Big Data Optimization in Machine Learning

by

Xiaocheng Tang

A Dissertation
Presented to the Graduate Committee
of Lehigh University
in Candidacy for the Degree of
Doctor of Philosophy
in
Industrial Engineering

Lehigh University
January, 2016

Approval of the Doctoral Committee

Big Data Optimization in Machine Learning

Xiaocheng Tang

Approved and recommended for acceptance as a dissertation in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Date

Dissertation Advisor

Committee Members:

Dr. Katya Scheinberg, Committee Chair

Dr. Frank E. Curtis, Dissertation Advisor

Dr. Nader Motee, Dissertation Advisor

Dr. Jorge Nocedal, External Member

Dedication

To my family.

Acknowledgments

First and foremost, I would like to express my greatest appreciation to my advisor and dissertation committee chair Dr. Katya Scheinberg who has provided me guidance and support ever since the beginning of my Ph.D. study. She is an amazing advisor and has always inspired me to reach for a higher level.

I would like to thank my committee members, Dr. Frank E. Curtis, Dr. Nader Motee and Dr. Jorge Nocedal. Their insightful comments and constructive advices have been invaluable. Without their guidance and persistent help, this dissertation would not have been possible.

I also wish to thank Peiyan who has been hugely supportive throughout the several years it has taken to finish this work.

And finally I would like to thank all my friends, professors and faculties at Lehigh, and everyone else who has helped me complete this dissertation. Without their continued efforts and support, I would have not been able to bring my work to a successful completion.

Contents

Dedication **iv**

Acknowledgments **v**

List of Tables **viii**

List of Figures **ix**

Abstract **1**

1 Introduction **3**

1.1 Hybrid approach and derivative-free optimization in structured learning 4

1.2 Adaptive active-set and communication-efficient coordinate descent . 9

1.3 Complexity of inexact proximal Newton methods and randomization . 12

1.4 A brief outline of the thesis 15

2 Machine Learning Applications of Convex Optimization **17**

2.1 Introduction 17

2.2 Sparse Logistic Regression 18

2.2.1 Optimization Algorithms 20

2.3 HiClass - An efficient Hierarchical Classification system 25

2.3.1 The Hierarchical Classification Framework 27

2.3.2 A Novel Hybrid Approach 29

2.3.3 Experiments 32
2.4 Distributed Learning for Short-term Forecasting 39
2.4.1 Problem Setting 42
2.4.2 Description of the Algorithm 44

3 Hierarchical Multi-label Learning 56
3.1 Introduction 56
3.2 Hierarchical sparsity-inducing norms 60
3.3 Hierarchical Multi-label Model 62
3.4 Learning ϕ 64
3.4.1 Derivative-Free Optimization 64
3.4.2 Complexity of hierarchical operator 65
3.5 Other hierarchical methods 67
3.6 Numerical Experiments 71
3.7 Conclusion 74

4 Complexity of Inexact Proximal Newton Methods 75
4.1 Introduction 75
4.2 Basic algorithmic framework and theoretical analysis 76
4.3 Basic results, assumptions and preliminary analysis 78
4.3.1 Analysis via proximal gradient approach 81
4.4 Analysis of sub linear convergence 88
4.4.1 The exact case 89
4.4.2 The inexact case 91
4.4.3 Complexity in terms of subproblem solver iterations 98
4.5 Analysis of the inexact case under random subproblem accuracy . . . 100
4.5.1 Analysis of Subproblem Optimization via Randomized Coordinate Descent 103

5 Adaptive Active-set in Proximal Quasi-Newton Method for Large Scale Sparse Optimization 106
5.1 Introduction 106
5.2 The proposed algorithm 107
5.3 Mutable matrix type in low-rank hessian approximation 110
5.3.1 Efficient matrix mutating operations 111
5.4 Self-adaptive greedy active-set selection 115
5.5 Fast coordinate descent on low-rank structure 118
5.6 Inner loop termination criteria and global convergence 119
5.7 Numerical experiments 123
5.7.1 Active-set methods 123
5.7.2 Inner loop termination and convergence 128
5.7.3 Performance comparison with other algorithms 131

6 Conclusions 140

Bibliography 151

Biography 152

List of Tables

2.1 Per Iteration Complexity 25

3.1 Datasets statistics used in experiments. Dataset *itp* and *ads* have the same exact dimension, but are constructed differently – *itp* targets users' general interest and includes users' activities from other category-related events besides ads view or ads click. 67

3.2 The comparison of FLAT, HML, Leaf Reduce (LR) and Root Map (RM) on all three datasets, with squared and logistic loss function, and evaluated by precision, recall and f1 score. 67

5.1 Row and column operations on `LMatrix` type $M \in \mathbb{R}^{n\times m}$. Note that while one `memcpy` or `memmove` on $O(n)$ data has a complexity of $O(n)$, the constant in front of n can be much smaller than 1 (hence much faster than normal n flops), due to the use of cache optimizations and specialized processor instructions such as SIMD in those built-in library functions. 112

5.2 Data sets used in the experiments. 123

5.3 Compare the performance of different active-set strategies. 127

5.4 Data statistics in sparse logistic regression experiments. 136

List of Figures

1.1 Visualization of taxonomy among labels in dataset *ads*. 6

2.1 The hierarchy tree of the labels. 29
2.2 HiClass system flow chart 32
2.3 CCAT subtree 34
2.4 Performance comparison at each node between above mentioned algorithms. For HYBRID, we use GLMNET in the upper level and IPM at bottom. The number in the leftmost block of each table denotes the corresponding performance measure on the whole tree. For example, as shown by the rightmost upper table, 70% aggregate accuracy is achieved using the novel HYBRID method. 36
2.5 MCAT subtree 37
2.6 Performance comparison at each node between above mentioned algorithms. 38
2.7 Illustration of Load Series Periodicity. 41

3.1 The tree structure is illustrated with blue rectangles as nodes and straight lines as edges. Groups g are green translucent rectangles covering nodes that are contained in the group. 61

3.2 Testing and training performance under different metrics as learning objective on *rcv* dataset. Take (b) for example. HML equipped with jaccard similarity as the learning objective is applied on a training set of 3000 instances. Solutions after each iteration (the x-axis) of HML are recorded and later evaluated under all three metrics on the full *rcv* dataset (test set). The red curve denotes the objective values evaluated on the test set. In the same subplot along with the red curve are the objective values evaluated at different iterations on the training set. Other two subplots illustrate the same solutions evaluated by other two metrics on the same test set. 68

5.1 Illustrate the dynamics of the four index sets (5.4.1) as $x \to x^*$. Note how $|\mathcal{Z}_1(x)|$ and $|\mathcal{Z}_2(x)|$ start from the top and the bottom, respectively, and come together in the end, each containing (roughly) half the size of non-zero elements in x^*. Also note that $|\mathcal{Z}_3(x)|$ vanishes at optimality. 126
5.2 Evolution of free sets in STD and GRDY-AD. *Left:* dataset mnist ($p = 784, N = 1,000,000$). *Right:* dataset gisette ($p = 5000, N = 6000$). The line plots are the size of free set against iterations and the bars denote the total sum of free set sizes over all iterations 126
5.3 Study different inner loop termination criteria and its effect on algorithm performance. Time (top) and iterations (bottom) comparison for different $l(k)$: log, linear, constant 5 and constant 200. 128
5.4 Consider linear function $l(k) = ak + b$ and study the effect of a on running time and iterations for different inner algorithm, i.e., coordinate descent (a) and proximal gradient (b). 129
5.5 Convergence plots on SICS (the y-axes on log scale). 135
5.6 RCD step count of LHAC on different SICS data sets. 136
5.7 Convergence plots on SLR (the y-axes on log scale). 137
5.8 RCD step count of LHAC on different SLR data sets. 138

