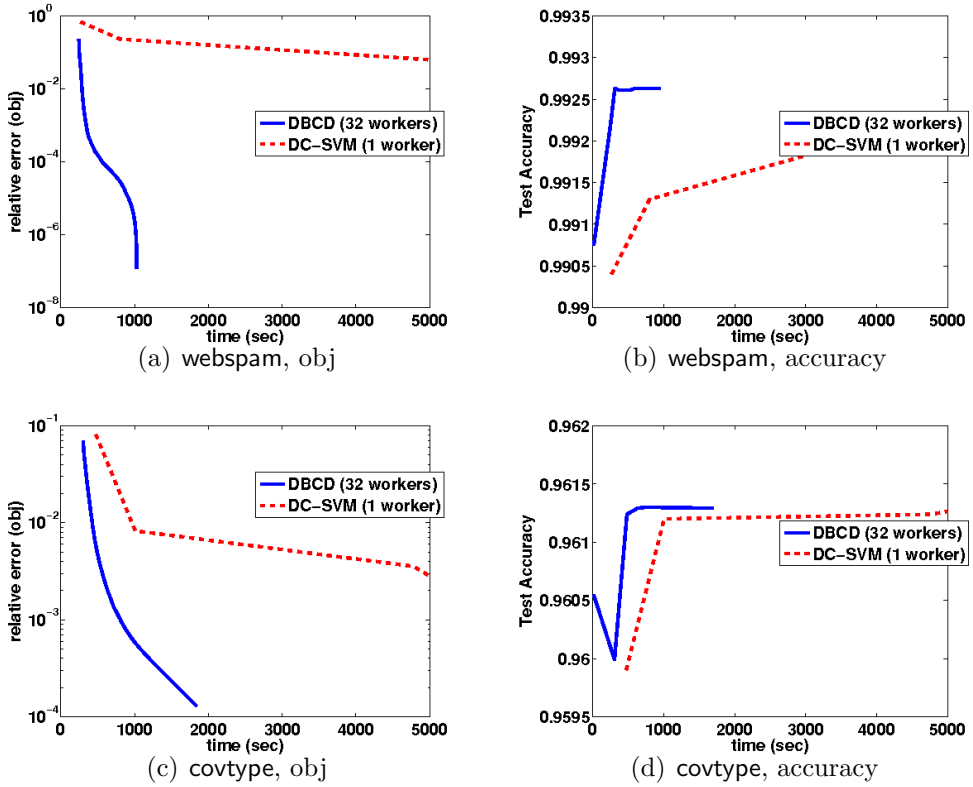


(a) webspam, comparison (b) covtype, comparison

(c) cifar, comparison (d) mnist8m, comparison

(e) webspam, scaling (f) covtype, scaling

Figure 5.2: (a)-(d): Comparison with other distributed SVM solvers using 32 workers. Markers for RFF-LIBLINEAR and NYS-LIBLINEAR are obtained by varying a number of random features and landmark points respectively. (e)-(f): The objective function of PBM as a function of computation time (time in seconds $\times$ the number of workers), when the number of workers is varied. Results show that PBM has good scalability.



(a) webspam, obj (b) webspam, accuracy

(c) covtype, obj (d) covtype, accuracy

Figure 5.3: Comparison with DC-SVM (a sequential kernel SVM solver).

Chapter 6

Theoretical Analysis for In-exact Proximal Gradient and Newton Methods

We have discussed several techniques to speedup optimization algorithms by exploiting structure of problem, model, and data distribution. In this chapter, we prove the global convergence and local convergence rate when applying these techniques. We consider two types of objective functions: (1) Functions that admit a global error bound (see Definition 10) and (2) Functions that admit a constant nullspace strong convexity (see Definition 12). These two assumptions cover most machine learning objective functions that may not be strongly convex (for example, SVM dual problem with a positive semidefinite kernel matrix, ℓ_1-regularized empirical risk minimization problems in a high-dimensional setting, and many others). We prove the convergence rate of the techniques proposed in this thesis, including:

- **Convergence rate of in-exact proximal Newton method.** We showed in Section 5.1 that the Hessian matrix of empirical risk minimization problems and simple matrix function optimization problems have special structures. In order to exploit the structure, we proposed a family of proximal Newton methods to solve the problem. At each

iteration, we form a quadratic approximation around the current solution using the Hessian matrix, and the resulting problem can be solved efficiently by coordinate descent or other optimization algorithms.

Since there is no close form solution of the quadratic subproblems, in practice we have to apply an iterative solver with a certain stopping condition, so only an "approximate" solution can be computed at each outer iteration. Unfortunately, most existing analysis focused on "exact" proximal Newton methods where they assume the subproblems are solved exactly. In this chapter we prove the following global and local convergence rate for in-exact proximal Newton methods:

1. If the subproblem solver $\mathcal{S}$ has global linear convergence, the proximal Newton method will also have global linear convergence if we apply $\mathcal{S}$ with ≥ 1 iteration for solving each quadratic subproblem.
2. If the subproblem solver $\mathcal{S}$ has global linear convergence, and we apply $\mathcal{S}$ with a fixed number of iterations for solving each quadratic subproblem, then we can obtain an ϵ-accurate solution using a total number of $O(\log(\frac{1}{\epsilon}))$ inner iterations.
3. When the sequence of stopping conditions at each outer iteartion $\{\eta_t\}$ converges to 0, then the proximal Newton method has an asymptotic super-linear convergence rate in terms of outer iterations.

- **Convergence rate of proximal Newton method with active sub-**

space selection. In Section 5.2, we discussed a general active subspace selection technique for exploiting model structure. At each proximal Newton iteration, we partition the solution space into active subspace and in-active subspace, and then we only search for the optimal solution within the active subspace. In this section, we show that the in-exact proximal Newton method with active subspace selection has the same linear convergence rate with the original in-exact proximal Newton method. Therefore, we can apply active subspace selection in proximal Newton methods to speedup the algorithm without changing the convergence rate.

- **Convergence rate of distributed proximal Newton methods.** In Section 5.3, we show a general framework of distributed proximal Newton methods. Using the analysis in this section we can prove the global linear convergence for these distributed proximal Newton methods.

In order to show the above theoretical results, we discuss a general algorithmic framework for in-exact proximal gradient and Newton methods in Section 6.1. We discuss the global linear convergence in Section 6.2, and local super-linear convergence in Section 6.3. Finally we will show in Section 6.4 that the in-exact proximal Newton (with or without active subspace selection) and the distributed proximal Newton methods are special cases of this general framework, therefore the global convergence rates are guaranteed using our analysis.

6.1 A Unified Algorithmic Framework for Composite Minimization Problems

We focus on the following composite minimization problem:

$$\underset{\boldsymbol{x}\in\mathbb{R}^d}{\operatorname{argmin}} \big\{ g(\boldsymbol{x}) + h(\boldsymbol{x}) \big\} := f(\boldsymbol{x}), \tag{6.1}$$

where $g(\boldsymbol{x})$ is a smooth convex function, and $h(\boldsymbol{x})$ is convex but not necessarily differentiable. Most of the machine learning algorithms can be written in this way. For example, in regularized loss minimization problems, $g(\boldsymbol{x})$ is the loss function that measures the quality of the model parameters $\boldsymbol{x}$ based on the data, and $h(\boldsymbol{x})$ is the regularization term that measures the model complexity.

We discuss a general class of descent algorithms for minimizing the composite problem (6.1). We use $\boldsymbol{x}$ to denote the current solution, and $\boldsymbol{x}_+$ to denote the next (outer) iteration. At each outer iteration, we obtain the descent direction by minimizing the following approximate function:

$$\hat{f}_{\boldsymbol{x},H}(\boldsymbol{d}) = \nabla g(\boldsymbol{x})^T \boldsymbol{d} + \frac{1}{2} \boldsymbol{d}^T H \boldsymbol{d} + h(\boldsymbol{x}+\boldsymbol{d}) - h(\boldsymbol{x}). \tag{6.2}$$

This subproblem is assumed to be solved exactly in "exact" proximal gradient or Newton methods:

$$\boldsymbol{d}^* = \underset{\boldsymbol{d}}{\operatorname{argmin}}\, \hat{f}_{\boldsymbol{x},H}(\boldsymbol{d}). \tag{6.3}$$

When $H = I$, this is equivalent to a proximal gradient operation, and when $H = \nabla^2 g(\boldsymbol{x})$ this will become a proximal Newton operation.

However, in many real applications we cannot solve subproblems exactly; therefore the quadratic subproblems $\hat{f}_{\boldsymbol{x},H}(\cdot)$ are usually solved "approximately" at each outer iteration. We use $\hat{\boldsymbol{d}}$ to denote the approximate solution

of (6.2), and we will define the "quality" of the approximate solution in Section 6.1.1.

After computing the direction $\hat{\boldsymbol{d}}$, we then update the solution by

$$\boldsymbol{x}_+ \leftarrow \boldsymbol{x} + \alpha\hat{\boldsymbol{d}}, \tag{6.4}$$

where α is computed by finding the largest number in $\{1, \beta, \beta^2, \dots\}$ such that $\boldsymbol{x} + \alpha\hat{\boldsymbol{d}}$ satisfies the following sufficient decrease condition:

$$f(\boldsymbol{x}_+) - f(\boldsymbol{x}) \leq \sigma\alpha\gamma, \quad \text{where} \quad \gamma = \nabla g(\boldsymbol{x})^T\hat{\boldsymbol{d}} + h(\boldsymbol{x} + \hat{\boldsymbol{d}}) - h(\boldsymbol{x}), \tag{6.5}$$

and $\sigma \in (0, 1)$ is a constant. Note that for the asymptotic super-linear convergence we require $\sigma < 0.5$.

This framework can be summarized in Algorithm 8.

Algorithm 8: Inexact Proximal Gradient (Newton) Method for Composite Minimization Problems.

Input : Objective function f, $H_t \in \mathbb{R}^{n\times n}$ at each iteration, line search parameters $\sigma \in (0, \frac{1}{2}), \beta \in (0, 1)$.

Output: Sequence $\{\boldsymbol{x}_t\}$ that converges to $\operatorname{argmin}_{\boldsymbol{x}} f(\boldsymbol{x})$.

1 **for** $t = 0, 1, \dots$ **do**

2 Compute an approximate solution $\hat{\boldsymbol{d}}$ of the subproblem $\hat{f}_{\boldsymbol{x}_t, H_t}(\boldsymbol{d})$.

3 Choose α to be the largest element of $\{\beta^j\}_{j=0,1,\dots}$ satisfying

$$f(\boldsymbol{x}_t + \alpha\hat{\boldsymbol{d}}) \leq f(\boldsymbol{x}_t) + \sigma\alpha\gamma$$

where $\gamma := \nabla g(\boldsymbol{x})^T\hat{\boldsymbol{d}} + h(\boldsymbol{x} + \hat{\boldsymbol{d}}) - h(\boldsymbol{x})$.

6.1.1 Quality of the approximate solution.

We make the following assumption on the "quality" of the approximate solution. The first assumption measures the quality of solution by the objective function, and the second assumption measures the quality by the magnitude of proximal gradient.

Objective Function Reduction Subproblem Solvers.

Assumption 8. *An inexact solver for minimizing $\hat{f}_{\boldsymbol{x},H}(\cdot)$ achieves an "η-obj reduction" if the inexact solution $\hat{\boldsymbol{d}}$ satisfies*

$$E[\hat{f}_{\boldsymbol{x},H}(\hat{\boldsymbol{d}})] - \hat{f}_{\boldsymbol{x},H}(\boldsymbol{d}^*) \leq \eta\big(\hat{f}_{\boldsymbol{x},H}(\boldsymbol{0}) - \hat{f}_{\boldsymbol{x},H}(\boldsymbol{d}^*)\big), \tag{6.6}$$

for some constant $\eta < 1$. Note that $\boldsymbol{d}^*$ is a minimizer of $\hat{f}_{\boldsymbol{x},H}(\cdot)$.

This assumption requires the subproblem solver to reduce the objective function by a linear rate, and a larger η indicates a more accurate subproblem solver. Many first order methods have global linear convergence rates, which means they can achieve linear improvement in objective function using 1 iteration. The constant η can thus be simply controlled by varying number of iterations of the subproblem solver. For example, [32] uses an increasing number of iterations, and [74] showed a sub-linear convergence rate by using this strategy. We will show a super-linear convergence rate if $\eta \to 1$, and a global linear convergence rate if η is a constant.

Gradient Reduction Subproblem Solvers. Assumption 8 cannot be easily measured for some subproblem solvers. Therefore, we also discuss another assumption that measures quality of solution by magnitude of proximal gradient.

We follow the notations used in [50]. For any given function $f(\boldsymbol{x}) = g(\boldsymbol{x}) + h(\boldsymbol{x})$, we define

$$G_{f,\alpha}(\boldsymbol{u}) = \frac{1}{\alpha}(u - \text{prox}_{\alpha h}(\boldsymbol{u} - \alpha\nabla g(\boldsymbol{u}))),$$

where the prox operator is defined by

$$\text{prox}_h(\boldsymbol{u}) = \underset{\boldsymbol{v}}{\text{argmin}}\, \frac{1}{2}\|\boldsymbol{v} - \boldsymbol{u}\|^2 + h(\boldsymbol{v}).$$

Similarly, for the quadratic subproblem

$$\hat{f}_{\boldsymbol{x}}(\boldsymbol{u}) = \hat{g}_{\boldsymbol{x}}(\boldsymbol{u}) + h(\boldsymbol{u}), \quad \hat{g}_{\boldsymbol{x}}(\boldsymbol{u}) := g(\boldsymbol{x}) + \nabla g(\boldsymbol{x})^T(\boldsymbol{u}-\boldsymbol{x}) + \frac{1}{2}(\boldsymbol{u}-\boldsymbol{x})^T\nabla^2 g(\boldsymbol{x})(\boldsymbol{u}-\boldsymbol{x}), \tag{6.7}$$

we define

$$G_{\hat{f}_{\boldsymbol{x}},\alpha}(\boldsymbol{u}) = \frac{1}{\alpha}(\boldsymbol{u} - \text{prox}_{\alpha h}(\boldsymbol{u} - \alpha\nabla\hat{g}(\boldsymbol{u}))).$$

For simplicity, we define $G_f := G_{f,1}$ when the step size is 1.

Assumption 9. *An inexact solver for minimizing $\hat{f}_{\boldsymbol{x}}(\cdot)$ achieves an "η-gradient reduction" if the inexact solution $\boldsymbol{x}_+$ satisfies*

$$\|G_{\hat{f}_{\boldsymbol{x}},\frac{1}{L}}(\boldsymbol{x}_+)\| \leq (1-\eta)\|G_{\hat{f}_{\boldsymbol{x}},\frac{1}{L}}(\boldsymbol{x})\|. \tag{6.8}$$

Note that by definition $G_{\hat{f}_{\boldsymbol{x}},\frac{1}{L}}(\boldsymbol{x}) = G_{f,\frac{1}{L}}(\boldsymbol{x})$, which is the initial value of composite gradient in the beginning at each outer iteration. Since $G_{\hat{f}_{\boldsymbol{x}},\frac{1}{L}}(\boldsymbol{x}_+)$ can usually be computed efficiently, this assumption can be used as a stopping criteria for the subproblem solver.

6.1.2 Assumption on the objective function.

We show the convergence for two types of objective functions.