Abstract

Modern machine learning practices at the interface of big data, distributed environment and complex learning objectives post great challenges to designing scalable optimization algorithms with theoretical guarantees. This thesis, built on the recent advances in randomized algorithms, concerns development of such methods in practice and the analysis of their theoretical implications in the context of large-scale structured learning problems, such as regularized regression/classification, matrix completion, hierarchical multi-label learning, etc. The first contribution of this work is thus a hybrid hierarchical learning system that achieve efficiency in a data-intensive environment. The intelligent decoding scheme inside the system further enhances the learning capacity by enabling a rich taxonomy representation to be induced in the label space. Important factors affecting the system scalability are studied and further generalized. This leads to the next contribution of the work – a globally convergent inexact proximal quasi-Newton framework and the novel global convergence rate analysis. This work constitutes the first global convergence rate result for an algorithm that uses randomized coordinate descent to inexactly optimize subproblems at each iteration. The analysis quantifies precisely the complexity structure of proximal Newton-type algorithms, which makes it possible to optimize based on that structure to reduce complexity. The final contribution of the work is a practical algorithm which enjoys global convergence guarantee from the framework. The algorithm is memory- and communication-efficient and directly addresses the big data learning cases when both N (samples) and n (features) are large. We demonstrated that this general algorithm is very effective in practice and is competitive with state-of-the-art

specialized methods.

Chapter 1

Introduction

The basic problem of interest in this work is the unconstrained minimization given by:

$$\min_{x \in \mathbb{R}^n} F(x) \equiv f(x) + g(x) \tag{1.0.1}$$

where $f, g : \mathbb{R}^n \to \mathbb{R}$ are both convex functions such that f is twice differentiable, with a bounded Hessian, and g is Lipschitz continuous, often non-smooth and in many applications has the following separable structure

$$g(x) = \sum_{j=1}^{J} \lambda_j g_j(x_j) \tag{1.0.2}$$

where each g_j is a convex function; $\lambda > 0$ is a parameter; and $x_j \in \mathbb{R}^{n_j}$ denotes a partition of the component of x, e.g., the widely-used ℓ_1 regularization is obtained by choosing $g_j = |\cdot|, n_j = 1, \forall j$.

Also known as composite optimization, problems of the form (1.0.1) have been the focus of much research lately at the interface of optimization and machine learning. This form encompasses a variety of machine learning models, in which feature selection or structured learning is desired, e.g.,

- regularized logistic regression, where f computes the logistic loss on the given training set and the regularizer g can be group-ℓ_1/ℓ_2 depending on the problem.

- inverse covariance estimation, where $f(X) = -\log\det X + \mathrm{tr}(SX)$, resulting from maximum conditional log-likelihood estimation like logistic regression, with S as the sample covariance matrix; $g(X) = \sum_{i,j} \lambda_{ij}|X_{ij}|$ is essential here in order to successfully recover the zero entries in the final inverse covariance matrix.

- LASSO or group LASSO, where f is a least-square function and g is either $||\cdot||_1$ or group ℓ_1 regularization depending on whether group sparsity is required.

- support vector machine, where $f(w) = ||w||_2^2$ and $g(w)$ is the Vapnik's ϵ-insensitive loss such that in the form (1.0.2) J is the sample size, each partition $n_j = n$ includes the full vector w and each g_j is the loss $\max(0, |y_i - wx_i| - \epsilon)$ for the i-th example (x_i, y_i) in the training set.

- matrix completion, where in one formulation that estimates a low-rank matrix $X = LR^T$, f computes the least-square loss between the linear mapping $\mathcal{A}(LR^T)$ and a given vector b, while $g = ||\cdot||_F^2$ regularizes the low-rank factor L and R.

1.1 Hybrid approach and derivative-free optimization in structured learning

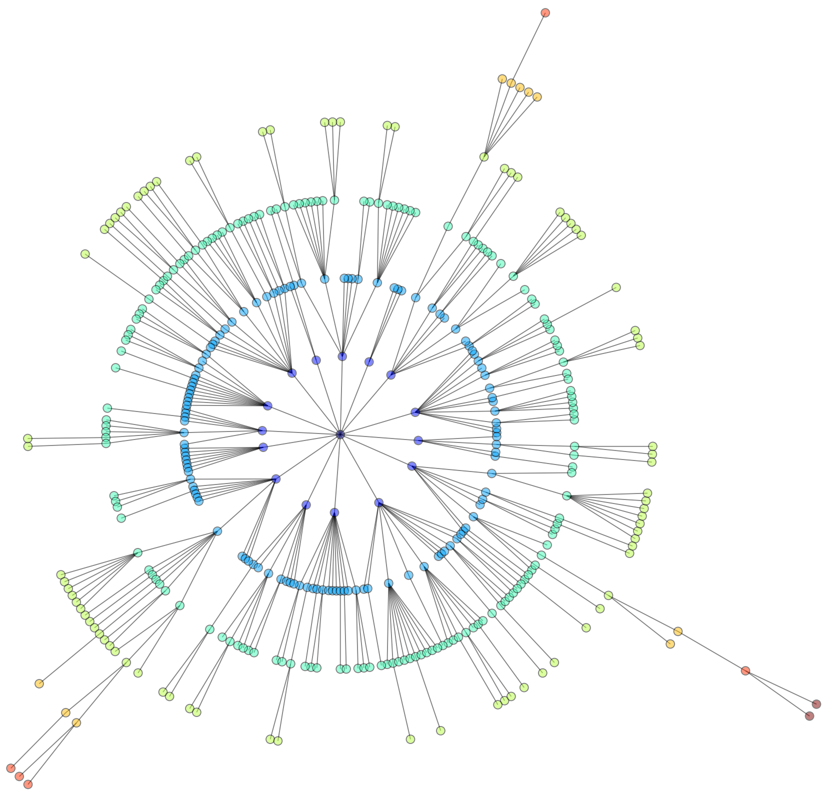
The above examples often present common difficulties to optimization algorithms due to their large scale. During the past decade most optimization effort aimed at these problems focused on development of efficient first-order methods, such as accelerated proximal gradient methods [1–3], block coordinate descent methods [4–7] and alternating directions methods [8]. These methods enjoy low per-iteration complexity, but typically have slow local convergence rates. Their performance is often hampered by small step sizes. This, of course, has been known about first-oder methods for a long time, however, due to the very large size of these problems, second-order methods are often not a practical alternative. In particular, constructing and storing a Hessian matrix, let alone inverting it, is prohibitively expensive for values of n larger than 10000, which often makes the use of the Hessian in large-scale problems prohibitive, regardless of the benefits of fast local convergence rate.

Hybrid approach and top-down scheme A fairly straightforward way to overcome the shortcomings of the two classes of methods is to adopt a hybrid approach. This is demonstrated in Chapter 2 where we propose a flexible classification system called HiClass for inducing hierarchical structure in the prediction results. It strikes a balance between the efficiency of first order methods and the accuracy enjoyed by second order methods. Hierarchical structure appears in many real-world recommendation and targeting systems. In display-ads targeting, for example, each user is associated with a set of labels such as sports, football, vehicles, luxury cars, etc., and each label contains ads campaign that this user is likely to interact with. It is often the case that the labels are not independent of each other, and that their interdependence can be characterized by a connected (tree) structure (see Figure 1.1), e.g., football is the child of sports. The task then is to assign each user with a set of labels that both appeal to the user and satisfy the underlying taxonomy. Overlooking such structure not only undermines the predictive performance (loss of information), but also makes the results hard to interpret, e.g., predicting that a user is interested in football but not in sports.

HiClass exploits the additional knowledge in the structure of the labels, and applies a recursive top-down approach: first discriminate the subsets of labels at the top level of the hierarchy before going down the next level to discern the labels (or sets of labels) in those subsets. This process is repeated until reaching the bottom level. The overall task is decoupled into a series of smaller ones, with each one defined for predicting membership to each class and trained independently with no attention to the hierarchy.