Functions with a Global Error Bound. We first describe the following definition of the global error bound defined in [55, 80, 84]:

Definition 10. *The problem $f(\boldsymbol{x}) := g(\boldsymbol{x}) + h(\boldsymbol{x})$ admits a "global error bound" if there is a constant κ such that*

$$\|\boldsymbol{x} - P_S(\boldsymbol{x})\| \leq \kappa \|\boldsymbol{d}_I(\boldsymbol{x})\|, \tag{6.9}$$

where $P_S(\cdot)$ is the Euclidean projection to the set S of optimal solutions, and $\boldsymbol{d}_I(\boldsymbol{x})$ is defined by

$$\boldsymbol{d}_I(\boldsymbol{x}) = \operatorname*{argmin}_{\boldsymbol{d}} \hat{f}_{\boldsymbol{x},I}(\boldsymbol{d}).$$

The algorithm satisfies a "global error bound from the beginning" if (6.9) *holds for the level set $\{\boldsymbol{x} : f(\boldsymbol{x}) \leq f(\boldsymbol{x}^0)\}$.*

Although the above definition is widely used, eq (6.9) is often hard to verify. Some composite functions that satisfy the global error bounds have been proved in the literature [80, 84, 62]:

Proposition 11. *The following composite functions satisfy Definition 10:*

- $g(\cdot)$ *is strongly convex and* $h(\cdot)$ *is convex.*
- $g(\boldsymbol{x}) = \tilde{g}(P\boldsymbol{x}) + \boldsymbol{b}^T\boldsymbol{x}$ *where* P *is a constant matrix,* $\tilde{g}$ *is strongly convex, and* $h(\boldsymbol{x})$ *is an indicator function of a polyhedral set.*

Functions with Constant Nullspace Strong Convexity. In addition to functions with global error bounds, we further consider the following Constant Nullspace Strong Convexity (CNSC) introduced in [87]. This assumption is easier to verify and can be naturally applied to empirical risk minimization problems.

Definition 12. *The problem* $f(\boldsymbol{x}) := g(\boldsymbol{x}) + h(\boldsymbol{x})$ *admits a Constant Nullspace Strong Convexity (CNSC) if* $g(\boldsymbol{x})$ *is twice differentiable, and there is a constant vector space* $\mathcal{T} \subseteq \mathbb{R}^d$ *such that the Hessian matrix* $\nabla^2 g(\boldsymbol{x})$ *satisfies*

$$\boldsymbol{u}^T(\nabla^2 g(\boldsymbol{x}))\boldsymbol{u} \geq m\|\boldsymbol{u}\|^2 \quad \forall \boldsymbol{u} \in \mathcal{T}, \boldsymbol{x} \in \mathbb{R}^d, \tag{6.10}$$

for some $m > 0$*, and*

$$\boldsymbol{u}^T\nabla^2 g(\boldsymbol{x})\boldsymbol{u} = 0 \quad \forall \boldsymbol{u} \in \mathcal{T}^\perp, \boldsymbol{x} \in \mathbb{R}^d. \tag{6.11}$$

A function satisfies CNSC from the beginning if (6.10) *and* (6.11) *are satisfied in the level set* $\{\boldsymbol{x} : f(\boldsymbol{x}) \leq f(\boldsymbol{x}^0)\}$.

It is easy to verify if a function satisfies CNSC. For example, we list several important application:

Proposition 13. *The following composite functions satisfy Definition 12:*

- $g(\cdot)$ *is strongly convex and* $h(\cdot)$ *is convex.*
- $g(\cdot) = \tilde{g}(P\boldsymbol{x})$ *where* $\tilde{g}$ *is a strongly convex function;* $h(\cdot)$ *is any convex function.*

The first case is easy to show, and the second case can be shown by taking the Hessian matrix of $g(\boldsymbol{x})$:

$$\nabla^2 g(\boldsymbol{x}) = P^T D P, \text{ where } D = \nabla^2 \tilde{g}(P\boldsymbol{x}).$$

If $\tilde{g}$ is strongly convex, then D is positive definite, which implies the CNSC condition with $\mathcal{T}^\perp := \text{null}(P)$. Note that the widely-used empirical risk minimization problems have the following form:

$$\underset{\boldsymbol{x}}{\text{argmin}} \sum_{i=1}^{n} \ell_i(\boldsymbol{x}^T \boldsymbol{a}_i, y_i) + h(\boldsymbol{x}),$$

where each $\boldsymbol{a}_i$ is a training data and y_i is the corresponding label. This type of problems clearly satisfies the second case of Proposition 13 if each ℓ_i is strongly convex in the level set. Examples include logistic loss and squared loss.

Other notations and constants:

- $\bar{\boldsymbol{x}}$: the closest optimal solution to $\boldsymbol{x}$.
- σ: constant in the line search condition (6.5).

- η: constant for inexact solver (Definition 8), $\eta \in [0, 1)$.
- κ: constant for global error bound (Definition 10), $\kappa > 0$.
- $\boldsymbol{d}_H^*$: optimal solution of (6.2), $\hat{\boldsymbol{d}}_H$: approximate solution of (6.2). satisfies Assumption 8 or 9.
- $\boldsymbol{d}_I$: optimal solution of (6.2) with $H = I$.
- $\|\boldsymbol{d}\|_H = \sqrt{\boldsymbol{d}^T H \boldsymbol{d}}$.
- $P_{\mathcal{T}}(\boldsymbol{x})$: the Euclidean projection of $\boldsymbol{x}$ onto the vector space $\mathcal{T} \in \mathbb{R}^d$.
- L_g: we assume $g(\cdot)$ is differentiable and $\nabla g(\cdot)$ is L_g-Lipchitz continuous.
- $E[f(\boldsymbol{x}_+)]$: the expectation of the objective function at the next iteration. We allow the subproblem solvers to be randomized algorithms, so $f(\boldsymbol{x}_+)$ can be a random variable.

6.2 Global Linear Convergence Rate for In-exact Proximal Gradient and Newton Methods

We first discuss the global convergence rate for in-exact proximal Gradient and Newton methods (Algorithm 8). We prove the linear convergence for functions with global error bound in Theorem 15 in Section 6.2.2, and linear convergence for functions with CNSC in Theorem 16 in Section 6.2.3. In this section we focus on the subproblem solvers with an η-obj reduction (Assumption 8). Guarantee when approximate solutions have an η-gradient reduction will be discussed in Section 6.3.

We make an assumption on H_t used at each iteration:

Assumption 14. *We consider slightly different assumptions for the matrices H used in the algorithm.*

1. *For functions admit global error bound (Definition 10), we assume $MI \succeq H \succeq mI$.*
2. *For functions satisfy CNSC (Definition 12), we assume $MI \succeq H$, $H\boldsymbol{u} = 0\ \forall \boldsymbol{u} \in \mathcal{T}^{\perp}$, and*

$$\boldsymbol{u}^T H \boldsymbol{u} \geq m \quad \text{if } \boldsymbol{u} \in \mathcal{T}.$$

6.2.1 Lemmas

In order to prove the global convergence rate, we first derive the following lemmas.

Lemma 4. *If $h(\cdot)$ is a convex function, then*

$$h(\boldsymbol{x} + \alpha \boldsymbol{d}) - h(\boldsymbol{x}) \leq \alpha(h(\boldsymbol{x} + \boldsymbol{d}) - h(\boldsymbol{x}))$$

for any $\alpha \in [0, 1]$, $\boldsymbol{x}, \boldsymbol{d} \in \mathbb{R}^d$.

Proof.

$$\begin{aligned} h(\boldsymbol{x} + \alpha \boldsymbol{d}) - h(\boldsymbol{x}) &= h(\alpha(\boldsymbol{x} + \boldsymbol{d}) + (1 - \alpha)\boldsymbol{x}) - h(\boldsymbol{x}) \\ &\leq \alpha h(\boldsymbol{x} + \boldsymbol{d}) + (1 - \alpha)h(\boldsymbol{x}) - h(\boldsymbol{x}) \quad \text{(by convexity of } h(\cdot)) \\ &= \alpha(h(\boldsymbol{x} + \boldsymbol{d}) - h(\boldsymbol{x})). \end{aligned}$$

□

Lemma 5. *If the approximate step size $\hat{\boldsymbol{d}}$ satisfies Assumption 8, and the objective function satisfy Definition 10 or Definition 12, then the step size α in Algorithm 8 satisfies*

$$\alpha \geq \underline{\boldsymbol{\alpha}} := \frac{m}{L_g}(1-\sigma)\beta.$$

Proof. We first consider functions satisfy Definition 10. Note that in this case, $g(\cdot)$ may not be twice differentiable.

$$\begin{aligned}
& f(\boldsymbol{x}_+) - f(\boldsymbol{x}) \\
=& g(\boldsymbol{x}_+) - g(\boldsymbol{x}) + h(\boldsymbol{x}_+) - h(\boldsymbol{x}) \\
\leq& g(\boldsymbol{x}_+) - g(\boldsymbol{x}) + \alpha(h(\boldsymbol{x}+\hat{\boldsymbol{d}}) - h(\boldsymbol{x})) \qquad \text{(by Lemma 4)} \\
\leq& \nabla g(\boldsymbol{x})^T(\alpha\hat{\boldsymbol{d}}) + \int_0^1 \left(\left(\nabla g(\boldsymbol{x}+s(\alpha\hat{\boldsymbol{d}})) - \nabla g(\boldsymbol{x})\right)^T(\alpha\hat{\boldsymbol{d}})\right) ds + \alpha(h(\boldsymbol{x}+\hat{\boldsymbol{d}}) - h(\boldsymbol{x})) \qquad (6.12) \\
\leq& \alpha\left(\nabla g(\boldsymbol{x})^T\hat{\boldsymbol{d}} + h(\boldsymbol{x}+\hat{\boldsymbol{d}}) - h(\boldsymbol{x})\right) + \int_0^1 \|\nabla g(\boldsymbol{x}+s\alpha\hat{\boldsymbol{d}}) - \nabla g(\boldsymbol{x})\|\|\alpha\hat{\boldsymbol{d}}\| ds \\
\leq& \alpha\gamma + \int_0^1 L_g\|s\alpha\hat{\boldsymbol{d}}\|\|\alpha\hat{\boldsymbol{d}}\| ds \qquad \text{(by } L_g\text{-Lipchitz continuous of } \nabla g(\cdot)) \\
=& \alpha\gamma + \frac{L_g\alpha^2\|\hat{\boldsymbol{d}}\|^2}{2} \qquad (6.13)
\end{aligned}$$

By the definition of "Objective Reduction Subproblem Solvers" in (6.6), we have

$$\hat{f}_{\boldsymbol{x},H}(\hat{\boldsymbol{d}}) \leq \hat{f}_{\boldsymbol{x},H}(\boldsymbol{0}) = 0.$$

Also, $\hat{f}_{\boldsymbol{x},H}(\hat{\boldsymbol{d}}) = \gamma + \frac{1}{2}\hat{\boldsymbol{d}}^T H\hat{\boldsymbol{d}}$ (from the definition of γ in (6.5)). Therefore,

$$\gamma + \frac{1}{2}\hat{\boldsymbol{d}}^T H\hat{\boldsymbol{d}} \leq 0,$$

and since H has lower bounded eigenvalue m (Assumption 14a),

$$\gamma \leq -\frac{1}{2}\hat{\boldsymbol{d}}^T H\hat{\boldsymbol{d}} \leq -\frac{m}{2}\|\hat{\boldsymbol{d}}\|^2. \tag{6.14}$$

Combining (6.13) and (6.14), we have

$$f(\boldsymbol{x}_+) - f(\boldsymbol{x}) \leq \alpha\gamma(1 - \frac{L_g\alpha}{m}).$$

Therefore, the line search condition (6.5) is satisfied for all $\alpha \leq (1-\sigma)m/L_g$. Since we try step sizes with $\alpha = \{1, \beta, \beta^2, \dots\}$, the step size selected by our algorithm will be larger than $(1-\sigma)m\beta/L_g$.

For objective functions that satisfy CNSC (Definition 12), since $g(\cdot)$ is twice differentiable, the integral in (6.12) can be rewritten as

$$\begin{aligned}
&\int_0^1 \left(\left(\nabla g(\boldsymbol{x} + s(\alpha\hat{\boldsymbol{d}})) - \nabla g(\boldsymbol{x})\right)^T(\alpha\hat{\boldsymbol{d}})\right) ds \\
=&\alpha^2 \int_0^1 s\hat{\boldsymbol{d}}^T\nabla^2 g(\boldsymbol{x}_s)\hat{\boldsymbol{d}} ds \quad \text{where } \boldsymbol{x}_s \text{ is some vector in line}(\boldsymbol{x}, \boldsymbol{x} + s\alpha\hat{\boldsymbol{d}}) \\
\leq&\alpha^2 \int_0^1 \alpha L_g \|P_{\mathcal{T}}(\hat{\boldsymbol{d}})\|^2 s ds \\
=&\frac{\alpha^2\|P_{\mathcal{T}}(\hat{\boldsymbol{d}})\|^2 L_g}{2}.
\end{aligned}$$

Therefore, eq (6.13) can be re-written as

$$f(\boldsymbol{x}_+) - f(\boldsymbol{x}) \leq \alpha\gamma + \frac{L_g\alpha^2\|P_{\mathcal{T}}(\hat{\boldsymbol{d}})\|^2}{2}.$$

Also, due to the definition of H in Assumption 14b, eq (6.14) will become

$$\gamma \leq -\frac{1}{2}\hat{\boldsymbol{d}}^T H\hat{\boldsymbol{d}} \leq -\frac{m}{2}\|P_{\mathcal{T}}(\hat{\boldsymbol{d}})\|^2.$$

Combining the above two inequalities we get

$$f(\boldsymbol{x}_+) - f(\boldsymbol{x}) \leq \alpha\gamma(1 - \frac{L_g\alpha}{m}),$$

and this proves the theorem for CNSC functions. □

Lemma 6. *The optimal direction* $\boldsymbol{d}_H^* = \operatorname{argmin}_{\boldsymbol{d}} \hat{f}_{\boldsymbol{x},H}(\boldsymbol{d})$ *satisfies*

$$\nabla g(\boldsymbol{x})^T \boldsymbol{d}_H^* + h(\boldsymbol{x} + \boldsymbol{d}_H^*) - h(\boldsymbol{x}) \leq -\|\boldsymbol{d}_H^*\|_H^2.$$

Any approximate direction $\hat{\boldsymbol{d}}_H$ *that satisfies Assumption 8 has the following property:*

$$E[\gamma] = E[\nabla g(\boldsymbol{x})^T \hat{\boldsymbol{d}}_H + h(\boldsymbol{x} + \hat{\boldsymbol{d}}_H) - h(\boldsymbol{x})] \leq -\frac{\eta}{2}\|\boldsymbol{d}_H^*\|_H^2.$$

Proof. Since $\boldsymbol{d}_H^*$ is the optimal solution of $\hat{f}_{\boldsymbol{x},H}(\boldsymbol{d})$, we have

$$\begin{aligned}
&\nabla g(\boldsymbol{x})^T \boldsymbol{d}_H^* + \frac{1}{2}(\boldsymbol{d}_H^*)^T H \boldsymbol{d}_H^* + h(\boldsymbol{x} + \boldsymbol{d}_H^*) - h(\boldsymbol{x}) \\
\leq &\nabla g(\boldsymbol{x})^T (t\boldsymbol{d}_H^*) + \frac{1}{2}(t\boldsymbol{d}_H^*)^T H (t\boldsymbol{d}_H^*) + h(\boldsymbol{x} + t\boldsymbol{d}_H^*) - h(\boldsymbol{x}) \\
\leq &t\nabla g(\boldsymbol{x})^T \boldsymbol{d}_H^* + \frac{1}{2}t^2(\boldsymbol{d}_H^*)^T H \boldsymbol{d}_H^* + t(h(\boldsymbol{x} + \boldsymbol{d}_H^*) - h(\boldsymbol{x})) \quad \text{(by Lemma 4)}
\end{aligned}$$

Therefore,

$$\begin{aligned}
(1-t)\nabla g(\boldsymbol{x})^T \boldsymbol{d}_H^* + \frac{1}{2}(1-t^2)(\boldsymbol{d}_H^*)^T H \boldsymbol{d}_H^* + (1-t)(h(\boldsymbol{x} + \boldsymbol{d}_H^*) - h(\boldsymbol{x})) \leq 0 \\
\nabla g(\boldsymbol{x})^T \boldsymbol{d}_H^* + \frac{1}{2}(1+t)(\boldsymbol{d}_H^*)^T H \boldsymbol{d}_H^* + h(\boldsymbol{x} + \boldsymbol{d}_H^*) - h(\boldsymbol{x}) \leq 0.
\end{aligned}$$

Taking $t \uparrow 1$ we get

$$\nabla g(\boldsymbol{x})^T \boldsymbol{d}_H^* + h(\boldsymbol{x} + \boldsymbol{d}_H^*) - h(\boldsymbol{x}) \leq -(\boldsymbol{d}_H^*)^T H \boldsymbol{d}_H^*. \tag{6.15}$$

To prove the property for the approximate solution $\hat{\boldsymbol{d}}_H$, by Assumption 8,

$$\begin{aligned}
&E\left[\nabla g(\boldsymbol{x})^T\hat{\boldsymbol{d}}_H + \frac{1}{2}\hat{\boldsymbol{d}}_H^T H\hat{\boldsymbol{d}}_H + h(\boldsymbol{x}+\hat{\boldsymbol{d}}_H) - h(\boldsymbol{x})\right]\\
\leq&\eta\left(\nabla g(\boldsymbol{x})^T\boldsymbol{d}_H^* + \frac{1}{2}(\boldsymbol{d}_H^*)^T H\boldsymbol{d}_H^* + h(\boldsymbol{x}+\boldsymbol{d}_H^*) - h(\boldsymbol{x})\right)\\
\leq&\eta\left(-\frac{1}{2}(\boldsymbol{d}_H^*)^T H\boldsymbol{d}_H^*\right) \quad \text{(by (6.15))}.
\end{aligned}$$

Therefore,

$$E[\gamma] \leq -\frac{\eta}{2}\|\boldsymbol{d}_H^*\|_H^2 - \frac{1}{2}\hat{\boldsymbol{d}}_H^T H\hat{\boldsymbol{d}}_H \leq -\frac{\eta}{2}\|\boldsymbol{d}_H^*\|_H^2.$$

□

Lemma 7. *If $MI \succeq H$, the optimal direction $\boldsymbol{d}_H^*$ of* $\operatorname{argmin}_{\boldsymbol{d}} \hat{f}_{\boldsymbol{x},H}(\boldsymbol{d})$ *satisfies*

$$\|\boldsymbol{d}_H^*(\boldsymbol{x})\| \geq \frac{1}{1+M}\|\boldsymbol{d}_I(\boldsymbol{x})\|.$$

Proof. Since $\boldsymbol{d}_H^*$ is the optimal solution of $\hat{f}_{\boldsymbol{x},H}(\boldsymbol{d})$, by the optimality condition,

$$0 \in \nabla g(\boldsymbol{x}) + H\boldsymbol{d}_H^* + \partial h(\boldsymbol{x}+\boldsymbol{d}_H^*). \tag{6.16}$$

And (6.16) is also the optimality condition of the following function:

$$p(\boldsymbol{d}) := (\nabla g(\boldsymbol{x}) + H\boldsymbol{d}_H^*)^T\boldsymbol{d} + h(\boldsymbol{x}+\boldsymbol{d}).$$

Therefore, we have

$$\boldsymbol{d}_H^* \in \operatorname*{argmin}_{\boldsymbol{d}}(\nabla g(\boldsymbol{x}) + H\boldsymbol{d}_H^*)^T\boldsymbol{d} + h(\boldsymbol{x}+\boldsymbol{d}) \tag{6.17}$$

$$\boldsymbol{d}_I \in \operatorname*{argmin}_{\boldsymbol{d}}(\nabla g(\boldsymbol{x}) + \boldsymbol{d}_I)^T\boldsymbol{d} + h(\boldsymbol{x}+\boldsymbol{d}). \tag{6.18}$$

Note that (6.18) can be shown by replacing H with I. By substituting $\boldsymbol{d}_I$ into (6.17) and $\boldsymbol{d}_H^*$ into (6.18) we have

$$(\nabla g(\boldsymbol{x}) + H\boldsymbol{d}_H^*)^T\boldsymbol{d}_H^* + h(\boldsymbol{x}+\boldsymbol{d}_H^*) \leq (\nabla g(\boldsymbol{x}) + H\boldsymbol{d}_H^*)^T\boldsymbol{d}_I + h(\boldsymbol{x}+\boldsymbol{d}_I)$$
$$(\nabla g(\boldsymbol{x}) + \boldsymbol{d}_I)^T\boldsymbol{d}_I + h(\boldsymbol{x}+\boldsymbol{d}_I) \leq (\nabla g(\boldsymbol{x}) + \boldsymbol{d}_I)^T\boldsymbol{d}_H^* + h(\boldsymbol{x}+\boldsymbol{d}_H^*).$$

Sum the above two inequalities we get

$$(\nabla g(\boldsymbol{x})+H\boldsymbol{d}_H^*)^T\boldsymbol{d}_H^*+(\nabla g(\boldsymbol{x})+\boldsymbol{d}_I)^T\boldsymbol{d}_I \leq (\nabla g(\boldsymbol{x})+H\boldsymbol{d}_H^*)^T\boldsymbol{d}_I+(\nabla g(\boldsymbol{x})+\boldsymbol{d}_I)^T\boldsymbol{d}_H^*.$$

Therefore,

$$(\boldsymbol{d}_H^*)^T H\boldsymbol{d}_H^* - (\boldsymbol{d}_H^*)^T H\boldsymbol{d}_I - \boldsymbol{d}_I^T\boldsymbol{d}_H^* + \boldsymbol{d}_I^T\boldsymbol{d}_I \leq 0$$
$$(\boldsymbol{d}_H^*)^T H\boldsymbol{d}_H^* - (\boldsymbol{d}_H^*)^T(H+I)\boldsymbol{d}_I + \boldsymbol{d}_I^T\boldsymbol{d}_I \leq 0$$
$$\big\|\boldsymbol{d}_I - \frac{(H+I)}{2}\boldsymbol{d}_H^*\big\|^2 - (\boldsymbol{d}_H^*)^T\big(\frac{H+I}{2}\big)^2\boldsymbol{d}_H^* + (\boldsymbol{d}_H^*)^T H\boldsymbol{d}_H^* \leq 0$$

As a result,

$$\|\boldsymbol{d}_I - \frac{(H+I)}{2}\boldsymbol{d}_H^*\|^2 \leq \frac{1}{4}\|(H+I)\boldsymbol{d}_H^*\|^2 - (\boldsymbol{d}_H^*)^T H\boldsymbol{d}_H^*$$
$$\|\boldsymbol{d}_I - \frac{(H+I)}{2}\boldsymbol{d}_H^*\| \leq \frac{1}{2}\|(H+I)\boldsymbol{d}_H^*\|$$
$$\|\boldsymbol{d}_I\| - \|\frac{(H+I)}{2}\boldsymbol{d}_H^*\| \leq \frac{1}{2}\|(H+I)\boldsymbol{d}_H^*\|$$

Since $MI \succeq H$, we have

$$\|\boldsymbol{d}_I\| \leq \|(H+I)\boldsymbol{d}_H^*\| \leq (1+M)\|\boldsymbol{d}_H^*\|.$$

□

Lemma 8. *If the objective function satisfies Definition 10 and H_t satisfies Assumption 14a, or objective function satisfies Definition 12 and H_t satisfies Assumption 14b, then*

$$E[f(\boldsymbol{x}_+)] - f(\boldsymbol{x}) \leq -\frac{\sigma\bar{\alpha}\eta}{2}\|\boldsymbol{d}_H^*\|_H^2.$$

Proof.

$$\begin{aligned} E[f(\boldsymbol{x}_+)] - f(\boldsymbol{x}) &\leq \sigma\alpha\gamma \quad \text{(line search condition)} \\ &\leq \sigma\bar{\alpha}\gamma \quad \text{(by Lemma 5)} \\ &\leq -\frac{\sigma\bar{\alpha}\eta}{2}\|\boldsymbol{d}_H^*\|_H^2 \quad \text{(by Lemma 6).} \end{aligned}$$

□

6.2.2 Global Linear Convergence for Functions with Global Error Bound

Theorem 15. *Assume the objective function admits a global error bound from the beginning (Definition 10), the H matrix used at each iteration satisfies Assumption 14a, and the subproblem solver has linear improvement in objective function (Assumption 8). Then the in-exact proximal Newton method has a global linear convergence rate:*

$$E[f(\boldsymbol{x}_+)] - f(\bar{\boldsymbol{x}}) \leq \frac{C}{1+C}\Big(f(\boldsymbol{x}) - f(\bar{\boldsymbol{x}})\Big),$$

where $\bar{\boldsymbol{x}}$ is an optimal solution and

$$C = \frac{2L_g}{m\sigma\bar{\alpha}}\big(1 + \kappa^2\frac{1+M}{\sqrt{\eta}}\big) + \frac{1}{\sigma\bar{\alpha}\eta} + \frac{\kappa^2 M(1+M)}{m\sigma\bar{\alpha}\eta}.$$

Proof. By Mean Value Theorem,

$$\begin{aligned} f(\boldsymbol{x}_+) - f(\bar{\boldsymbol{x}}) &= g(\boldsymbol{x}_+) - g(\bar{\boldsymbol{x}}) + h(\boldsymbol{x}_+) - h(\bar{\boldsymbol{x}}) \\ &= \nabla g(\psi)^T(\boldsymbol{x}_+ - \bar{\boldsymbol{x}}) + h(\boldsymbol{x}_+) - h(\bar{\boldsymbol{x}}), \end{aligned}$$

where $\psi = t\boldsymbol{x}_+ + (1-t)\bar{\boldsymbol{x}}$ for some $t \in [0,1]$. Therefore we have

$$f(\boldsymbol{x}_+) - f(\bar{\boldsymbol{x}}) \tag{6.19}$$

$$\begin{aligned} =&\Big(\nabla g(\psi) - \nabla g(\boldsymbol{x})\Big)^T(\boldsymbol{x}_+ - \bar{\boldsymbol{x}}) + \nabla g(\boldsymbol{x})^T(\boldsymbol{x}_+ - \bar{\boldsymbol{x}}) + h(\boldsymbol{x}_+) - h(\bar{\boldsymbol{x}}) \\ &- \frac{1}{2}(\bar{\boldsymbol{x}} - \boldsymbol{x})^T H(\bar{\boldsymbol{x}} - \boldsymbol{x}) + \frac{1}{2}(\bar{\boldsymbol{x}} - \boldsymbol{x})^T H(\bar{\boldsymbol{x}} - \boldsymbol{x}) \\ =& \underbrace{\Big(\nabla g(\psi) - \nabla g(\boldsymbol{x})\Big)^T(\boldsymbol{x}_+ - \bar{\boldsymbol{x}})}_{①} + \underbrace{\Big(\nabla g(\boldsymbol{x})^T(\boldsymbol{x}_+ - \boldsymbol{x}) + h(\boldsymbol{x}_+) - h(\boldsymbol{x})\Big)}_{②} \\ &\underbrace{-\Big(\nabla g(\boldsymbol{x})^T(\bar{\boldsymbol{x}} - \boldsymbol{x}) + \frac{1}{2}(\bar{\boldsymbol{x}} - \boldsymbol{x})^T H(\bar{\boldsymbol{x}} - \boldsymbol{x}) + h(\bar{\boldsymbol{x}}) - h(\boldsymbol{x})\Big)}_{③} + \underbrace{\frac{1}{2}(\bar{\boldsymbol{x}} - \boldsymbol{x})^T H(\bar{\boldsymbol{x}} - \boldsymbol{x})}_{④} \end{aligned} \tag{6.20}$$

Now we want to bound each term in (6.20). To bound the second term, since $\boldsymbol{x}_+ = \boldsymbol{x} + \alpha\hat{\boldsymbol{d}}$,

$$\begin{aligned} ② &= \alpha\Big(\nabla g(\boldsymbol{x})^T(\hat{\boldsymbol{d}}) + h(\boldsymbol{x} + \hat{\boldsymbol{d}}) - h(\boldsymbol{x})\Big) \quad \text{(by Lemma 4)} \\ &\le \bar{\alpha}\gamma \quad \text{(by Lemma 5).} \\ &\le -\frac{\eta\bar{\alpha}}{2}\|\boldsymbol{d}_H^*\|_H^2 \quad \text{(by Lemma 6)} \\ &\le 0. \end{aligned} \tag{6.21}$$

For the third term, since $\boldsymbol{d}_H^*$ is the optimal solution of $\hat{f}_{\boldsymbol{x},H}(\boldsymbol{d})$, we have

$$
\begin{aligned}
③ &\le -\left(\nabla g(\boldsymbol{x})^T \boldsymbol{d}_H^* + \frac{1}{2}(\boldsymbol{d}_H^*)^T H \boldsymbol{d}_H^* + h(\boldsymbol{x}+\boldsymbol{d}_H^*) - h(\boldsymbol{x})\right) \\
&\le -\frac{1}{\eta} E\left[\nabla g(\boldsymbol{x})^T \hat{\boldsymbol{d}} + \frac{1}{2}\hat{\boldsymbol{d}}^T H \hat{\boldsymbol{d}} + h(\boldsymbol{x}+\hat{\boldsymbol{d}}) - h(\boldsymbol{x})\right] \quad \text{(by Assumption 8)} \\
&\le -\frac{1}{\eta} E[\gamma] \qquad \text{(by } H \succeq 0\text{)} \\
&\le \frac{1}{\eta\alpha\sigma} E\big[f(\boldsymbol{x}) - f(\boldsymbol{x}_+)\big] \quad \text{(by eq (6.5))} \\
&\le \frac{1}{\eta\bar{\alpha}\sigma} E\big[f(\boldsymbol{x}) - f(\boldsymbol{x}_+)\big] \quad \text{(by Lemma 5).}
\end{aligned} \tag{6.22}
$$

For the fourth term,

$$
\begin{aligned}
④ &= \frac{1}{2}(\bar{\boldsymbol{x}} - \boldsymbol{x})^T H (\bar{\boldsymbol{x}} - \boldsymbol{x}) \\
&\le \frac{M}{2}\|\bar{\boldsymbol{x}} - \boldsymbol{x}\|^2 \\
&\le \frac{\kappa M}{2}\|\boldsymbol{d}_I(\boldsymbol{x})\|^2 \quad \text{(by the global error bound (Definition 10))} \\
&\le \frac{\kappa^2 M(1+M)}{2}\|\boldsymbol{d}_H^*(\boldsymbol{x})\|^2 \quad \text{(by Lemma 7)} \\
&\le \frac{\kappa^2 M(1+M)}{2m}\|\boldsymbol{d}_H^*(\boldsymbol{x})\|_H^2 \quad \text{(since } H \succeq mI\text{)} \\
&\le \frac{\kappa^2 M(1+M)}{m\sigma\bar{\alpha}\eta} E\big[f(\boldsymbol{x}) - f(\boldsymbol{x}_+)\big] \quad \text{(by Lemma 8)}
\end{aligned} \tag{6.23}
$$