HiClass employs a scalable first-order optimization algorithm to train the high-level classifiers, and a fast second-order algorithm for the low-level classification problems. The main aim of this hybrid approach is to alleviate the shortcomings of the

Figure 1.1: Visualization of taxonomy among labels in dataset *ads*.



two methods, i.e. lower accuracy of the first-order method and slower speed of the second-order method, by adjusting the algorithms to the training data sizes that vary from node to node.

It exploits a typical scenario in such top-down approach that size of the data becomes smaller at the lower levels of the hierarchy, as the classes become more specific. Hence, using the same optimization method to solve the classification problems through the hierarchy without accounting for different data distribution can be either inefficient or inaccurate – with IPM we need to solve a sequence of large-scale Quadratic Programming (QP) problems that can quickly become intractable simply because of the sheer size of data, and with first-order methods it can be painfully slow to reach the solution at nodes in the lower level where higher accuracy is often desirable.

Experiments on the text categorization data set RCV1-v2 show that HiClass achieves better predictive performance (30% more) than first-order method SCD [9] given the same timing budget, and scales to much larger data sets than second-order method IPM [10]. More details about this work are described in [11, 12], and [13] that appears in Learning and Intelligent Optimization Conference.

Structured optimization and parameter learning The problem with the top-down scheme in HiClass is that it predicts the part of the label vector that correspond to the upper levels of the taxonomy first, without considering the lower levels. This clearly is a suboptimal approach because the optimization is not performed over the entire set of variables.

In this work we will introduce a two-stage hierarchical multi-label learning model that improves upon the top-down scheme to achieve a richer and more powerful hierarchical representation. The proposed model combines base learners of each label with an optimal decoding scheme which induces the hierarchical structure among the outputs of base learners by solving a convex optimization problem based on structured sparse learning.

Given a predictor W, computed during training, and a input vector x, the product Wx is typically an unstructured dense vector. The true label vector, however, is not only sparse (with the non zeros in y denoting positive labels) but has to satisfy the hierarchical structure. In prior approaches a unstructured sparse vector has been recovered given Wx, for example, by using a sparsity inducting ℓ_1 norm.

In this work we make use of a particular variant of group-lasso penalty function [14], called hierarchical norm. This norm is constructed with a recursive group structure based on the given taxonomy. Minimizing objective (1.0.1) equipped with the hierarchical norm, for instance, yields in variable x a particular sparse pattern that respects the underlying taxonomy.

Compared to other hierarchical inducing methods, the use of hierarchical norm turns out to be more powerful and enables a richer hierarchical representation on any input vector to be induced in a very efficient way. But to make it work well in practice is not easy and often requires carefully choosing the parameter/weight for each group. For ℓ_1-norm this is easily done using a grid search since it has only one parameter determining the sparsity. But for hierarchical norm the number of parameters can be several hundreds, depending on the number of subtrees in a given taxonomy.

Hence the two-stage hierarchical model we propose chains the learning and inducing in a single pipeline where each can be efficiently approached. In particular, we base the first-stage training on the multi-label Empirical Risk Minimization (ERM) framework [15] where various convex loss functions can be employed, i.e., least squares, logistic, or hinge loss. The output of the first problem is then used as the input of the second stage. The hierarchical weights are learned in the second stage by directly optimizing popular metrics such as f1, jaccard similarity score or hamming loss, while taking into account of both the classifiers and the encoded hierarchical information present in the training data. While the second problem still is a bi-level optimization problem, its dimension is now significantly reduced to the order of the number of labels that is well under 1000 in this work. In this work we approach the bi-level problem as a black-box optimization problem, which we solve using an efficient derivative-free optimization methods [16]. The details can be found in Section 3.4.

Finally, we demonstrate the model on large-scale real-world data sets from document classification, as well as ad targeting domains that involves two million users' daily activities. The results demonstrate the potential of the proposed model.

1.2 Adaptive active-set and communication-efficient coordinate descent

HiClass improves the efficiency and scalability of hierarchical classification by launching different algorithms to optimize sparse logistic regression for different nodes of the hierarchy, thus overcoming the shortcomings of both first-order and second-order methods. The next question we ask is whether we can borrow from these ideas to improve the existing algorithms for sparse logistic regression, or any sparsity-inducing models with objective F (1.0.1) in general. The objective F in these settings are often constructed from a N by n data matrix that can present common difficulties to optimization algorithms due to the large scale of instances N or features n. Here we propose a fast, memory- and communication-efficient algorithm that employs a two-loop scheme designed to directly address such cases when both N and n are large and that combines first- and second-order information within one single optimization algorithm.

This two-loop scheme implements the global-convergent framework that is discussed in Chapter 4 and is designed to separate the optimization procedures into two parts, the outer loop whose complexity depend on both n and N but can be easily distributed, and the inner loop that only relies on n but is sequential in nature. We further propose a self-adaptive active-set selection mechanism, which enhances the inner loop procedures such that the dependence on n can be replaced by dependence on the support/non-zero subspace of the optimal solution x^*. The improvement is significant for (5.1.1) when feature selection is desirable, i.e., the support of x^* is small although n is large. Finally, we show that establishing convergence rate guarantee of the active-set method is simple by applying the theory presented in Chapter 4 for a particular quadratic function Q constructed according to the active-set selection.

The algorithm proposed in this work is backed by convergence analysis presented in Chapter 4 yet outperforms popular methods recently proposed for sparse optimization [4, 17–19]. These methods explore the following special properties of the sparse problems: at optimality many of the elements of x are expected to equal 0, hence methods which explore active set-like approaches can benefit from small sizes of subproblems. Whenever the subproblems are not small, these new methods exploit the idea that the subproblems do not need to be solved accurately. In particular several successful methods employ coordinate descent to approximately solve the subproblems. Other approaches to solve Lasso subproblem were considered in [19], but none generally outperform coordinate descent. [4] proposes a specialized GLMNET [5] implementation for sparse logistic regression, where coordinate descent method is applied to the unconstrained Lasso subproblem constructed using the Hessian of $f(x)$ – the smooth component of the objective $F(x)$. Two main improvements increase efficiency of GLMNET for larger problems – exploitation of the special structure of the Hessian to reduce the complexity of each coordinate step so that it is linear in the number of training instances, and a two-level shrinking scheme proposed to focus the minimization on smaller subproblems. Similar ideas are used in [17] in a specialized algorithm called QUIC for sparse inverse covariance selection.

The algorithm we propose is communication-efficient such that, unlike QUIC and GLMNET, the inner coordinate descent does not endure extra communications even when the data is distributed across a network, is *general purpose* such that its efficiency does not rely on the Hessian structure, and is backed by theoretical analysis and convergence rates [20] yet outperforms the state-of-the-art specialized methods such as QUIC and GLMNET. As these two and other methods, mentioned above, we consider the following general framework:

- At each iteration $F(x)$ is approximated, near the current iterate x^k, by a convex quadratic function $Q(x)$.
- Then an algorithm is applied to optimize (approximately) the function $Q(x)$, to compute a trial point.
- The trial point is accepted as the new iterate if it satisfies some sufficient decrease condition.

- Otherwise, a different model $Q(x)$ may be computed, or a line search applied to compute a new trial point.

We make use of similar ideas as in [4] and [17], but further improve upon them to obtain efficient general schemes, which apply beyond the special cases of sparse logistic regression and covariance selection and for which convergence rates can be established. In particular, the active-set selection mechanism we propose consists of two phases, the greedy phase and the convergent phase, and the transition between phases is self-adaptive. In the greedy phase the mechanism adjusts the size of the active set adaptively, while in the convergent phase it includes all entries with dual constraint violations. Convergence can be established for the convergent phase by applying the theory in [20]. Compared to QUIC and GLMNET, this mechanism generates smaller inner problems through the process, as we demonstrate empirically.