Finally, for the first term,

$$
\begin{aligned}
① &= (\nabla g(\psi) - \nabla g(\boldsymbol{x}))^T (\boldsymbol{x}_+ - \bar{\boldsymbol{x}}) \\
&\le L_g \|\boldsymbol{x}_+ - \boldsymbol{x}\| \|\boldsymbol{x}_+ - \bar{\boldsymbol{x}}\| \quad \text{(since } \nabla g(\cdot) \text{ is } L_g\text{-Lipchitz continuous)} \\
&\le L_g \|\boldsymbol{x}_+ - \boldsymbol{x}\| \left(\|\boldsymbol{x}_+ - \boldsymbol{x}\| + \|\boldsymbol{x} - \bar{\boldsymbol{x}}\|\right) \\
&= L_g \|\boldsymbol{x}_+ - \boldsymbol{x}\|^2 + L_g \|\boldsymbol{x}_+ - \boldsymbol{x}\| \|\boldsymbol{x} - \bar{\boldsymbol{x}}\|.
\end{aligned} \tag{6.24}
$$

Now we bound each term. First,

$$\|\boldsymbol{x}_+ - \boldsymbol{x}\| = \alpha\|\hat{\boldsymbol{d}}\| \le \frac{1}{\sqrt{m}}\|\hat{\boldsymbol{d}}\|_H \le \sqrt{\frac{-2\gamma}{m}},$$

where the last inequality is from $\gamma + \frac{1}{2}\|\hat{\boldsymbol{d}}\|_H^2 = \hat{f}_{\boldsymbol{x},H}(\hat{\boldsymbol{d}}) \le 0$. Also, from the line search condition (6.5),

$$-\gamma \le \frac{f(\boldsymbol{x}) - f(\boldsymbol{x}_+)}{\alpha\sigma} \le \frac{f(\boldsymbol{x}) - f(\boldsymbol{x}_+)}{\bar{\alpha}\sigma}, \tag{6.25}$$

where the last inequality is from Lemma 5. Therefore,

$$\|\boldsymbol{x}_+ - \boldsymbol{x}\| \le \sqrt{\frac{2}{m\sigma\bar{\alpha}}}\sqrt{f(\boldsymbol{x}) - f(\boldsymbol{x}_+)}. \tag{6.26}$$

Finally, we bound $\|\boldsymbol{x} - \bar{\boldsymbol{x}}\|$ by

$$\begin{aligned}
\|\boldsymbol{x} - \bar{\boldsymbol{x}}\| &\le \kappa\|\boldsymbol{d}_I(\boldsymbol{x})\| \quad \text{(Global error bound)} \\
&\le (1+M)\kappa\|\boldsymbol{d}_H^*\| \quad \text{(Lemma 7)} \\
&\le \frac{(1+M)\kappa}{\sqrt{m}}\|\boldsymbol{d}_H^*(\boldsymbol{x})\|_H \quad (\text{since } H \succeq mI) \\
&\le \frac{(1+M)\kappa\sqrt{2}}{\sqrt{m\sigma\bar{\alpha}\eta}}\sqrt{E\big[f(\boldsymbol{x}) - f(\boldsymbol{x}_+)\big]}. \quad \text{(by Lemma 8).}
\end{aligned} \tag{6.27}$$

Combining (6.27), (6.26), and (6.24) we get

$$① \le \frac{2L_g}{m\sigma\bar{\alpha}}\big(1 + \kappa^2\frac{1+M}{\sqrt{\eta}}\big)E\big[f(\boldsymbol{x}) - f(\boldsymbol{x}_+)\big]. \tag{6.28}$$

By combining (6.28), (6.21), (6.22), (6.23), we have

$$E\big[f(\boldsymbol{x}_+) - f(\bar{\boldsymbol{x}})\big] \le CE\big[f(\boldsymbol{x}) - f(\boldsymbol{x}_+)\big],$$

where

$$C = \frac{2L_g}{m\sigma\bar{\alpha}}\big(1 + \kappa^2\frac{1+M}{\sqrt{\eta}}\big) + \frac{1}{\sigma\bar{\alpha}\eta} + \frac{\kappa^2 M(1+M)}{m\sigma\bar{\alpha}\eta}$$

Finally,

$$\begin{aligned} E[f(\boldsymbol{x}_+)] - f(\bar{\boldsymbol{x}}) &\leq C(f(\boldsymbol{x}) - E[f(\boldsymbol{x}_+)]) \\ &= C(f(\boldsymbol{x}) - f(\bar{\boldsymbol{x}}) + f(\bar{\boldsymbol{x}}) - E[f(\boldsymbol{x}_+)]) \\ &= C(f(\boldsymbol{x}) - f(\bar{\boldsymbol{x}})) - C(E[f(\boldsymbol{x}_+)] - f(\bar{\boldsymbol{x}})) \end{aligned}$$

Therefore,

$$E[f(\boldsymbol{x}_+)] - f(\bar{\boldsymbol{x}}) \leq \frac{C}{1+C}(f(\boldsymbol{x}) - f(\bar{\boldsymbol{x}})).$$

□

6.2.3 Global Linear Convergence for Functions with Constant Nullspace Strong Convexity (CNSC)

To prove the convergence rate, we first state the following important lemma for CNSC functions.

Lemma 9. *If $f(\boldsymbol{x})$ satisfies CNSC from the beginning (Definition 12) and H satisfies the condition in Assumption 14b, then for any $\boldsymbol{x} \in \{\boldsymbol{x} : f(\boldsymbol{x}) \leq f(\boldsymbol{x}^0)\}$,*

$$\|P_{\mathcal{T}}(\boldsymbol{x} - \bar{\boldsymbol{x}})\| \leq \bar{\kappa}\|P_{\mathcal{T}}(\boldsymbol{d}_H^*(\boldsymbol{x}))\|$$

where $\bar{\boldsymbol{x}}$ is any optimal solution and $\bar{\kappa} = \frac{M+L_g}{m}$.

Proof. By definition, $\boldsymbol{d}_H^*(\boldsymbol{x})$ is the solution of the following problem:

$$\boldsymbol{d}_H^*(\boldsymbol{x}) = \operatorname*{argmin}_{\boldsymbol{d}} \nabla g(\boldsymbol{x})^T \boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^T H \boldsymbol{d} + h(\boldsymbol{x} + \boldsymbol{d}),$$

therefore, it is also the solution of the following problem since they have the same optimality condition.

$$\boldsymbol{d}_H^*(\boldsymbol{x}) = \underset{\boldsymbol{d}}{\operatorname{argmin}}(\nabla g(\boldsymbol{x}) + H\boldsymbol{d}_H^*(\boldsymbol{x}))^T\boldsymbol{d} + h(\boldsymbol{x}+\boldsymbol{d}).$$

Therefore, for any optimal solution $\bar{\boldsymbol{x}}$,

$$(\nabla g(\boldsymbol{x})+H\boldsymbol{d}_H^*(\boldsymbol{x}))^T\boldsymbol{d}_H^*(\boldsymbol{x})+h(\boldsymbol{x}+\boldsymbol{d}_H^*(\boldsymbol{x})) \leq (\nabla g(\boldsymbol{x})+H\boldsymbol{d}_H^*(\boldsymbol{x}))^T(\bar{\boldsymbol{x}}-\boldsymbol{x})+h(\bar{\boldsymbol{x}}). \tag{6.29}$$

Also, since $\bar{\boldsymbol{x}}$ is an optimal solution of $f(\cdot)$, $0 \in \nabla g(\bar{\boldsymbol{x}}) + \partial h(\bar{\boldsymbol{x}})$, therefore $\bar{\boldsymbol{x}}$ is also the solution of the following problem:

$$\bar{\boldsymbol{x}} = \underset{\boldsymbol{x}}{\operatorname{argmin}}\, g(\bar{\boldsymbol{x}})^T\boldsymbol{x} + h(\boldsymbol{x}).$$

So we have

$$\nabla g(\bar{\boldsymbol{x}})^T\bar{\boldsymbol{x}} + h(\bar{\boldsymbol{x}}) \leq \nabla g(\bar{\boldsymbol{x}})^T(\boldsymbol{x}+\boldsymbol{d}_H^*(\boldsymbol{x})) + h(\boldsymbol{x}+\boldsymbol{d}_H^*(\boldsymbol{x})). \tag{6.30}$$

Adding (6.29) and (6.30) together we get

$$\begin{aligned}&(\nabla g(\boldsymbol{x}) + H\boldsymbol{d}_H^*(\boldsymbol{x}))^T\boldsymbol{d}_H^*(\boldsymbol{x}) + \nabla g(\bar{\boldsymbol{x}})^T\bar{\boldsymbol{x}}\\ \leq&(\nabla g(\boldsymbol{x}) + H\boldsymbol{d}_H^*(\boldsymbol{x}))^T(\bar{\boldsymbol{x}}-\boldsymbol{x}) + \nabla g(\bar{\boldsymbol{x}})^T(\boldsymbol{x}+\boldsymbol{d}_H^*(\boldsymbol{x}))\end{aligned}$$

By rearranging terms, we get

$$\begin{aligned}&(\boldsymbol{d}_H^*(\boldsymbol{x}))^T H(\boldsymbol{d}_H^*(\boldsymbol{x})) + (\nabla g(\boldsymbol{x}) - \nabla g(\bar{\boldsymbol{x}}))^T(\boldsymbol{x}-\bar{\boldsymbol{x}})\\ \leq&(\boldsymbol{d}_H^*(\boldsymbol{x}))^T H(\bar{\boldsymbol{x}}-\boldsymbol{x}) + (\nabla g(\bar{\boldsymbol{x}}) - \nabla g(\boldsymbol{x}))^T\boldsymbol{d}_H^*(\boldsymbol{x})\end{aligned} \tag{6.31}$$

We first bound the left hand side.

$$\begin{aligned}\text{the left hand side} &\geq (\nabla g(\boldsymbol{x}) - \nabla g(\bar{\boldsymbol{x}}))^T(\boldsymbol{x} - \bar{\boldsymbol{x}}) \\ &= (\boldsymbol{x} - \bar{\boldsymbol{x}})^T \nabla^2 g(\phi)(\boldsymbol{x} - \bar{\boldsymbol{x}}) \quad \text{(by Mean Value Theorem)} \\ &\geq m\|P_{\mathcal{T}}(\boldsymbol{x} - \bar{\boldsymbol{x}})\|^2 \quad \text{(by Definition 12)} \end{aligned} \tag{6.32}$$

where $\psi = t\boldsymbol{x} + (1-t)\bar{\boldsymbol{x}}$ for some $t \in [0,1]$. Also,

$$\begin{aligned}\text{the right hand side} &\leq (\boldsymbol{d}_H^*(\boldsymbol{x}))^T H(\bar{\boldsymbol{x}} - \boldsymbol{x}) + (\bar{\boldsymbol{x}} - \boldsymbol{x})\nabla^2 g(\psi)\boldsymbol{d}_H^*(\boldsymbol{x}) \\ &\leq (M + L_g)\|P_{\mathcal{T}}(\boldsymbol{d}_H^*(\boldsymbol{x}))\|\|P_{\mathcal{T}}(\bar{\boldsymbol{x}} - \boldsymbol{x})\| \\ &\quad \text{(by Definition 12 and Assumption 14b).}\end{aligned} \tag{6.33}$$

Combining (6.31), (6.32), and (6.33), we get

$$\|P_{\mathcal{T}}(\boldsymbol{x} - \bar{\boldsymbol{x}})\| \leq \frac{M + L_g}{m}\|P_{\mathcal{T}}(\boldsymbol{d}_H^*(\boldsymbol{x}))\|.$$

□

Theorem 16. *If the objective function satisfies CNSC from the beginning (Definition 12), the H matrix used at each iteration satisfies Assumption 14b, and the subproblem solver has linear improvement in objective function (Assumption 8). Then the in-exact proximal Newton method has a global linear convergence rate:*

$$E[f(\boldsymbol{x}_+)] - f(\bar{\boldsymbol{x}}) \leq \frac{C}{1+C}\Big(f(\boldsymbol{x}) - f(\bar{\boldsymbol{x}})\Big),$$

where $\bar{\boldsymbol{x}}$ is an optimal solution of $f(\cdot)$ and

$$C = \frac{2L_g}{m\sigma\bar{\alpha}}\big(1 + \frac{\bar{\kappa}^2}{\sqrt{\eta}}\big) + \frac{1}{\sigma\bar{\alpha}\eta} + \frac{\bar{\kappa}^2 M}{m\sigma\bar{\alpha}\eta}.$$

Proof. Follow the poorf of Theorem 15, eq. (6.20) also satisfies for objective functions with CNSC assumption. We thus need to bound the four terms ①, ②, ③, ④.

For ②, ③, eq (6.21) and (6.22) still hold since we do not use any assumption on H and Global Error Bound.

For ④, we have

$$\begin{aligned}
④ &= \frac{1}{2}(\bar{\boldsymbol{x}} - \boldsymbol{x})^T H(\bar{\boldsymbol{x}} - \boldsymbol{x}) \\
&\leq \frac{M}{2}\|P_{\mathcal{T}}(\bar{\boldsymbol{x}} - \boldsymbol{x})\|^2 \\
&\leq \frac{\bar{\kappa}^2 M}{2}\|P_{\mathcal{T}}(\boldsymbol{d}_H^*(\boldsymbol{x}))\|^2 \quad \text{(by Lemma 9).} \\
&\leq \frac{\bar{\kappa}^2 M}{2m}\|\boldsymbol{d}_H^*(\boldsymbol{x})\|_H^2 \quad \text{(by Definition 12).} \\
&\leq \frac{\bar{\kappa}^2 M}{m\sigma\bar{\alpha}\eta}\big(f(\boldsymbol{x}) - E[f(\boldsymbol{x}_+)]\big) \quad \text{(by Lemma 8).}
\end{aligned} \tag{6.34}$$

For ①, we have

$$\begin{aligned}
① &= (\nabla g(\psi) - \nabla g(\boldsymbol{x}))^T(\boldsymbol{x}_+ - \bar{\boldsymbol{x}}) \\
&= (\psi - \boldsymbol{x})^T \nabla^2 g(\bar{\psi})(\boldsymbol{x}_+ - \bar{\boldsymbol{x}}),
\end{aligned}$$

where $\bar{\psi} = t\psi + (1-t)\boldsymbol{x}$ for some $t \in [0,1]$ by Mean Value Theorem. Therefore,

$$\begin{aligned}
① &\leq L_g\|P_{\mathcal{T}}(\psi - \boldsymbol{x})\|\|P_{\mathcal{T}}(\boldsymbol{x}_+ - \bar{\boldsymbol{x}})\| \quad \text{(by CNSC in Definition 12)} \\
&\leq L_g\|P_{\mathcal{T}}(\boldsymbol{x}_+ - \boldsymbol{x})\|\|P_{\mathcal{T}}(\boldsymbol{x}_+ - \bar{\boldsymbol{x}})\| \quad \text{(by definition of } \psi) \\
&\leq L_g\|P_{\mathcal{T}}(\boldsymbol{x}_+ - \boldsymbol{x})\|(\|P_{\mathcal{T}}(\boldsymbol{x}_+ - \boldsymbol{x})\| + \|P_{\mathcal{T}}(\boldsymbol{x} - \bar{\boldsymbol{x}})\|) \quad \text{(triangular inequality)} \\
&\leq L_g\|P_{\mathcal{T}}(\boldsymbol{x}_+ - \boldsymbol{x})\|^2 + L_g\|P_{\mathcal{T}}(\boldsymbol{x}_+ - \boldsymbol{x})\|\|P_{\mathcal{T}}(\boldsymbol{x} - \bar{\boldsymbol{x}})\|.
\end{aligned} \tag{6.35}$$

We further bound each term. First,

$$\|P_{\mathcal{T}}(\boldsymbol{x}_+ - \boldsymbol{x})\| = \alpha\|P_{\mathcal{T}}(\hat{\boldsymbol{d}})| \leq \frac{1}{\sqrt{m}}\|\hat{\boldsymbol{d}}\|_H \leq \sqrt{\frac{-2\gamma}{m}}.$$

And thus, by (6.25),

$$\|P_{\mathcal{T}}(\boldsymbol{x}_+ - \boldsymbol{x})\| \leq \sqrt{\frac{2}{m\sigma\bar{\alpha}}}\sqrt{f(\boldsymbol{x}) - f(\boldsymbol{x}_+)}. \tag{6.36}$$

Also follow the derivation in (6.27),

$$\|P_{\mathcal{T}}(\boldsymbol{x} - \bar{\boldsymbol{x}})\| \leq \bar{\kappa}\|P_{\mathcal{T}}(\boldsymbol{d}_H^*)\| \leq \sqrt{\bar{\kappa}}\sqrt{m}\|\boldsymbol{d}_H^*\|_H \leq \frac{\bar{\kappa}\sqrt{2}}{\sqrt{m\sigma\bar{\alpha}\eta}}\sqrt{f(\boldsymbol{x}) - E[f(\boldsymbol{x}_+)]}. \tag{6.37}$$

Combining (6.35), (6.36), (6.37), we get

$$① \leq \frac{2L_g}{m\sigma\bar{\alpha}}(1 + \frac{\bar{\kappa}^2}{\sqrt{\eta}})\big(f(\boldsymbol{x}) - E[f(\boldsymbol{x}_+)]\big). \tag{6.38}$$

Combining (6.38), (6.21), (6.22) and (6.34), we get

$$E[f(\boldsymbol{x}_+)] - f(\bar{\boldsymbol{x}}) \leq C(f(\boldsymbol{x}) - E[f(\boldsymbol{x}_+)]),$$

where

$$C = \frac{2L_g}{m\sigma\bar{\alpha}}\big(1 + \frac{\bar{\kappa}^2}{\sqrt{\eta}}\big) + \frac{1}{\sigma\bar{\alpha}\eta} + \frac{\bar{\kappa}^2 M}{m\sigma\bar{\alpha}\eta}$$

Following the last part of the proof of Theorem 15, we can prove this theorem.

□

6.3 Local Super-linear Convergence Rate for In-exact Proximal Gradient and Newton Methods

To show the asymptotic convergence rate, we focus on the functions that satisfy the CNSC assumption (Definition 12), and set the $H_t = \nabla^2 g(\boldsymbol{x}_t)$

at each iteration. Therefore, we will simplify the notation of $\hat{\boldsymbol{d}}_H, \boldsymbol{d}_H^*$ by $\hat{\boldsymbol{d}}, \boldsymbol{d}^*$ respectively. We need the following assumption for proving the super linear asymptotic convergence rate:

Assumption 17. *$\nabla^2 g(\cdot)$ is L_2-Lipchitz continuous:*

$$\|\nabla^2 g(\boldsymbol{x}) - \nabla^2 g(\boldsymbol{y})\|_2 \leq L_2 \|\boldsymbol{x} - \boldsymbol{y}\|_2.$$

6.3.1 Asymptotic Convergence Rate with Objective Function Reduction Subproblem Solvers

In this section we show the asymptotic convergence rate when the subproblem solver satisfies Assumption 8, which means the inner solver improves the objective function of the quadratic subproblem by a certain rate. To prove the convergence rate, we first bound the reduction of objective function $f(\cdot)$ by the following lemmas:

Lemma 10. *Any approximate direction $\hat{\boldsymbol{d}}_H$ uses an "objective function reduction subproblem solver" (Assumption 8) has the following property:*

$$E[\gamma] = E[\nabla g(\boldsymbol{x})^T \hat{\boldsymbol{d}} + h(\boldsymbol{x} + \hat{\boldsymbol{d}}) - h(\boldsymbol{x})] \leq -\frac{1}{\eta} E[\|\hat{\boldsymbol{d}}\|_H^2].$$

Proof. By the definition of objective function linear reduction in Assumption 8:

$$\begin{aligned}
&E\big[\nabla g(\boldsymbol{x})^T\hat{\boldsymbol{d}} + \frac{1}{2}\hat{\boldsymbol{d}}^T H\hat{\boldsymbol{d}} + h(\boldsymbol{x}+\hat{\boldsymbol{d}}) - h(\boldsymbol{x})\big] \\
\leq &\eta\big(\nabla g(\boldsymbol{x})^T(t\hat{\boldsymbol{d}}) + \frac{1}{2}(t\hat{\boldsymbol{d}})^T H(t\hat{\boldsymbol{d}}) + h(\boldsymbol{x}+t\hat{\boldsymbol{d}}) - h(\boldsymbol{x})\big) \\
= &\eta t\nabla g(\boldsymbol{x})^T\hat{\boldsymbol{d}} + \frac{1}{2}\eta t^2\hat{\boldsymbol{d}}^T H\hat{\boldsymbol{d}} + \eta t\big(h(\boldsymbol{x}+\hat{\boldsymbol{d}}) - h(\boldsymbol{x})\big) \quad \text{(by Lemma 4).}
\end{aligned}$$

Therefore,

$$\begin{aligned}
(1-\eta t)E\big[\nabla g(\boldsymbol{x})^T\hat{\boldsymbol{d}}\big] + \frac{1}{2}(1-\eta t^2)E\big[\hat{\boldsymbol{d}}^T H\hat{\boldsymbol{d}}\big] + (1-\eta t)E\big[h(\boldsymbol{x}+\hat{\boldsymbol{d}}) - h(\boldsymbol{x})\big] \leq 0 \\
E\big[\nabla g(\boldsymbol{x})^T\hat{\boldsymbol{d}}\big] + \frac{1}{2}\frac{1-\eta t^2}{1-\eta t}E\big[\hat{\boldsymbol{d}}^T H\hat{\boldsymbol{d}}\big] + E\big[h(\boldsymbol{x}+\hat{\boldsymbol{d}}) - h(\boldsymbol{x})\big] \leq 0
\end{aligned} \tag{6.39}$$

Taking $t \uparrow \frac{1}{\eta}$, using L'Hospital's rule, we have

$$\lim_{t\uparrow\frac{1}{\eta}} \frac{1-\eta t^2}{1-\eta t} = \lim_{t\uparrow\frac{1}{\eta}} \frac{-2\eta t}{-\eta} = \frac{2}{\eta} \tag{6.40}$$

Combining (6.39) and (6.40) we get

$$E[\gamma] = E\big[\nabla g(\boldsymbol{x})^T\hat{\boldsymbol{d}} + h(\boldsymbol{x}+\hat{\boldsymbol{d}}) - h(\boldsymbol{x})\big] \leq -\frac{1}{\eta}E\big[\hat{\boldsymbol{d}}^T H\hat{\boldsymbol{d}}\big].$$

□

We then proof the step size will always be 1 when $\boldsymbol{x}$ is close enough to the optimal solution. The following proof is similar to Proposition 5 in [32].

Lemma 11. *Assume the objective function satisfies CNSC (Definition 12) from the beginning, the subproblem solver reduces objective function (Assumption 8), and* $\sigma < 0.5$ *in the line search condition* (6.5)*. Then the step size* $\alpha = 1$ *satisfies the sufficient decrease condition* (6.5) *if* $\boldsymbol{x}$ *is close enough to an optimal solution* $\bar{\boldsymbol{x}}$*.*

Proof. We define $\tilde{g}(t) = g(\boldsymbol{x}+t\hat{\boldsymbol{d}})$ and by chain rule we have $\tilde{g}''(t) = \hat{\boldsymbol{d}}^T \nabla g(\boldsymbol{x}+t\hat{\boldsymbol{d}})\hat{\boldsymbol{d}}$. Thus we have

$$\begin{aligned}
|\tilde{g}''(t) - \tilde{g}''(0)| &= |\hat{\boldsymbol{d}}^T(\nabla^2 g(\boldsymbol{x}+t\hat{\boldsymbol{d}}) - \nabla^2 g(\boldsymbol{x}))\hat{\boldsymbol{d}}| \\
&\leq \|\nabla^2 g(\boldsymbol{x}+t\hat{\boldsymbol{d}}) - \nabla^2 g(\boldsymbol{x})\| \|P_{\mathcal{T}}(\hat{\boldsymbol{d}})\|^2 \\
&\leq L_2 \|P_{\mathcal{T}}(\hat{\boldsymbol{d}})\|^3.
\end{aligned}$$

Therefore,

$$\begin{aligned}
\tilde{g}''(t) &\leq \tilde{g}''(0) + tL_2\|P_{\mathcal{T}}(\hat{\boldsymbol{d}})\|^3 \\
&= \hat{\boldsymbol{d}}^T \nabla^2 g(\boldsymbol{x})\hat{\boldsymbol{d}} + tL_2\|P_{\mathcal{T}}(\hat{\boldsymbol{d}})\|^3.
\end{aligned}$$

Integrate both side twice we get

$$\tilde{g}(t) \leq \tilde{g}(\boldsymbol{0}) + t\hat{\boldsymbol{d}}^T \nabla g(\boldsymbol{x}) + \frac{1}{2}t^2 \hat{\boldsymbol{d}}^T \nabla^2 g(\boldsymbol{x})\hat{\boldsymbol{d}} + \frac{1}{6}t^3 L_2 \|P_{\mathcal{T}}(\hat{\boldsymbol{d}})\|^3.$$

Taking $t = 1$ we get

$$g(\boldsymbol{x}+\hat{\boldsymbol{d}}) \leq g(\boldsymbol{x}) + \hat{\boldsymbol{d}}^T \nabla g(\boldsymbol{x}) + \frac{1}{2}\hat{\boldsymbol{d}}^T \nabla^2 g(\boldsymbol{x})\hat{\boldsymbol{d}} + \frac{1}{6}L_2\|\hat{\boldsymbol{d}}\|^3.$$

As a result,

$$\begin{aligned}
f(\boldsymbol{x}+\hat{\boldsymbol{d}}) &= g(\boldsymbol{x}+\hat{\boldsymbol{d}}) + h(\boldsymbol{x}+\hat{\boldsymbol{d}}) \\
&= f(\boldsymbol{x}) + \nabla g(\boldsymbol{x})^T\hat{\boldsymbol{d}} + h(\boldsymbol{x}+\hat{\boldsymbol{d}}) - h(\boldsymbol{x}) + \frac{1}{2}\hat{\boldsymbol{d}}^T \nabla^2 g(\boldsymbol{x})\hat{\boldsymbol{d}} + \frac{1}{6}L_2\|P_{\mathcal{T}}(\hat{\boldsymbol{d}})\|^3 \\
&\leq f(\boldsymbol{x}) + \gamma + \frac{1}{2}\|\hat{\boldsymbol{d}}\|_H^2 + \frac{1}{6m}L_2\|\hat{\boldsymbol{d}}\|_H^2\|P_{\mathcal{T}}(\hat{\boldsymbol{d}})\| \quad \text{(by Assumption 12)} \\
&\leq f(\boldsymbol{x}) + \gamma - \frac{1}{2\eta}\gamma + \frac{L_2}{6m\eta}\|P_{\mathcal{T}}(\hat{\boldsymbol{d}})\|\gamma \quad \text{(by Lemma 10)} \\
&= f(\boldsymbol{x}) + \big(1 - \frac{1}{2\eta} - \frac{L_2\|P_{\mathcal{T}}(\hat{\boldsymbol{d}})\|}{6m\eta}\big)\gamma.
\end{aligned}$$

Since $\eta < 1$, $\|\hat{\boldsymbol{d}}\| \to 0$, and $\sigma < 0.5$, for $\boldsymbol{x}$ is close enough to $\boldsymbol{x}^*$ we have

$$f(\boldsymbol{x} + \hat{\boldsymbol{d}}) - f(\boldsymbol{x}) \leq \sigma\gamma.$$

□

Finally, we prove the following theorem showing the asymptotic convergence speed of the in-exact proximal Newton method.

Theorem 18. *If the objective function satisfies CNSC from the beginning (Definition 12), the subproblem solver satisfies Assumption 8, and $H_t = \nabla^2 g(\boldsymbol{x}_t)$ at each iteration of Algorithm 8, then we have*

$$E[f(\boldsymbol{x}_+)] - f(\boldsymbol{x}^*) \leq (1-\eta)\big(f(\boldsymbol{x}) - f(\bar{\boldsymbol{x}})\big) + C\big(f(\boldsymbol{x}) - f(\bar{\boldsymbol{x}})\big)^{\frac{3}{2}},$$

when $\boldsymbol{x}$ is close enough to an optimal solution $\bar{\boldsymbol{x}}$, where

$$C = \frac{L_2}{2}\Big(\frac{1}{m\eta}\Big)^{\frac{3}{2}}\Big(1 + \eta\Big(\frac{\bar{\kappa}^2}{\sigma}\Big)^{\frac{3}{2}}\Big).$$

Proof. We bound the improvement made at each step. By mean value theorem, there exists a $\psi \in line(\boldsymbol{x}, \boldsymbol{x}_+)$ such that

$$\begin{aligned}
f(\boldsymbol{x}_+) - f(\boldsymbol{x}) &= g(\boldsymbol{x}_+) - g(\boldsymbol{x}) + h(\boldsymbol{x}_+) - h(\boldsymbol{x}) \\
&= \nabla g(\boldsymbol{x})^T(\boldsymbol{x}_+ - \boldsymbol{x}) + \frac{1}{2}(\boldsymbol{x}_+ - \boldsymbol{x})^T\nabla^2 g(\psi)(\boldsymbol{x}_+ - \boldsymbol{x}) + h(\boldsymbol{x}_+) - h(\boldsymbol{x}) \\
&= \nabla g(\boldsymbol{x})^T(\boldsymbol{x}_+ - \boldsymbol{x}) + \frac{1}{2}(\boldsymbol{x}_+ - \boldsymbol{x})^T\nabla^2 g(\boldsymbol{x})(\boldsymbol{x}_+ - \boldsymbol{x}) + h(\boldsymbol{x}_+) - h(\boldsymbol{x}) \\
&\quad + \frac{1}{2}(\boldsymbol{x}_+ - \boldsymbol{x})^T\big(\nabla^2 g(\psi) - \nabla^2 g(\boldsymbol{x})\big)(\boldsymbol{x}_+ - \boldsymbol{x}).
\end{aligned}$$

By Lemma 11, $\alpha = 1$, so $\boldsymbol{x}_+ = \boldsymbol{x} + \hat{\boldsymbol{d}}$. Since $\nabla^2 g(\boldsymbol{x})$ is L_2-Lipchitz continuous (Assumption 17),

$$f(\boldsymbol{x}_+) - f(\boldsymbol{x}) \leq \hat{f}_{\boldsymbol{x},H}(\hat{\boldsymbol{d}}) + \frac{1}{2} L_2 \|P_{\mathcal{T}}(\boldsymbol{x}_+ - \boldsymbol{x})\|^3 \tag{6.41}$$

By mean value theory, there exists a $\bar{\psi} \in line(\boldsymbol{x}, \bar{\boldsymbol{x}})$ such that

$$g(\bar{\boldsymbol{x}}) = g(\boldsymbol{x}) + \nabla g(\boldsymbol{x})^T (\bar{\boldsymbol{x}} - \boldsymbol{x}) + \frac{1}{2} (\bar{\boldsymbol{x}} - \boldsymbol{x})^T \nabla^2 g(\bar{\psi}) (\bar{\boldsymbol{x}} - \boldsymbol{x}).$$