We employ a compact representation of LBFGS methods [21] to construct the function Q in our algorithm. While the use of LBFGS for (5.1.1) is not new, e.g., [22], to use it in the two-loop framework along with the active-set selection and coordinate descent is novel and has several implications both in theory and in practical algorithm design: 1) it separates the data from the inner solver such that the inner solver can be programmed in a generic way for a vector, the gradient, and two matrices G and $\hat{G}$ as we will see later, with no dependence on the actual form of function $L(x)$; 2) data is solely handled in outer iterations which can also be programmed in a generic way while specifically optimized for how data is stored, e.g., in memory, on disk or across the network; 3) by carefully exploiting the low rank structure present in the compact representation of LBFGS the complexity of each coordinate descent step can be reduced to a constant! 4) the convergence of LBFGS with active-set selection and inexact subproblem minimization has not been analyzed before, but can be established, as we will see, by applying the theory in [20].

One of the key for the algorithm to be fast relies on efficiently computing compact LBFGS representation, which requires constantly performing mutating operations on large matrices. A naive implementation that allocates and frees new chunk of memory every iteration can be several order slower due to increasing memory fragmentation, allocation overhead, and most importantly memory and cache inefficiency which can further slow down the coordinate descent process. In this work we discuss a mutable matrix type that is designed to exploit the specific mutating patterns present in LBFGS such that the mutating operations are done repeatedly on a single chunk of continuous memory, without consuming extra memory and with ultra-low latency.

1.3 Complexity of inexact proximal Newton methods and randomization

The algorithm proposed in Chapter 5 belongs to a class of wide range of algorithms called inexact proximal Newton-type methods. The convergence behaviors and overall complexity of inexact proximal Newton-type methods are further analyzed in Chapter 4. The contributions of this work are mainly twofolds:

1. We discuss theoretical properties of the above framework in terms of global convergence rates. In particular, we show that if we replace the line search by a prox-parameter update mechanism, we can derive sublinear global complexity result for the above methods under mild assumptions on Hessian approximation matrices, which can include diagonal, quasi-Newton and limited memory quasi-Newton approximations. We also provide the convergence rate for the case of inexact subproblem optimization. It turns out that standard global convergence analysis of proximal gradient methods (see [2, 23]) does not extend in a natural way to proximal quasi-Newton frameworks, hence we use a different technique derived for smooth optimization in [1, 24, 25], in a novel way, to obtain the global complexity result.

2. The heuristic of applying $l(k)$ passes of coordinate descent to the subproblem is very useful in practice, but has not yet been theoretically justified, due to the lack of known complexity estimates. Here we use probabilistic complexity

bounds of randomized coordinate descent to show that this heuristic is indeed well justified theoretically. In particular, it guarantees the sufficiently rapid decrease of the error in the subproblems (with some probability) and hence allows for sublinear global convergence rate to hold for the entire algorithm (in expectation). This gives us the first complete global convergence rate result for the algorithmic schemes for practical (inexact) proximal Newton-type methods. Moreover, using the new analysis from [24–26] we are able to provide lower overall complexity bound than the one that follows from [23].

Let us elaborate a bit further on the new approaches and results developed in this chapter and discuss related prior work.

In [27] Byrd et al. propose that the methods in the framework described above should be referred to as sequential quadratic approximation (SQA) instead of proximal Newton methods. They reason that there is no proximal operator or proximal term involved in this framework. This is indeed the case, if a line search is used to ensure sufficient decrease. Here we propose to consider a prox term as a part of the quadratic approximation. Instead of a line search procedure, we update the prox term of our quadratic model, which allows us to extend global convergence bounds of proximal gradient methods to the case of proximal (quasi-)Newton methods. The criteria for accepting a new iteration is based on sufficient decrease condition (much like in trust region methods, and unlike that in proximal gradient methods). We show that our mechanism of updating the prox parameter, based on sufficient decrease condition, leads to an improvement in performance and robustness of the algorithm compared to the line search approach as well as enabling us to develop global convergence rates.

Convergence results for the proximal Newton method have been shown in [28] and [27] (with the same sufficient decrease condition as ours, but applied within a line search). These papers also demonstrate super linear local convergence rate of the proximal Newton and a proximal quasi-Newton method. Thus it is confirmed in [27, 28] that using second order information is as beneficial for problems of the form (4.1.1) as it is for the smooth optimization problems. These results apply to our framework when exact Hessian (or a quasi-Newton approximation) of $f(x)$ is used to construct $q(x)$ (and if the matrices have bounded eigenvalues). However, theory in [27, 28] does not provide global convergence rates for these methods, and just as in the case on smooth optimization, the super linear local convergence rates, generally, do not apply in the case of LBFGS Hessian approximations.

The convergence rate that we show is sub linear, which is generally the best that can be expected from a proximal (quasi-)Newton method with no assumptions on the accuracy of the Hessian approximations. Practical benefits of using LBFGS Hessian approximations is well known for smooth optimization [21] and have been exploited in many large scale applications. In this paper we demonstrate this benefit in the composite optimization setting (4.1.1). Some prior work showing benefit of limited memory quasi-Newton method in proximal setting include [29, 30]. We also emphasize in our theoretical analysis the potential gain over proximal gradient methods in terms of constants occurring in the convergence rate.

To prove the sub linear rate we borrow a technique from [24–26]. The technique used in [2] and [23] for the proof of convergence rates of the (inexact) proximal gradient method do not seem to extend to general positive definite Hessian approximation matrix. In a related work [30] the authors analyze global convergence rates of an accelerated proximal quasi-Newton method, as an extension of FISTA method [2]. The convergence rate they obtain match that of accelerated proximal gradient methods, hence it is a faster rate than that of our method presented here. However, they have to impose much stricter conditions on the Hessian approximation matrix, in particular they require that the difference between any two consecutive Hessian approximations (i.e., $H_k - H_{k+1}$) is positive semidefinite. This is actually in contradiction to FISTA's requirement that the prox parameter is never increased. Such a condition is very restrictive as is impractical. In this paper we briefly show how results in [2] and [23] can be extended under some (also possibly strong) assumptions of the Hessian approximations, to give a simple and natural convergence rate analysis. We then present an alternative analysis, which only requires the Hessian approximations to have bounded eigenvalues. Moreover, the bounds on the subproblem optimization error in our analysis are looser than those in [23].

Finally, we use the complexity analysis of randomized coordinate descent in [31] to provide a simple and efficient stopping criterion for the subproblems and thus derive the total complexity of proximal (quasi-)Newton methods based on randomized coordinate descent to solve Lasso subproblems.

1.4 A brief outline of the thesis

The remainder of this thesis is organized as follows.

In Chapter 2 (see [11, 12], and [13]; joint work with Siemens Research) various applications of machine learning models are discussed, particularly for sparse logistic regression. Three specialized algorithm for optimizing large sparse logistic regression problem are analyzed and compared, and a hybrid classification system that incorporates those methods is proposed for inducing hierarchical structure in classification results. In the last section the application of support vector machine and non-negative matrix factorization in time series forecasting is also described.

In Chapter 3 (joint work with Yahoo Labs, submitted to AAAI) we discuss a particular application of structured learning in large-scale online-ad targeting. We start by introducing the optimal decoding scheme and the hierarchical operator. The proposed learning algorithm and its complexity are discussed in Section 3.4. Other relevant hierarchical methods and their connections to the hierarchical operator are discussed in Section 3.5 and examples are given at the end of the section to illustrate the difference between those methods. Finally we present numerical experiments.

In Chapter 4 (submitted to Mathematical Programming Series A) we discuss the theoretical properties of the above introduced algorithmic framework in terms of global convergence rates. We start with the algorithmic framework. Then, we present some of the assumptions and discussions and convergence rate analysis based on [2] and [23]. We show the new convergence rate analysis using [24–26] for exact and inexact version of our framework. We then extend the analysis for the cases where inexact solution to a subproblem is random and in particular to the randomized coordinate descent.