We define $\bar{H} = \nabla^2 g(\bar{\psi})$, $\bar{\boldsymbol{d}} = \bar{\boldsymbol{x}} - \boldsymbol{x}$, $\boldsymbol{d}^* = \operatorname{argmin}_{\boldsymbol{d}} \hat{f}_{\boldsymbol{x},H}(\boldsymbol{d})$, and $H = \nabla^2 g(\boldsymbol{x})$. Then we have

$$\begin{aligned}
\hat{f}_{\boldsymbol{x},H}(\boldsymbol{d}^*) &\leq \hat{f}_{\boldsymbol{x},H}(\bar{\boldsymbol{d}}) \\
&= \nabla g(\boldsymbol{x})^T \bar{\boldsymbol{d}} + \frac{1}{2} \bar{\boldsymbol{d}}^T H \bar{\boldsymbol{d}} + h(\boldsymbol{x} + \bar{\boldsymbol{d}}) - h(\boldsymbol{x}) \\
&= \nabla g(\boldsymbol{x})^T \bar{\boldsymbol{d}} + \frac{1}{2} \bar{\boldsymbol{d}}^T \bar{H} \bar{\boldsymbol{d}} + h(\boldsymbol{x} + \bar{\boldsymbol{d}}) - h(\boldsymbol{x}) + \frac{1}{2} \bar{\boldsymbol{d}}^T (H - \bar{H}) \bar{\boldsymbol{d}} \\
&\leq \hat{f}_{\boldsymbol{x},\bar{H}}(\bar{\boldsymbol{d}}) + \frac{1}{2} L_2 \|P_{\mathcal{T}}(\bar{\boldsymbol{d}})\|^3 \\
&= f(\bar{\boldsymbol{x}}) - f(\boldsymbol{x}) + \frac{1}{2} L_2 \|P_{\mathcal{T}}(\bar{\boldsymbol{d}})\|^3.
\end{aligned} \tag{6.42}$$

From the definition of $\hat{\boldsymbol{d}}$, $\hat{f}_{\boldsymbol{x},H}(\hat{\boldsymbol{d}}) \leq \eta \hat{f}_{\boldsymbol{x},H}(\boldsymbol{d}^*)$. Combining this with (6.41) and (6.42) we get

$$f(\boldsymbol{x}_+) - f(\boldsymbol{x}) \leq \eta \big(f(\bar{\boldsymbol{x}}) - f(\boldsymbol{x}) \big) + \frac{\eta L_2}{2} \|P_{\mathcal{T}}(\boldsymbol{x} - \bar{\boldsymbol{x}})\|^3 + \frac{L_2}{2} \|P_{\mathcal{T}}(\boldsymbol{x}_+ - \boldsymbol{x})\|^3. \tag{6.43}$$

Now we further bound each term in (6.43). First,

$$\begin{aligned}
\|P_{\mathcal{T}}(\boldsymbol{x} - \bar{\boldsymbol{x}})\|^2 &\leq \bar{\kappa}^2 \|P_{\mathcal{T}}(\boldsymbol{d}_H^*(\boldsymbol{x}))\|^2 \quad \text{(by Lemma 9)} \\
&\leq \frac{\bar{\kappa}^2}{m} \|P_{\mathcal{T}}(\boldsymbol{d}_H^*(\boldsymbol{x}))\|_H^2 \quad \text{(by CNSC assumption)} \\
&\leq \frac{\bar{\kappa}^2}{m \sigma \eta} (f(\boldsymbol{x}) - f(\boldsymbol{x}_+)) \quad \text{(by Lemma 8 and Lemma 11).}
\end{aligned} \tag{6.44}$$

Also,

$$\|P_{\mathcal{T}}(\boldsymbol{x}_+ - \boldsymbol{x})\| = \|P_{\mathcal{T}}(\hat{\boldsymbol{d}})\| \leq \frac{1}{\sqrt{m}}\|\hat{\boldsymbol{d}}\|_H \leq \sqrt{\frac{-\gamma}{m\eta}}, \tag{6.45}$$

where the last inequality is from Lemma 10. Combining (6.43), (6.44), (6.45) we have

$$f(\boldsymbol{x}_+)-f(\boldsymbol{x}) \leq \eta\big(f(\bar{\boldsymbol{x}})-f(\boldsymbol{x})\big)+\frac{\eta L_2}{2}(\frac{\bar{\kappa}^2}{m\sigma\eta})^{\frac{3}{2}}\big(f(\boldsymbol{x})-f(\boldsymbol{x}_+)\big)^{\frac{3}{2}}+\frac{L_2}{2}(\frac{1}{m\eta})^{\frac{3}{2}}\big(F(\boldsymbol{x})-F(\boldsymbol{x}_+)\big)^{\frac{3}{2}}.$$

Adding $f(\boldsymbol{x}) - f(\bar{\boldsymbol{x}})$ to both sides and using the fact that

$$f(\boldsymbol{x}) - f(\boldsymbol{x}_+) \leq f(\boldsymbol{x}) - f(\bar{\boldsymbol{x}}),$$

we get

$$f(\boldsymbol{x}_+) - f(\bar{\boldsymbol{x}}) \leq (1-\eta)\big(f(\boldsymbol{x}) - f(\bar{\boldsymbol{x}})\big) + C\big(f(\boldsymbol{x}) - f(\bar{\boldsymbol{x}})\big)^{\frac{3}{2}},$$

where

$$C = \frac{L_2}{2}(\frac{1}{m\eta})^{\frac{3}{2}}\big(1+\eta(\frac{\bar{\kappa}^2}{\sigma})^{\frac{3}{2}}\big).$$

□

Based on Theorem 18, we know the convergence rate of the algorithm with different setting of η_t. For example, we can show the super-linear convergence when $\lim_{t\to\infty}\eta_t = 1$.

Corollary 19. *Under the same assumption of Theorem 18 and if* $\lim_{t\to\infty}\eta_t = 1$*, then* $f(\boldsymbol{x})$ *converges super-linearly to* $f(\bar{\boldsymbol{x}})$.

Proof. To show the super linear convergence rate, we compute

$$\begin{aligned}
&\lim_{t\to\infty} \frac{f(\boldsymbol{x}^{t+1}) - f(\bar{\boldsymbol{x}})}{f(\boldsymbol{x}^{t}) - f(\bar{\boldsymbol{x}})} \\
&= \lim_{t\to\infty} \frac{(1-\eta)(f(\boldsymbol{x}^{t}) - f(\bar{\boldsymbol{x}})) + C(f(\boldsymbol{x}^{t}) - f(\bar{\boldsymbol{x}}))^{\frac{3}{2}}}{f(\boldsymbol{x}^{t}) - f(\bar{\boldsymbol{x}})} \\
&= \lim_{t\to\infty} (1-\eta) + C(f(\boldsymbol{x}^{t}) - f(\bar{\boldsymbol{x}}))^{\frac{1}{2}}.
\end{aligned}$$

Therefore, when $(1-\eta) \to 0$,

$$\lim_{t\to\infty} \frac{f(\boldsymbol{x}^{t+1}) - f(\bar{\boldsymbol{x}})}{f(\boldsymbol{x}^{t}) - f(\bar{\boldsymbol{x}})} = 0.$$

□

6.3.2 Asymptotic Convergence Rate and Global Convergence Rate with Gradient Reduction Subproblem Solvers

Next we discuss the convergence rate if each subproblem solver satisfies Assumption 9, which means they reduce the magnitude of composite gradient of the quadratic subproblem by a certain ratio (η_t). In this case, instead of proving the convergence in terms of objective function value $\{f(\boldsymbol{x}_t)\}$, we can show the convergence of $\{\boldsymbol{x}_t\}$, which is more standard in the literature for proving the quadratic convergence of second order methods. Note that the following analysis is similar to [50], but they only consider the case when $g(\cdot)$ is strongly convex, which is not true for many machine learning problems. Instead, we consider a more general CNSC objective functions which may not be strongly convex.

Note that [87] showed that if $\boldsymbol{x}$ is decomposed into $\boldsymbol{z} + \boldsymbol{y}$, where $\boldsymbol{z} = P_{\mathcal{T}}(\boldsymbol{x})$ and $\boldsymbol{y} = P_{\mathcal{T}^{\perp}}(\boldsymbol{x})$, then the proximal Newton operation can be rewritten

as

$$\boldsymbol{z}^{t+1} = \underset{\boldsymbol{z}\in\mathcal{T}}{\operatorname{argmin}}\, h(\boldsymbol{z}+\hat{\boldsymbol{y}}(\boldsymbol{z})) + \nabla g(\boldsymbol{x}^t)^T(\boldsymbol{z}-\boldsymbol{z}^t) + \frac{1}{2}(\boldsymbol{z}-\boldsymbol{z}^t)^T\nabla^2 g(\boldsymbol{x}^t)(\boldsymbol{z}-\boldsymbol{z}^t),$$

where

$$\hat{\boldsymbol{y}}(\boldsymbol{z}) = \underset{\boldsymbol{y}\in\mathcal{T}^\perp}{\operatorname{argmin}}\, h(\boldsymbol{z}+\boldsymbol{y}),$$

so the optimality is actually determined by $P_{\mathcal{T}}(\boldsymbol{x}^t)$ at the t-th iteration. Moreover, this also implies the convergence in the objective function by the following lemma proved in [87] for CNSC functions:

Lemma 12. *If $g(\cdot)$ is L_g-Lipschitz continuous and $h(\cdot)$ is L_h-Lipschitz continuous, then for proximal Newton method*

$$f(\boldsymbol{x}_+) - f(\boldsymbol{x}) \leq \max(L_g, L_h)\|P_{\mathcal{T}}(\boldsymbol{x}-\bar{\boldsymbol{x}})\|.$$

Therefore, for CNSC functions, instead of showing the convergence rate of $\|\boldsymbol{x}-\bar{\boldsymbol{x}}\|$, we show the convergence of $\|P_{\mathcal{T}}(\boldsymbol{x}-\bar{\boldsymbol{x}})\|$. We first show the following lemmas. Note that the notations $G_f(\cdot)$ was defined in Section 6.1.1.

Lemma 13. *Given a composite function $f(\boldsymbol{u}) = g(\boldsymbol{u}) + h(\boldsymbol{u})$ that satisfies CNSC from the beginning (Definition 12), and $\hat{f}_{\boldsymbol{x}}$ defined in* (6.7)*, then*

$$\|G_f(\boldsymbol{u}) - G_{\hat{f}_{\boldsymbol{x}}}(\boldsymbol{u})\| \leq \frac{L_2}{2}\|P_{\mathcal{T}}(\boldsymbol{x}-\boldsymbol{u})\|^2.$$

Note that $\hat{f}_{\boldsymbol{x}} := \hat{f}_{\boldsymbol{x},\nabla^2 g(\boldsymbol{x})}$.

Proof.

$$\begin{aligned}\|G_f(\boldsymbol{u}) - G_{\hat{f}_{\boldsymbol{x}}}(\boldsymbol{u})\| &\leq \|\operatorname{prox}_h(\boldsymbol{u} - \nabla g(\boldsymbol{u})) - \operatorname{prox}_h(\boldsymbol{u} - \nabla \hat{g}(\boldsymbol{u}))\| \\ &\leq \|\nabla g(\boldsymbol{u}) - \nabla \hat{g}(\boldsymbol{u})\| \quad \text{(by non-expensive of prox}(\cdot)) \\ &= \|\nabla g(\boldsymbol{u}) - \nabla g(\boldsymbol{x}) - \nabla^2 g(\boldsymbol{x})(\boldsymbol{u} - \boldsymbol{x})\| \quad \text{(by definition of } \hat{g}(\boldsymbol{x})) \\ &\leq \frac{L_2}{2}\|P_{\mathcal{T}}(\boldsymbol{x} - \boldsymbol{u})\|^2\end{aligned}$$

□

Lemma 14. *If $f(\cdot)$ satisfies CNSC from the beginning (Definition 12), then for any $\alpha \leq \frac{1}{L_g}$ we have*

$$\begin{aligned}(\boldsymbol{u} - \boldsymbol{v})^T(G_{f,\alpha}(\boldsymbol{u}) - G_{f,\alpha}(\boldsymbol{v})) &\geq \frac{m}{2}\|P_{\mathcal{T}}(\boldsymbol{u} - \boldsymbol{v})\|^2, \\ \|G_{f,\frac{1}{L}}(\boldsymbol{u}) - G_{f,\frac{1}{L}}(\boldsymbol{v})\| &\geq \frac{m}{2}\|P_{\mathcal{T}}(\boldsymbol{u} - \boldsymbol{v})\|.\end{aligned}$$

Proof. Using the Moreau decomposition,

$$\boldsymbol{x} = \operatorname{prox}_h(\boldsymbol{x}) + \operatorname{prox}_{h^*}(\boldsymbol{x}) \quad \forall \boldsymbol{x},$$

where h^* is the conjugate of h. Therefore,

$$G_{f,\alpha}(\boldsymbol{u}) - G_{f,\alpha}(\boldsymbol{v}) = \nabla g(\boldsymbol{u}) - \nabla g(\boldsymbol{v}) + \frac{1}{\alpha}(\operatorname{prox}_{(\alpha h)^*}(\boldsymbol{u} - \alpha\nabla g(\boldsymbol{u})) - \operatorname{prox}_{(\alpha h)^*}(\boldsymbol{v} - \nabla g(\boldsymbol{v}))).$$

Let

$$\begin{aligned}\boldsymbol{w} &= \operatorname{prox}_{(\alpha h)^*}(\boldsymbol{u} - \alpha\nabla g(\boldsymbol{u})) - \operatorname{prox}_{(\alpha h)^*}(\boldsymbol{v} - \alpha\nabla g(\boldsymbol{v})) \text{ and} \\ \boldsymbol{d} &= (\boldsymbol{u} - \boldsymbol{v}) - \alpha(\nabla g(\boldsymbol{u}) - \nabla g(\boldsymbol{v})) \\ W &= \frac{\boldsymbol{w}\boldsymbol{w}^T}{\boldsymbol{w}^T\boldsymbol{d}}.\end{aligned}$$

Then we have

$$G_{f,\alpha}(\boldsymbol{u}) - G_{f,\alpha}(\boldsymbol{v}) = \nabla g(\boldsymbol{u}) - \nabla g(\boldsymbol{v}) + \frac{1}{\alpha} W \boldsymbol{d}.$$

Multiply both sides by $(\boldsymbol{u} - \boldsymbol{v})$ we get

$$\begin{aligned}
&(\boldsymbol{u} - \boldsymbol{v})^T (G_{f,\alpha}(\boldsymbol{u}) - G_{f,\alpha}(\boldsymbol{v})) \\
=&(\boldsymbol{u} - \boldsymbol{v})^T (\nabla g(\boldsymbol{u}) - \nabla g(\boldsymbol{v})) + \frac{1}{\alpha} (\boldsymbol{u} - \boldsymbol{v})^T W \bigg((\boldsymbol{u} - \boldsymbol{v}) - \alpha (\nabla g(\boldsymbol{u}) - \nabla g(\boldsymbol{v})) \bigg).
\end{aligned}$$