In Chapter 5 (a preliminary version of which appearing in NIPS workshop, submitted to Mathematical Programming Series C) we present a fast, communication- and memory-efficient algorithm that implements the global-convergent framework proposed in Chapter 4. We begin with outlining the algorithm and defining basic terminologies. We then discuss: 1) compact LBFGS and its low-rank structure, paying particular attention to the mutable matrix type; 2) active-set strategies and the convergence guarantee; and 3) different termination conditions for the inner loop and their implications on the global convergence performance. Finally, we present numerical studies of different active-set methods and termination conditions on algorithm running time.

Chapter 2

Machine Learning Applications of Convex Optimization

2.1 Introduction

In the optimization literature the second order methods, such as the Interior Point Methods (IPM), are well known for their ability to solve QP problems to high accuracy. However, in applications like classifying web content and text documents in a hierarchical structure where large amounts of data exist at higher levels of the hierarchy, the use of IPMs may not be the most appropriate. This is because IPMs require the solution of a system of linear equations (i.e., the Newton system) at every iteration and, as a result, their performance can quickly deteriorate as the size of that system becomes very large.

The recent burst of research at the interface of optimization and machine learning has produced a large family of algorithms that require only the first-order (gradient) information and possibly work on a small subset of the training data in each iteration. Hence, the per-iteration work of these algorithms is small, making them suitable for training on large-scale data.

In this chapter we demonstrate the benefit of combining the first order method and the second order method in a hybrid approach for large scale hierarchical classification. The sparse logistic regression model is first of all introduced and adapted to the optimization context. We then discuss and compare the complexity difference between two Coordinate Descent approaches [9, 32] and a variant of Interior-Point Method (IPM) [10], all of which are designed specifically for sparse logistic regression. The practical performance of the three algorithms are later compared in a hybrid hierarchical classification problem, which combines them in a top-down approach in order to achieve a good balance between efficiency and accuracy. We finally discuss the use of support vector machine and non-negative matrix factorization in time series forecasting problem. The efficient first order methods for solving the above problems are briefly described in the end.

2.2 Sparse Logistic Regression

The objective function of sparse logistic regression is given by

$$\min_{w} \quad F(w) = \lambda ||w||_1 + \frac{1}{N} \sum_{n=1}^{N} \log(1 + \exp(-y_n \cdot w^T x_n)) \tag{2.2.1}$$

where the first component is a regularization term to select relevant features and enforce sparsity in the solution or classifier w, and the second term $\frac{1}{N}\sum_{n=1}^{N} \log(1 + \exp(-y_n \cdot w^T x_n))$ is the average logistic loss function. For each training pair $\{x_n, y_n\}$, the logistic loss function is expected to return a positive value (note that the return of the exponent equation inside the log function is always larger than 1), as a measure of whether the predicting label given by $w^T x_n$ corresponds with the "true" label y_n assigned manually before training. Particularly, a different sign between $w^T x_n$ and y_n will force the value given by $\exp(-y_n \cdot w^T x_n)$ to quickly go much larger than 1, resulting in a large loss as a whole; on the other hand, a similar sign, indicating a good prediction, will lead to a small loss by driving $\exp(-y_n \cdot w^T x_n)$ below 1 all the way close to zero. By summing the logistic loss for each sample across the whole training set and dividing it by the cardinality of that set, we obtain the average logistic loss defined on that training set, and by minimizing the average loss, we are able to obtain the classifier that guarantees to return the smallest error on the training set. A regularization term $\lambda||w||_1$ is also added to the objective function such that it controls the complexity of the classifier by selecting the most relevant features based on the current training set, which contains important information on how the features relate to each other in predicting the right labels.

Gradient and Hessian To compute the gradient and the Hessian of logistic loss, without loss of generality, we reformulate the problem (5.7.1) as the following by squeezing λ and $1/N$ into one parameter C:

$$\min_{w} \quad F(w) = ||w||_1 + C\sum_{n=1}^{N} \log(1+\exp(-y_n \cdot w^T x_n)) \tag{2.2.2}$$

Recall our loss function L is given by $L(\mathbf{w}) = C\sum_{k=1}^{N} l(\mathbf{w};\mathbf{x_k}, y_k) = C\sum_{k=1}^{N}\log(1+\exp(-y_k \cdot \mathbf{w}^T\mathbf{x_k}))$ where N is the size of our sample points. The partial derivative of $L(\mathbf{w})$ with respect to w_j can be thus calculated by

$$\frac{\partial L}{\partial w_j} = C\sum_{k=1}^{N} \frac{-1}{1+e^{y_k \mathbf{w}^T\mathbf{x_k}}} y_k(\mathbf{x_k})_j = C\mathbf{G}^T\mathbf{X_j} \tag{2.2.3}$$

where

$$\mathbf{B} = \begin{bmatrix} G_1 \\ \vdots \\ G_N \end{bmatrix} \in \mathbf{R}^N, \quad G_k = \frac{-y_k}{1+e^{y_k \mathbf{w}^T\mathbf{x_k}}} \tag{2.2.4}$$

and $\mathbf{X}_j$ is the jth column of $\mathbf{X}$ as in

$$\mathbf{X} = \begin{bmatrix} \mathbf{X}_1 & \dots & \mathbf{X}_p \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} \in \mathbf{R}^{N\times p} \tag{2.2.5}$$

Hence, the gradient of $L(\mathbf{w})$ can be written as $\nabla^T L(\mathbf{w}) = C\mathbf{G}^T\mathbf{X}$. Similarly, from (2.2.3) we can obtain the second partial derivatives of $L(\mathbf{w})$

$$\frac{\partial L}{\partial w_j \partial w_i} = C\sum_{k=1}^{N} \frac{e^{-y_k \mathbf{w}^T \mathbf{x_k}}}{(1+e^{-y_k \mathbf{w}^T \mathbf{x_k}})^2} y_k^2 (\mathbf{x_k})_j (\mathbf{x_k})_i \tag{2.2.6}$$

$$= C\sum_{k=1}^{N} \frac{e^{-y_k \mathbf{w}^T \mathbf{x_k}}}{(1+e^{-y_k \mathbf{w}^T \mathbf{x_k}})^2} (\mathbf{x_k})_j (\mathbf{x_k})_i \tag{2.2.7}$$

We can then obtain Hessian by $B = C\mathbf{X}^T\mathbf{D}\mathbf{X}$ where $\mathbf{D} \in \mathbf{R}^{N\times N}$ is a diagonal matrix with $D_{kk} = \frac{e^{-y_k \mathbf{w}^T \mathbf{x_k}}}{(1+e^{-y_k \mathbf{w}^T \mathbf{x_k}})^2}$. So to compute the full Hessian matrix B, even if we assume the matrix D is available to use after computing the gradient because the component $e^{-y_k \mathbf{w}^T \mathbf{x_k}}$ we need for D is also required for the gradient, it still takes $O(N^2 p) + O(Np^2)$ flops to finish the computations.

2.2.1 Optimization Algorithms

There are many algorithms tailored for solving sparse logistic regression. Here we are going to discuss three of them, two Coordinate Descent approaches [9, 32] and an Interior-Point Method (IPM) [10]. One of the main difference between those methods lies in the way they handle the second order information. Particularly, the first Coordinate Descent approach (CD1) we are proposing can be considered as a first order method using only gradient information; IPM is the pure second order method solving a Newton system at each step and the second Coordinate Descent approach (CD2) acts more like a quasi-Newton method where the per iteration complexity is cheap because the second order matrix, the actual Hessian in this case rather than the Hessian approximation, is handled efficiently and only matrix-vector multiplications are involved. Next we give an overview of each of the above mentioned algorithms.