Let $H(\theta) = \nabla^2 g(\boldsymbol{x} + \theta(\boldsymbol{y} - \boldsymbol{x}))$ for $\theta \in [0, 1]$, we have

$$\begin{aligned}
&(\boldsymbol{u} - \boldsymbol{v})^T (G_{f,\alpha}(\boldsymbol{u}) - G_{f,\alpha}(\boldsymbol{v})) \\
=& \int_0^1 (\boldsymbol{u} - \boldsymbol{v})^T H(\theta) (\boldsymbol{u} - \boldsymbol{v}) d\theta + \int_0^1 (\boldsymbol{u} - \boldsymbol{v})^T \frac{W}{\alpha} (\boldsymbol{u} - \boldsymbol{v}) d\theta - \int_0^1 (\boldsymbol{u} - \boldsymbol{v}) W H(\theta) (\boldsymbol{u} - \boldsymbol{v}) d\theta \\
=& \int_0^1 (\boldsymbol{u} - \boldsymbol{v})^T (H(\theta) - \frac{1}{2}(WH(\theta) + H(\theta)W) + \frac{1}{\alpha} W)(\boldsymbol{u} - \boldsymbol{v}) d\theta.
\end{aligned}$$

Since

$$\alpha H(\theta)^2 + \frac{1}{\alpha} W^2 \geq WH(\theta) + H(\theta)W,$$

we have

$$(\boldsymbol{u} - \boldsymbol{v})^T (G_{f,\alpha}(\boldsymbol{u}) - G_{f,\alpha}(\boldsymbol{v})) = \int_0^1 (\boldsymbol{u} - \boldsymbol{v})^T \bigg(H(\theta) - \frac{\alpha}{2} H(\theta)^2 + \frac{1}{\alpha} (W - \frac{1}{2} W^2) \bigg) (\boldsymbol{u} - \boldsymbol{v}) d\theta.$$

Since $\text{prox}(\cdot)$ is non-expensive, $\|\boldsymbol{w}\|^2 \leq \boldsymbol{w}^T \boldsymbol{d}$, so

$$W = \frac{\boldsymbol{w}\boldsymbol{w}^T}{\boldsymbol{w}^T \boldsymbol{d}} = \frac{\|\boldsymbol{w}\|^2}{\boldsymbol{w}^T \boldsymbol{d}} \frac{\boldsymbol{w}\boldsymbol{w}^T}{\|\boldsymbol{w}\|^2} \preceq I$$

So

$$\begin{aligned}
(\boldsymbol{u} - \boldsymbol{v})^T (G_{f,\alpha}(\boldsymbol{u}) - G_{f,\alpha}(\boldsymbol{v})) &\geq \int_0^1 (\boldsymbol{u} - \boldsymbol{v})^T (H(\theta) - \frac{\alpha}{2} H(\theta)^2)(\boldsymbol{u} - \boldsymbol{v}) d\theta \\
&\geq \frac{m}{2} \|P_{\mathcal{T}}(\boldsymbol{u} - \boldsymbol{v})\|^2 \quad (\text{since } \alpha \leq \frac{1}{L} \text{ and } H \preceq L_g I).
\end{aligned}$$

□

We show the following theorem characterize the convergence rate of in-exact proximal Newton method.

Theorem 20. *If the objective function satisfies CNSC from the beginning (Definition 12), the subproblem solver satisfies (Assumption 9), and at each iteration $H_t = \nabla^2 g(\boldsymbol{x}_t)$, then*

$$\|P_{\mathcal{T}}(\boldsymbol{x}_+ - \bar{\boldsymbol{x}})\| \leq \frac{L_2}{mL}\|P_{\mathcal{T}}(\boldsymbol{x} - \bar{\boldsymbol{x}})\|^2 + \frac{4(1-\eta)}{m^2}\|P_{\mathcal{T}}(\boldsymbol{x} - \bar{\boldsymbol{x}})\|,$$

where $\bar{\boldsymbol{x}}$ is an optimal solution of f.

Proof.

$$\begin{aligned}
\|P_{\mathcal{T}}(\boldsymbol{x}_+ - \bar{\boldsymbol{x}})\| &\leq \frac{2}{m}\|G_{\hat{f},\frac{1}{L}}(\boldsymbol{x}_+) - G_{\hat{f},\frac{1}{L}}(\bar{\boldsymbol{x}})\| \quad \text{(Lemma 14)} \\
&\leq \frac{2}{m}\left(\|G_{\hat{f},\frac{1}{L}}(\boldsymbol{x}_+)\| + \|G_{\hat{f},\frac{1}{L}}(\bar{\boldsymbol{x}})\|\right) \text{(triangular inequality)} \\
&\leq \frac{2}{m}\left((1-\eta)\|G_{f,\frac{1}{L}}(\boldsymbol{x})\| + \|G_{\hat{f},\frac{1}{L}}(\bar{\boldsymbol{x}})\|\right) \quad \text{(by Assumption 9)} \\
&\leq \frac{2}{m}\left(\frac{2}{m}(1-\eta)\|P_{\mathcal{T}}(\boldsymbol{x} - \bar{\boldsymbol{x}})\| + \|G_{\hat{f},\frac{1}{L}}(\bar{\boldsymbol{x}})\|\right) \quad \text{(Lemma 14)} \\
&\leq \frac{4(1-\eta)}{m^2}\|P_{\mathcal{T}}(\boldsymbol{x} - \bar{\boldsymbol{x}})\| + \frac{2}{m}\big(\|G_{f,\frac{1}{L}}(\bar{\boldsymbol{x}})\| + \frac{L_2}{2L}\|P_{\mathcal{T}}(\boldsymbol{x} - \bar{\boldsymbol{x}})\|^2\big) \\
&\qquad\qquad \text{(by Lemma 13)} \\
&\leq \frac{4(1-\eta)}{m^2}\|P_{\mathcal{T}}(\boldsymbol{x} - \bar{\boldsymbol{x}})\| + \frac{L_2}{mL}\|P_{\mathcal{T}}(\boldsymbol{x} - \bar{\boldsymbol{x}})\|^2.
\end{aligned}$$

□

Based on Theorem 20, we know the convergence rate of the algorithm with different setting of η_t. For example, we can show the super-linear convergence when $\lim_{t\to\infty} \eta_t = 1$.

Corollary 21. *Under the same assumption of Theorem 20 and if* $\lim_{t\to\infty}\eta_t = 1$*, then* $\|P_{\mathcal{T}}(\boldsymbol{x}-\bar{\boldsymbol{x}})\|$ *converges super-linearly to* 0*.*

Furthermore, if η is controlled properly, we can have a quadratic convergence rate:

Corollary 22. *Under the same assumption of Theorem 20 and if* $1-\eta_t \leq \theta G_f(\boldsymbol{x}_t)$ *for each* t *with a constant* θ*, then* $\|P_{\mathcal{T}}(\boldsymbol{x}-\bar{\boldsymbol{x}})\|$ *converges quadratically to* 0*.*

Proof. Since $g(\cdot)$ is L_g-Lipschitz continuous, we have

$$G_f(\boldsymbol{x}_t) \leq L_g\|P_{\mathcal{T}}(\boldsymbol{x}_t-\bar{\boldsymbol{x}})\|.$$

Combining with Theorem 20 we have

$$\|P_{\mathcal{T}}(\boldsymbol{x}_+-\bar{\boldsymbol{x}})\| \leq \frac{L_2}{mL_g}\|P_{\mathcal{T}}(\boldsymbol{x}-\bar{\boldsymbol{x}})\|^2 + \frac{4\theta L_g}{m^2}\|P_{\mathcal{T}}(\boldsymbol{x}-\bar{\boldsymbol{x}})\|^2,$$

which implies $\|P_{\mathcal{T}}(\boldsymbol{x}-\bar{\boldsymbol{x}})\|$ converges to 0 quadratically. □

6.3.3 Subproblem solvers that can be used

Any optimization algorithm can be used to solve the subproblem $\hat{f}_{\boldsymbol{x},H}(\cdot)$ approximately in Algorithm 8. Based on Theorem 15, we can easily derive the following Corollary:

Corollary 23. *If a subproblem solver* $\mathbb{S}$ *used in Algorithm 4 has a global linear convergence rate in terms of objective function, then Algorithm 4 also has a global linear convergence rate if each subproblem is solved by* $\mathbb{S}$ *with* ≥ 1

iterations. Moreover, if each subproblem is solved by $\mathbb{S}$ *with a fixed number of iterations, then the total number of inner iterations needed for obtaining an* ϵ*-accurate solution is* $O(\log(\frac{1}{\epsilon}))$.

We list some commonly used subproblem solvers with global linear convergence rate:

1. Randomized Coordinate Descent: [62] proved the convergence of randomized coordinate descent under the global error bound assumption.

2. Cyclic Coordinate Descent: [80] proved the convergence of randomized coordinate descent under the global error bound assumption.

3. Greedy Coordinate Descent: [80] proved the convergence of greedy coordinate descent (with Gauss-Southwell rule) under the global error bound assumption.

4. Proximal Gradient Descent: [87] showed a global linear convergence rate for proximal gradient descent when $h(\cdot)$ satisfies certain condition (for example, $h(\boldsymbol{x})$ is lasso or group-lasso regularization).

6.4 Applications

6.4.1 Convergence Rate of In-exact Proximal Gradient Descent and Newton Method

When $H_t = \nabla^2 g(\boldsymbol{x}_t)$ at each iteration, Algorithm 8 is the proximal Newton method used in many applications, including sparse inverse covariance estimation [32] and ℓ_1-regularized logistic regression [89].

In sparse inverse covariance estimation, the objective function for the ℓ_1-regularized maximum likelihood estimator can be written as

$$\underset{X \succ 0}{\operatorname{argmin}} - \log\det(X) + \operatorname{trace}(SX) + \lambda \sum_{i,j} |X_{ij}|.$$

It has been shown in [32] that the smooth part is strongly convex in the level set ($\{X : f(X) \leq f(X^0)\}$), so it satisfies both global error bound (Definition 10) and CNSC (Definition 12) from the beginning. The proximal Newton method has been proposed for solving this sparse inverse covariance estimation problem [32], and the algorithm is implemented in the QUIC package. In the implementation, each subproblem is solved with an increasing number of coordinate descent operations. This has not been justified in the previous theoretical analysis [32, 50], but we here we provide the analysis of the convergence rate.

Our framework also provides the convergence guarantee when applying inexact proximal Newton methods to regularized loss minimization problems:

$$\underset{\boldsymbol{x}}{\operatorname{argmin}} \sum_{i=1}^{n} \ell_i(\boldsymbol{x}^T \boldsymbol{a}_i) + h(\boldsymbol{x}), \tag{6.46}$$

where $\{\boldsymbol{a}_i\}_{i=1}^n$ are training samples, ℓ_i measures the training error defined on each sample, and $h(\boldsymbol{x})$ is the regularization to minimize the model complexity. Since $g(\boldsymbol{x}) = \sum_{i=1}^n \ell_i(\boldsymbol{x}^T \boldsymbol{a}_i)$, the Hessian matrix can be written as

$$\nabla^2 g(\boldsymbol{x}) = ADA^T,$$

where $A = [\boldsymbol{a}_1,\ \boldsymbol{a}_2,\ \ldots, \boldsymbol{a}_n]$, and D is a diagonal matrix with

$$D_{ii} = \ell_i''(u)\ |_{u=\boldsymbol{x}^T \boldsymbol{a}_i}\ .$$

Therefore, if ℓ_i is strongly convex and nonzero in the level set, the function $g(\boldsymbol{x})$ will satisfy CNSC condition from the beginning (Definition 12) where $\mathcal{T} = \text{col}(A)$.

Therefore, we show the linear convergence when applying in-exact proximal Newton method to any regularized loss minimization problems. For example, the GLMNET [89] algorithm for solving the ℓ_1-regularized logistic regression problem falls into our framework.

6.4.2 Global linear convergence for in-exact Proximal Gradient Descent or Newton Method with Active Subspace Selection

As discussed in Section 5.2, an active subspace selection technique can be used to speed up proximal Newton methods when the regularization term $h(\cdot)$ is a decomposable norm. At each iteration, the space is partitioned into the active subspace $\mathcal{S}_{free}$ and the complementary subspace $\mathcal{S}_{fixed}$ by eq (4.25):

$$\mathcal{S}_{fiexed} := [\mathcal{T}(\boldsymbol{x})]^{\perp} \cap [\mathcal{T}(\text{prox}(\boldsymbol{x} - \nabla g(\boldsymbol{x})))]^{\perp} \text{ and } \mathcal{S}_{free} = \mathcal{S}_{fixed}^{\perp},$$

where $\mathcal{T}(\boldsymbol{x})$ is the support of $\boldsymbol{x}$. and we solve the quadratic subproblem only within the active subspace:

$$\tilde{\boldsymbol{d}} = \underset{\boldsymbol{d} \in \mathcal{S}_{free}}{\text{argmin}} \left\{ \nabla g(\boldsymbol{x})^T \boldsymbol{d} + \frac{1}{2} \boldsymbol{d}^T \nabla^2 g(\boldsymbol{x}) \boldsymbol{d} + h(\boldsymbol{x} + \boldsymbol{d}) - h(\boldsymbol{x}) \right\} := \hat{f}_{x, \nabla^2 g(\boldsymbol{x})}(\boldsymbol{d}). \tag{6.47}$$

The following theory shows that $\tilde{\boldsymbol{d}}$ is equivalent to the solution of $\hat{f}_{\boldsymbol{x},H}(\boldsymbol{d})$ with a specific H matrix:

Theorem 24. *If $\boldsymbol{d}^*$ is the optimal solution of* (6.47)*, then it is also the optimal*

solution of $\hat{f}_{\boldsymbol{x},H}(\boldsymbol{d})$ *with* $H = R(R^T\nabla^2 g(\boldsymbol{x})R)R^T + R^{\perp}(R^{\perp})^T$, *where* $R = [\boldsymbol{r}_1, \ldots, \boldsymbol{r}_q]$ *are the orthogonal basis for* $\mathbb{S}_{free}$.

Combining this theorem with Theorem 15 and 16, we can conclude that (in-exact) proximal Newton method with active subspace selection has a global linear convergence rate.

6.4.3 Global linear convergence for Parallel Algorithms

The framework discussed in Algorithm 8 can also be applied to distributed proximal-Newton typed algorithms described in Section 5.3. This algorithm has been applied to solve the dual problem of linear SVM/logistic regression in [86, 35, 47], and in Section 5.3 we generalized it to any composite function, and showed it is effective for solving kernel machines.

We revisit the proposed distributed proximal-Newton framework again. In this framework, the variables are first partitioned into k disjoint index sets

$$S_1 \cup S_2 \cup \cdots \cup S_k = \{1, \ldots, d\} \quad \text{and} \quad S_p \cap S_q = \phi \ \forall p \neq q,$$

and we use $\pi(i)$ to denote the cluster indicator that i belongs to. Each worker r is associated with a subset of variables $\boldsymbol{x}_{S_r} := \{x_i \mid i \in S_r\}$.

At each synchronization point, a worker obtain the latest global vector $\boldsymbol{x}_t$, and then runs coordinate descent (or other algorithms) to update the local variables $\boldsymbol{d}_{S_r}$ while keep other variables fixed. These algorithms are equivalent to Algorithm 8 with H_t set by

$$H = \begin{cases} \nabla^2 g(\boldsymbol{x})_{ij} & \text{if } \pi(i) = \pi(j) \\ 0 & \text{otherwise.} \end{cases}$$

Moreover, since the subproblems are usually solved by coordinate descent, using our analysis we can prove the global linear convergence rate of these parallel solvers. For example [86, 35, 47] focus on the dual problem of the ℓ_2-regularized empirical risk minimization, and the parallel proximal Newton method discussed in Section 5.3 also has a global linear convergence rate.