An Interior-Point Method

To apply interior-point method to sparse logistic regression, we begin with transforming the standard unconstrained non-smooth formulation to one with linear smooth constraints and smooth objective, as following

$$\begin{aligned} \min_{w,u} \quad & L(w) + \lambda \sum_{i=1}^{p} u_i \\ \text{s.t.} \quad & -u_i \le w_i \le u_i, \quad i = 1, ..., p \end{aligned} \tag{2.2.8}$$

where $L(w) = \frac{1}{N}\sum_{k=1}^{N} l(w; x_k, y_k)$ and λ is the average logistic loss function the regularization parameter. Note that at the optimality we must have $u_i = |w_i|$, in which case the objective in (2.2.8) becomes the same as standard sparse logistic regression objective (5.1.1). Hence, we say problem (2.2.8) is equivalent to the standard formulation. After adding slack variables $\bar{s}_i, \underline{s}_i$, we reformulate problem (2.2.8) into one with only equality constraints.

$$\begin{aligned} \min_{w,u} \quad & L(w) + \lambda \sum_{i=1}^{p} u_i - \mu \sum_{i=1}^{p} \log \underline{s}_i - \mu \sum_{i=1}^{p} \log \bar{s}_i \\ \text{s.t.} \quad & w_i - u_i + \bar{s}_i = 0, \\ & -w_i - u_i + \underline{s}_i = 0, \\ & i = 1, ..., p \end{aligned} \tag{2.2.9}$$

where the logarithmic barrier is added in the objective to force slack variables to stay positive without explicitly adding bound constraints, and $\mu > 0$ is a parameter defining the *central path* as $\mu \to 0$. To solve problem (2.2.9) by interior-point method, we can iteratively apply Newton method to its optimality conditions for a sequence of barrier parameters such that the limit of the sequence goes to zero. Noticing that in the constraints we have $\bar{s}_i = u_i - w_i$ and $\underline{s}_i = w_i + u_i$, we substitute these equalities into the objective, thus eliminating both the constraints and slack variables and obtaining the following unconstrained smooth convex problem

$$\min_{w,u} \quad \phi_\mu(w, u) = L(w) + \lambda \sum_{i=1}^{p} u_i - \mu \sum_{i=1}^{p} \log(u_i^2 - w_i^2)$$

whose optimality condition is simply its gradient equal to zero

$$\nabla \phi_\mu(w^*, u^*) = 0 \tag{2.2.10}$$

In primal interior-point method, Newton method is used to solve (2.2.10) for a search direction and line search, often with backtracking, is used to determine the step length before we decrease μ and solve (2.2.10) again at the new iterate. That sequence of iterates we compute forms the *central path*, which eventually leads us to the optimality. The main cost of IPM lies in forming and solving the Newton system, which takes $O(N^2 p)$ flops [10].

Two Coordinate Descent Methods

Basic Idea Following the standard approach, we divide the objective in (5.1.1) into two parts

$$\min_{w} \ F(w) = g(w) + L(w), \tag{2.2.11}$$

where $g(w) = \|w\|_1$ is convex but non-differentiable and the second component $L(w) = C \sum_{k=1}^{N} l(w; x_k, y_k)$ is both smooth and convex. With w_k as the current iteration, the problem is to calculate the direction d_k to progress to our next iteration w_{k+1}

$$\min_{d_k} \ F(w_k + d_k) = g(w_k + d_k) + L(w_k + d_k) \tag{2.2.12}$$

Note that F is non-smooth because g is non-differentiable at zeros, so we cannot directly use gradient information of F to calculate the direction. Instead, we consider the second-order approximation of L by Taylor expansion since L is twice differentiable:

$$\begin{aligned} \min_{d_k} \ q(d_k) = & L(w_k) + \nabla^T L(w_k) d_k + \\ & \frac{1}{2} d_k^T H_k d_k + g(w_k + d_k) \end{aligned} \tag{2.2.13}$$

To facilitate the discussion, we will ignore the constant for now and use $L_k, d, q_k(d), g_k(d)$ instead of $L(w_k), d_k, q(d_k), g(w_k + d_k)$:

$$\min_{d} \ q_k(d) = \nabla^T L_k d + \frac{1}{2} d^T H_k d + g_k(d) \tag{2.2.14}$$

Let us now consider the coordinate step on (2.2.14). We start by randomly choosing a coordinate, say j, to update. Let $d = ze_j$ where z is a scalar, and e_j is a vector with the same dimension of d. All coordinates of e_j are zeros except coordinate j being 1. Substituting $d = ze_j$ into (2.2.14), we obtain

$$\min_{ze_j} \; q_k(ze_j) = \nabla^T L_k ze_j + \frac{1}{2} ze_j^T H_k ze_j + g_k(ze_j) \tag{2.2.15}$$

since $e_j^T H_k e_j = (H_k)_{jj}, \nabla^T L_k e_j = (\nabla^T L_k)_j$, and $g_k(ze_j) = |(w_k)_j + z|$, (2.2.15) is in fact a one-dimension quadratic problem:

$$\min_{z} \; q_k(ze_j) = (\nabla^T L_k)_j z + \frac{1}{2}(H_k)_{jj} z^2 + |(w_k)_j + z| \tag{2.2.16}$$

It is well known that (2.2.16) has a simple closed-form solution

$$z = \begin{cases} \frac{(\nabla^T L_k)_j + 1}{-(H_k)_{jj}} & \text{if } (\nabla L_k)_j + 1 \leq (H_k)_{jj}(w_k)_j, \\ \frac{(\nabla^T L_k)_j - 1}{-(H_k)_{jj}} & \text{if } (\nabla L_k)_j - 1 \geq (H_k)_{jj}(w_k)_j, \\ -(w_k)_j & \text{otherwise.} \end{cases} \tag{2.2.17}$$

Finally, we calculate step length α based on the Armijo rule [33] in the line search procedure:

$$\begin{aligned} & F(w_k + \alpha d_k) - F(w_k) \\ \leq & \sigma\alpha((\nabla^T L_k)_j z + |(w_k)_j + z| - |(w_k)_j|) \end{aligned} \tag{2.2.18}$$

The First Order Variant (CD1) To avoid calculating $(H_k)_{jj}$ in (2.2.16), we replace it with an upper bound β on the second derivative of the loss such that

$$\beta \geq (H_k)_{jj}, \quad \forall k \text{ and } j \tag{2.2.19}$$

and (2.2.16) now becomes

$$\min_{z} \; \hat{q}_k(ze_j) = (\nabla^T L_k)_j z + \frac{1}{2}\beta z^2 + |(w_k)_j + z|, \tag{2.2.20}$$

which is an upper-bound function of $q_k(ze_j)$ because any z that leads to a decrease of $\hat{q}$ will inevitably follow by a decrease of q since

$$0 \leq \hat{q_k}(0) - \hat{q_k}(ze_j) \leq q_k(0) - q_k(ze_j). \tag{2.2.21}$$

Replacing the $(H_k)_{jj}$ in (2.2.17) by β, we obtain the closed-form solution of (2.2.20).

Consider the complexity of a full coordinate descent. Updating one coordinate of w_k requires only constant time, so the main cost is updating $\nabla^T L_k$, which takes $O(N)$ given that only one coordinate in w_k is changed. Hence, the total complexity is $O(Np)$ to update all p coordinates in w_k.

The Second Order Variant (CD2) Unlike the above methods, which update ∇L and H as soon as one coordinate is updated, CD2 compute the optimal solution of (2.2.14) before updating the objective function. That is, iteratively applying coordinate descent update to (2.2.14) until it satisfies the optimality conditions. Each coordinate update z is computed by solving the following minimization problem

$$\begin{aligned} \min_z \ & q_k(d_s + ze_j) \\ =& [\frac{1}{2}(H_k d_s)_j + (\nabla^T L_k)_j]z + \\ & \frac{1}{2}(H_k)_{jj}z^2 + |(w_k)j + (d_s)_j + z| \end{aligned} \tag{2.2.22}$$

A closed-form solution of (2.2.22) can be obtained in a very similar fashion as (2.2.17).

Following similar complexity analysis of CD1, we end up with the same $O(Np)$ flops in order to apply a full coordinate descent in CD2. Note that to achieve that complexity, we need to take advantage of some careful arrangement and caching of the computations [32].

The computational complexity per iteration for each of the above three algorithms is summarized in Table 2.1.

	IPM	CD1	CD2
Flops	$O(pN^2)$	$O(pN)$	$O(pN)$

Table 2.1: Per Iteration Complexity

2.3 HiClass - An efficient Hierarchical Classification system

Introduction Hierarchical classification is an extension to the multi-class classification by taking into account the hierarchical structure of the labels. This structure is ignored in the standard, or flat multi-classification since each class is treated equally in a flat fashion. However, in many practical applications like classifying web contents [34] and text documents [35, 36], such hierarchical structure of the labels is particularly prevalent and labels can often be more appropriately understood as sets of labels with closely related labels grouped in the same set [37]. Thus labels from the same set are more difficult to discriminate than those from different sets, and it can be beneficial to train a classifier specifically for a set of labels in order to capture the subtle distinction between them. Based on this intuition, hierarchical classification exploits the additional knowledge in the structure of the labels by applying a recursive top-down approach: first discriminate the subsets of labels at the top level of the hierarchy before going down the next level to discern the labels (or sets of labels) in those subsets. This process is repeated until reaching the bottom level. In this paper, we propose a highly flexible and efficient framework for hierarchical classification by combining the strengths of first-order and second-order optimization methods through the top-down approach.

Motivation The primary aim of our proposed framework, HiClass, is to provide a flexible solution which is well balanced between efficiency and accuracy for hierarchical classification. We achieve this by adapting the use of second order information to the number of data presented in different levels of the hierarchy. To train classifiers in hierarchical classification, we are faced with solving Quadratic Programming (QP) problems whose sizes depend on the number of training data. Nodes that exist at higher levels of the hierarchical tree usually contain many more data than the nodes at the lower levels of the tree. Using the same optimization method to solve the classification problems at different nodes of the tree would result in computational inefficiencies and inaccuracies.

A typical scenario in top-down-based hierarchical classification is that although there is a large amount of data at the high levels of the hierarchy, the size of the training sets are considerably smaller at the low levels, since the classes become much more specific [34–37].

The above observations have motivated us to apply an efficient and scalable first-order optimization algorithm to train the high-level classifiers, and an accurate second-order algorithm for the low-level classification problems. The main aim of this hybrid approach is to alleviate the shortcomings of the two methods, i.e. lower accuracy of the first-order method and slower speed of the second-order method, by adaptively adjusting the algorithms to the training data sizes that are dynamically changing from node to node. We remark that although first-order methods often are not able to solve a problem as accurately as methods using second-order information (e.g. IPM), the large amount of training data available at top-level nodes help compensate the lower accuracy in the solution.

Related Methods In many applications, such as text classifications and fault detection, class labels are arranged as nodes in a tree to represent a given hierarchical relation. The traditional way to approach this kind of problem is to simply ignore the hierarchical structure and treat it as a multi-class classification problem. This allows us to take advantage of existing techniques such as multi-class Support Vector Machine (SVM) [38]. Several variants of this formulation have recently been proposed to take into account the extra information from the hierarchy of classes to improve the loss function [35, 39], but two major drawbacks still remain – inconsistencies in child-parent relations are difficult to prevent and inaccuracy in solving the training problem results in an inferior generalization quality. Accordingly, several methods including the top-down procedure [34, 36, 40], the bottom-up procedure [41] and the Bayesian network [42] have been proposed.

In this paper, we focus on a novel hybrid approach for training different nodes in the hierarchy. We combine different optimization techniques in the hybrid approach in order to establish a good enough balance between efficiency and accuracy. We also decouple the problem into a series of classification problems defined for each class and train each one independently with no regard to the hierarchy to obtain the classifier that is responsible for predicting membership to that particular class by means of a binary or real-valued output. This provides us with an opportunity to tailor optimization techniques to each problem and solve them in parallel, thus avoiding directly optimizing a single problem that can be much more complex and difficult to even store in memory. From there we discuss the important use of second order information in our hybrid approach and explore ways to build it into our HiClass system.

2.3.1 The Hierarchical Classification Framework

Let $\mathcal{X} \subset \mathcal{R}^p$ be the instance domain and let $\mathcal{Y} = \{1, ..., m\}$ be the index set of categories. We are interested in categories arranged in a tree structure. Each category, or node, in that tree structure is denoted by a unique number $i \in \mathcal{Y}$ (0 for the root of the tree). $\bar{\mathcal{Y}} = \mathcal{Y} \cup \{0\}$ is the set of all the nodes in the tree. Every node i has a unique parent $\mathcal{P}(i)$, a set of children $\mathcal{C}(i)$ and siblings $\mathcal{S}(i)$. Let $\mathcal{S}^+(i) = \mathcal{S}(i) \cup \{i\}$. Also let $\mathcal{L}$ be the set of leaf nodes with no children:

$$\mathcal{L} = \{i \in \mathcal{Y} \mid \mathcal{C}(i) = \emptyset\} \tag{2.3.1}$$

A training set is given as $\{(x_k, y_k)\}_{k=1}^{N} \in \mathcal{X} \times \mathcal{L}$. Note that every instance x_k is labeled as y_k from leaf node set $\mathcal{L}$ instead of $\mathcal{Y}$. One implication for this is illustrated in Figure 2.1, where an instance x_k is automatically attached with node 1 and then the root node if it is categorized to node 4. Let a *path* denotes a set of labels with parent-child relations, and for every instance x_k, let $\mathcal{P}_{ath}(k)$ be the *path* that includes y_k. Then in this case $\mathcal{P}_{ath}(k)$ will be $\{4,1,0\}$ (remember 0 is the root node). If we further assume that all the instances are categorized until it is found in one and only one leaf node, this leaf node alone is enough to represent the whole path an instance is attached with, which is why we assign every instance only one label y_k. So in previous example, y_k is 4, the index of leaf node is x_k classified to.

The challenge here is to learn a function $f : \mathcal{X} \to \mathcal{Y}$ that is efficient to train and easy to test. Assuming for each node $i \in \mathcal{Y}$ a vector $w_i \in \mathcal{R}^p$ has been given, a top-down procedure is utilized to classify an instance from the root to one leaf node [34, 36, 41, 42].

Algorithm 1: Top-Down Procedure for Classification

```
1 Inputs: ${\mathcal{C}(i)}_{i=1}^m, {w_i}_{i=1}^m, x_k$
2 Outputs: $i$
3 LET $i \leftarrow 0$
4 while $\mathcal{C}(i)$ is not empty do
5     LET $i = \arg\max_{j \in \mathcal{C}(i)} w_j^T x_k$
```

What Algorithm 1 is trying to do is to assign one leaf node to every unknown instance. Given any node i an instance x_k belongs to, Algorithm 1 classifies x_k among $\mathcal{S}^+(i)$, updating i and repeating the procedure until i is a leaf node. In this way we decouple the problem into a series of multi-class classification problems at each level.