6.4.4 Summary of the Contribution

In this chapter we provide a comprehensive theoretical study of in-exact proximal gradient and Newton methods. The work is under preparation for submitting to an optimization journal.

Bibliography

[1] Michael Berry, Murray Browne, Amy Langville, Paul Pauca, and Robert Plemmon. Algorithms and applications for approximate nonnegative matrix factorization. *Computational Statistics and Data Analysis*, 2006.

[2] Dimitri P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, MA 02178-9998, second edition, 1999.

[3] A. Bordes, S. Ertekin, J. Weston, and L. Bottou. Fast kernel classifiers with online and active learning. *JMLR*, 6:1579–1619, 2005.

[4] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, 7th printing edition, 2009.

[5] S. P. Boyd, N. Parikh, E. Peleato, and J. Eckstein. Distributed optimization and statistical learning via alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[6] E. Candes and B. Recht. Simple bounds for recovering low-complexity models. *Mathemetical Programming*, 2012.

[7] E. Chan, M. Heimlich, A. Purkayastha, and R. van de Geijn. Collective communication: Theory, practice, and experience. *Concurrency and Computation: Practice and Experience*, 19:1749–1783, 2007.

[8] V. Chandrasekaran, P. A. Parrilo, and A. S. Willsky. Latent variable graphical model selection via convex optimization. *The Annals of Statistics*, 2012.

[9] V. Chandrasekaran, S. Sanghavi, P. A. Parrilo, and A. S. Willsky. Rank-sparsity incoherence for matrix decomposition. *Siam J. Optim*, 21(2):572–596, 2011.

[10] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[11] Edward Chang, Kaihua Zhu, Hao Wang, Hongjie Bai, Jian Li, Zhihuan Qiu, and Hang Cui. Parallelizing support vector machines on distributed computers. In *NIPS*, pages 257–264. 2008.

[12] A. Cichocki and A-H. Phan. Fast local algorithms for large scale nonnegative matrix and tensor factorizations. *IEICE Transaction on Fundamentals*, E92-A(3):708–721, 2009.

[13] M. Collins, S. Dasgupta, and R. E. Schapire. A generalization of principal component analysis to the exponential family. In *NIPS*, 2012.

[14] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.

[15] I. S. Dhillon, Y. Guan, and B. Kulis. Weighted graph cuts without eigenvectors: A multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 29:11:1944–1957, 2007.

[16] N. Djuric, L. Lan, S. Vucetic, and Z. Wang. Budgetedsvm: A toolbox for scalable svm approximations. *JMLR*, 14:3813–3817, 2013.

[17] P. Drineas and M. W. Mahoney. On the Nyström method for approximating a Gram matrix for improved kernel-based learning. *JMLR*, 6:2153–2175, 2005.

[18] S. Fine and K. Scheinberg. Efficient SVM training using low-rank kernel representations. *JMLR*, 2:243–264, 2001.

[19] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3:95–110, 1956.

[20] J. Friedman, T. Hastie, H. Höfling, and R. Tibshirani. Pathwise coordinate optimization. *Annals of Applied Statistics*, 1(2):302–332, 2007.

[21] Jerome H. Friedman, Trevor Hastie, and Robert Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010.

[22] Radha Ghitta, Rong Jin, Timothy C. Havens, and Anil K. Jain. Approximate kernel k-means: Solution to large scale kernel clustering. In *KDD*, 2011.

[23] Edward F. Gonzales and Yin Zhang. Accelerating the Lee-Seung algorithm for non-negative matrix factorization. Technical report, Department of Computational and Applied Mathematics, Rice University, 2005.

[24] H. P. Graf, E. Cosatto, L. Bottou, I. Dundanovic, and V. Vapnik. Parallel support vector machines: The cascade SVM. In *NIPS*, 2005.

[25] Patrik O. Hoyer. Non-negative sparse coding. In *Proceedings of IEEE Workshop on Neural Networks for Signal Processing*, pages 557–565, 2002.

[26] C.-J. Hsieh and I. S. Dhillon. Fast coordinate descent methods with variable selection for non-negative matrix factorization. In *KDD*, 2011.

[27] C.-J. Hsieh, I. S. Dhillon, P. Ravikumar, and A. Banerjee. A divide-and-conquer method for sparse inverse covariance estimation. In *NIPS*, 2012.

[28] C.-J. Hsieh, I. S. Dhillon, P. Ravikumar, S. Becker, and P. A. Olsen. Quic & Dirty: A quadratic approximation approach for dirty statistical models. In *NIPS*, 2014.

[29] C.-J. Hsieh and P. A. Olsen. Nuclear norm minimization via active subspace selection. In *ICML*, 2014.

[30] C.-J. Hsieh, S. Si, and I. S. Dhillon. A divide-and-conquer solver for kernel support vector machines. In *ICML*, 2014.

[31] C.-J. Hsieh, M. A. Sustik, I. S. Dhillon, and P. Ravikumar. Sparse inverse covariance matrix estimation using quadratic approximation. In *NIPS*, 2011.

[32] C.-J. Hsieh, M. A. Sustik, I. S. Dhillon, and P. Ravikumar. QUIC: Quadratic approximation for sparse inverse covariance estimation. *JMLR*, 2014.

[33] C.-J. Hsieh, M. A. Sustik, I. S. Dhillon, P. Ravikumar, and R. A. Poldrack. Big & Quic: Sparse inverse covariance estimation for a million variables. In *NIPS*, 2013.

[34] D. Hsu, S. M. Kakade, and T. Zhang. Robust matrix decomposition with sparse corruptions. *IEEE Trans. Inform. Theory*, 57:7221–7234, 2011.

[35] Martin Jaggi, Virginia Smith, Martin Takáč, Jonathan Terhorst, Thomas Hofmann, and Michael I Jordan. Communication-efficient distributed dual coordinate ascent. In *NIPS*. 2014.

[36] A. Jalali, P. Ravikumar, S. Sanghavi, and C. Ruan. A dirty model for multi-task learning. In *NIPS*, 2010.

[37] T. Joachims. Making large-scale SVM learning practical. In *Advances in Kernel Methods – Support Vector Learning*, pages 169–184, 1998.

[38] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1999.

[39] S. S. Keerthi, O. Chapelle, and D. DeCoste. Building support vector machines with reduced classifier complexity. *JMLR*, 7:1493–1515, 2006.

[40] S. Sathiya Keerthi, Kaibo Duan, Shirish Shevade, and Aun Neow Poo. A fast dual algorithm for kernel logistic regression. *Machine Learning*, 61:151–165, 2005.

[41] S. Sathiya Keerthi, Shirish Krishnaj Shevade, Chiranjib Bhattacharyya, and Karuturi Radha Krishna Murthy. Improvements to Platt's SMO algorithm for SVM classifier design. *Neural Computation*, 13:637–649, 2001.

[42] D. Kim, S. Sra, and I. S. Dhillon. Fast Newton-type methods for the least squares nonnegative matrix appoximation problem. *Proceedings of the Sixth SIAM International Conference on Data Mining*, pages 343–354, 2007.

[43] Jingu Kim and Haesun Park. Non-negative matrix factorization based on alternating non-negativity constrained least squares and active set method. *SIAM Journal on Matrix Analysis and Applications*, 30(2):713–730, 2008.

[44] Jingu Kim and Haesun Park. Toward faster nonnegative matrix factorization: A new algorithm and comparisons. *Proceedings of the IEEE International Conference on Data Mining*, pages 353–362, 2008.

[45] S. Kumar, M. Mohri, and A. Talwalkar. Ensemble nyström method. In *NIPS*, 2009.

[46] Q. V. Le, T. Sarlos, and A. J. Smola. Fastfood – approximating kernel expansions in loglinear time. In *ICML*, 2013.

[47] C.-P. Lee and D. Roth. Distributed box-constrained quadratic optimization for dual linear SVM. In *ICML*, 2015.

[48] Daniel D. Lee and H. Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999.

[49] Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. In Todd K. Leen, Thomas G. Dietterich, and Volker Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 556–562. MIT Press, 2001.

[50] J. D. Lee, Y. Sun, and M. A. Saunders. Proximal Newton-type methods for convex optimization. In *NIPS*, 2012.

[51] L. Li and K.-C. Toh. An inexact interior point method for L1-reguarlized sparse covariance selection. *Mathematical Programming Computation*, 2:291–315, 2010.

[52] Chih-Jen Lin. Projected gradient methods for non-negative matrix factorization. *Neural Computation*, 19:2756–2779, 2007.

[53] Chih-Jen Lin, Ruby C. Weng, and S. Sathiya Keerthi. Trust region Newton method for large-scale logistic regression. In *Proceedings of the 24th International Conference on Machine Learning (ICML)*, 2007.

[54] Zhi-Quan Luo and Paul Tseng. On the convergence of coordinate descent method for convex differentiable minimization. *Journal of Optimization Theory and Applications*, 72(1):7–35, 1992.

[55] Zhi-Quan Luo and Paul Tseng. Error bounds and convergence analysis of feasible descent methods: a general approach. *Annals of Operations Research*, 46:157–178, 1993.

[56] S. Ma, L. Xue, and H. Zou. Alternating direction methods for latent variable Gaussian graphical model selection. *Neural Computation*, 25(8):2172–2198, 2013.

[57] D. Mahajan, S. S. Keerthi, and S. Sundararajan. A distributed algorithm for training nonlinear kernel machines. 2014.

[58] Rahul Mazumder and Trevor Hastie. Exact covariance thresholding into connected components for large-scale graphical lasso. *Journal of Machine Learning Research*, 13:723–736, 2012.

[59] A. K. Menon. Large-scale support vector machines: algorithms and theory. Technical report, University of California, San Diego, 2009.

[60] John Moody and Christian J. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, pages 281–294, 1989.

[61] M. Nandan, P. R. Khargonekar, and S. S. Talathi. Fast svm training using approximate extreme points. *JMLR*, 15:59–98, 2014.

[62] I. Necoara and D. Clipici. Parallel random coordinate descent method for composite minimization. 2013.

[63] S. N. Negahban, P. Ravikumar, M. J. Wainwright, and B. Yu. A unified framework for high-dimensional analysis of m-estimators with decomposable regularizers. *Statistical Science*, 27(4):538–557, 2012.

[64] Y. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

[65] P. A. Olsen, S. J. Rennie, and V. Goel. Efficient automatic differentiation of matrix functions. *Recent Advances in Algorithmic Differentiation*, 2012.

[66] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 130–136, 1997.

[67] Pentti Paatero and Unto Tapper. Positive matrix factorization: A non-negative factor model with optimal utilization of error. *Environmetrics*, 5:111–126, 1994.

[68] F. Pérez-Cruz, A. R. Figueiras-Vidal, and A. Artés-Rodríguez. Double chunking for solving SVMs for very large datasets. In *Proceedings of Learning*, 2004.

[69] Jon Piper, Paul Pauca, Robert Plemmons, and Maile Giffin. Object characterization from spectral data using nonnegative factorization and information theory. In *Proceedings of AMOS Technical Conference*, 2004.

[70] J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods - Support Vector Learning*, 1998.

[71] A. Rahimi and B. Recht. Random features for large-scale kernel machines. *NIPS*, 2008.

[72] P. Ravikumar, M. J. Wainwright, G. Raskutti, and B. Yu. High-dimensional covariance estimation by minimizing ℓ_1-penalized log-determinant divergence. *ejs*, 5:935–980, 2011.

[73] K. Scheinberg, S. Ma, and D. Goldfarb. Sparse inverse covariance selection via alternating linearization methods. *NIPS*, 2010.

[74] Katya Scheinberg and Xiaocheng Tang. Practical inexact proximal quasi-newton method with global complexity analysis. 2014.

[75] O. Shamir, N. Srebro, and T. Zhang. Communication efficient distributed optimization using an approximate newton-type method. In *ICML*, 2014.

[76] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[77] S. Si, C. J. Hsieh, and I. S. Dhillon. Memory efficient kernel approximation. In *International Conference on Machine Learning (ICML)*, June 2014.

[78] A. Tewari, P. Ravikumar, and I. Dhillon. Greedy algorithms for structurally constrained high dimensional problems. In *NIPS*, 2011.

[79] P. Tseng and S. Yun. A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming*, 117:387–423, 2007.

[80] P. Tseng and S. Yun. A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming*, 117:387–423, 2009.

[81] M. van Breukelen, R. P. W. Duin, D. M. J. Tax, and J. E. den Hartog. Handwritten digit recognition by combined classifiers. *Kybernetika*, 34(4):381–386, 1998.

[82] Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[83] C. Wang, D. Sun, and K.-C. Toh. Solving log-determinant optimization problems by a Newton-CG primal proximal point algorithm. *SIAM J. Optimization*, 20:2994–3013, 2010.

[84] Po-Wei Wang and Chih-Jen Lin. Iteration complexity of feasible descent methods for convex optimization. *Journal of Machine Learning Research*, 15:1523–1548, 2014.

[85] Christopher K. I. Williams and Matthias Seeger. Using the Nyström method to speed up kernel machines. In *NIPS*, 2001.

[86] T. Yang. Trading computation for communication: Distributed stochastic dual coordinate ascent. In *NIPS*, 2013.

[87] I. Yen, C.-J. Hsieh, P. Ravikumar, and I. S. Dhillon. Constant nullspace strong convexity and fast convergence of proximal methods under high-dimensional settings. In *Neural Information Processing Systems Conference (NIPS)*, December 2014.

[88] Hsiang-Fu Yu, Fang-Lan Huang, and Chih-Jen Lin. Dual coordinate descent methods for logistic regression and maximum entropy models. *Machine Learning*, 85(1-2):41–75, October 2011.

[89] G.-X. Yuan, C.-H. Ho, and C.-J. Lin. An improved GLMNET for l1-regularized logistic regression. In *ACM SIGKDD*, 2011.

[90] R. Zdunek and A. Cichocki. Non-negative matrix factorization with quasi-newton optimization. *Eighth International Conference on Artificial Intelligence and Soft Computing, ICAISC*, pages 870–879, 2006.

[91] K. Zhang, L. Lan, Z. Wang, and F. Moerchen. Scaling up kernel SVM on limited resources: A low-rank linearization approach. In *AISTATS*, 2012.

[92] K. Zhang, I. W. Tsang, and J. T. Kwok. Improved Nyström low rank approximation and error analysis. In *ICML*, 2008.

[93] T. Zhang. Sequential greedy approximation for certain convex optimization problems. *IEEE Transactions on Information Theory*, 49(3):682–691, 2003.

[94] Y. Zhang, J. Duchi, and M. Wainwright. Communication-efficient algorithms for statistical optimization. *JMLR*, 14:3321–3363, 2013.

[95] Zeyuan A. Zhu, Weizhu Chen, Gang Wang, Chenguang Zhu, and Zheng Chen. P-packSVM: Parallel primal gradient descent kernel SVM. In *Proceedings of the IEEE International Conference on Data Mining*, 2009.

[96] M. Zinkevich, M. Weimer, A. Smola, and L. Li. Parallelized stochastic gradient descent. In *NIPS*, 2010.

Vita

Cho-Jui Hsieh is a Ph.D. student at University of Texas at Austin. His research focus is developing new algorithms and optimization techniques for large-scale machine learning problems. Cho-Jui obtained his B.S. degree in 2007 and M.S. degree in 2009 from National Taiwan University (advisor: Chih-Jen Lin). Currently he is a member of Center for Big Data Analytics led by Inderjit Dhillon. He is the recipient of the IBM Ph.D. fellowship in 2013-2015, the best research paper award in KDD 2010, and the best paper award in ICDM 2012.

Email address: cjhsieh@cs.utexas.edu

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.