There are different methods to train multi-class classifiers, either one-versus-rest, one-versus-one or solving a single optimization problem [38]. In this paper, we have adopted the one-against-rest approach, where for each node i, a binary classification problem is solved independently to learn a classifier that is responsible for predicting the membership of an instance against $\mathcal{S}^+(i)$. Let $\mathcal{T}(i) = \{k \mid (x_k, y_k) \text{ where } \mathcal{P}(i) \subset \mathcal{P}_{ath}(k)\}$ be the training index set for the classification problem at node i. Further

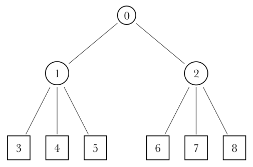


Figure 2.1: The hierarchy tree of the labels.

we define for each node $i \in \bar{\mathcal{Y}}$ and $k \in \mathcal{T}(i)$

$$y_k^{(i)} = \begin{cases} 1 & i \in \mathcal{P}_{ath}(k) \\ -1 & \text{otherwise} \end{cases}$$

then the training set for node i would be

$$\bar{\mathcal{X}}(i) = \{(x_k, y_k^{(i)}) \mid k \in \mathcal{T}(i)\} \qquad (2.3.2)$$

The procedures are schematized in Algorithm 2.

Algorithm 2: Independent Classifiers Training

```
1 Inputs: $\bar{\mathcal{Y}}, \{(x_k, y_k)\}_{k=1}^N$
2 Outputs: $\{w_i\}_{i=1}^m$
3 for $i = 0 \to m$ do
4     Construct training set $\bar{\mathcal{X}}(i)$
5     Training
6     Save $w_i$
```

2.3.2 A Novel Hybrid Approach

As motivated in Section 5.1, we want to build a hierarchical classification system that (1) provides scalability and portability, (2) imposes consistency in parent-child relations, (3) reduces inaccuracy in training classifier for each node, (4) adapts tailored models and algorithms to each node based on different conditions and requirements. While the first two points are addressed in Section 2.3 by introducing the top-down classifying procedure and independent training approach for each node, the last problem that is essential to the system's overall speed and accuracy is taken care of by our novel hybrid approach presented in this section. As mentioned in Section 2.3, in most hierarchical classification problems there are a large amount of data at the high levels of the class hierarchy, and the sizes the training sets are considerably smaller at the low levels as the classes become much more specific. The difference in problem sizes at nodes of different levels presents us with different challenges and emphases in solving the underlying optimization problem. In particular, for those nodes with large amount of data, a small generalization error of the model is inherently easy to obtain since more information is presented to the model. But the resulting optimization problem becomes very large and thus brings about computational difficulties. On the other hand, for those nodes with scarcity of data, it is relatively easy to solve the optimization problem but a more accurate solution is required to compensate for the lack of training information.

Regarding that, we present our novel hybrid approach, which combines the coordinate descent methods and the IPM algorithm in the hierarchical classification framework. Ideally we can use one algorithm from top to bottom if there exists such an algorithm that outperform others both in speed and accuracy. In fact this is what people have been using in practice most of the time, and Interior Point Methods (IPM) with its fast convergence rate makes such a good candidate, demonstrating satisfying performance as long as the size of the problem is small. However, the need to solve a Newton system markedly limits the scalability of IPM-based algorithm since the time and memory it takes to solve the Newton system at every iteration increases substantially as the problem size becomes large. Hence, we only use an IPM-based method as the problem becomes small enough, i.e. when we reach the bottom level of the hierarchy. For the top level of the hierarchy where the problem size is often much larger than that of bottom-level problems, in order to train the classifiers efficiently, we consider the two variants of coordinate descent methods.

Software Architecture The HiClass system is the combination and implementation of what we have discussed in previous sections, including the hierarchical classification framework and our novel hybrid approach with the implementation of IPM by [10], the first order coordinate descent by [9] called SCD and the second order variant by [32] called GLMNET, to solve the hierarchical classification problems accurately and efficiently. The strength of our HiClass system is that it is designed to be highly modular and flexible. The core of the system consists of four basic modules:

1. **Hierarchy Tree** The tree module implements the framework described in Section 2.3 and forms the foundation of the system.

2. **Data Reader** The reader module reads the data from text files of different formats (e.g. LIBSVM or UCI) and stores it in matrix and vector data structures in the memory.

3. **QP Writer** The writer module transforms the raw data into the appropriate matrices and vectors and writes them in the format required by different solvers as the input file.

4. **Solver Engine** The solver engine provides implementation for the algorithms described above.

Figure 2.2 shows the general architecture of the software system. Given the training data sets and the hierarchical structure of the labels, *HiClass_problem_read* transfers those data in memory to *HiTree_data*, which then distribute to every node a different training set (as defined in (2.3.2)) that are necessary for the solvers to compute the classifier of that node. *HiClass_node_write* interfaces with the data and the solvers that consists of GLMNET used in the top (first) level of the hierarchy along with IPM for the rest. Note that the process is easy to parallelize since every classifier for each node is trained independently, as illustrated in the figure.

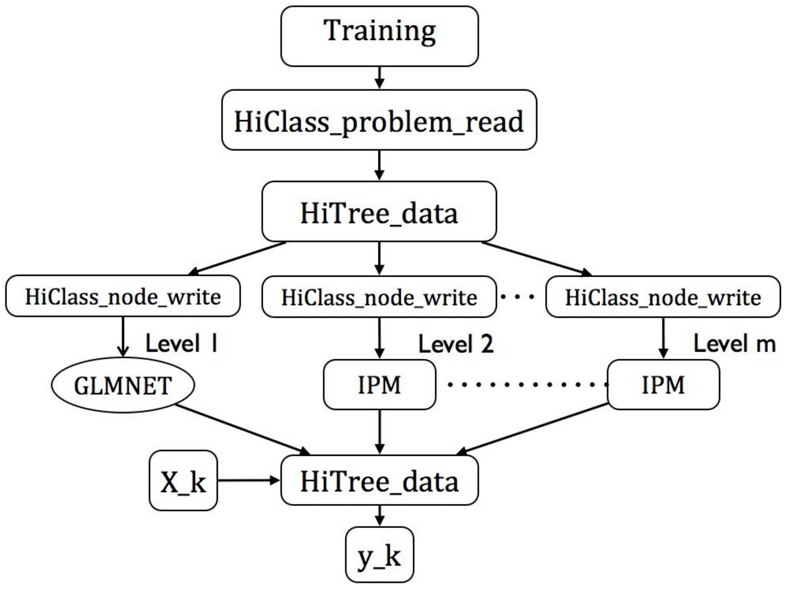


Figure 2.2: HiClass system flow chart

2.3.3 Experiments

In this section, we first discuss the measure of performance we use in every node of hierarchy and we test the performance of HiClass on real-world data set.

The Evaluation Metrics

To evaluate the performance of the HiClass system, we use the micro-average F_1 measure [43]. On a single category the F_β is given by

$$F_\beta = \frac{(\beta^2+1)A}{(\beta^2+1)A + B + \beta^2 C}, \tag{2.3.3}$$

where A is the number of documents a system correctly assigns to the category (true positives), B is the number of documents a system incorrectly assigns to the category (false positives), and C is the number of documents that belong to the category but which the system does not assign to the category (false negatives). Let $\beta = 1.0$, then we have the harmonic mean of recall and precision:

$$F_1 = \frac{2A}{2A+B+C} = \frac{2RP}{R+P}, \tag{2.3.4}$$

where R is recall, i.e., $A/(A+C)$, and P is precision, i.e., $A/(A+B)$.

In our case, every category i will have its own A_i, B_i and C_i. There are two methods, called macro-average and micro-average to combine those statistics into one measure in order to evaluate the general performance on the hierarchical structure. If we have category from 1 to l, then macro-average F_1 is just the unweighted mean of F-measure across all categories

$$macroF_1 = \frac{\sum_{i=1}^{l}(F_1)_i}{l} \tag{2.3.5}$$

and micro-average F_1 is given by

$$microF_1 = \frac{2\sum_{i=1}^{l} A_i}{\sum_{i=1}^{l} 2A_i + B_i + C_i} \tag{2.3.6}$$

In fact, (2.3.6) can be simplified. Since $\sum_{i=1}^{l} A_i + C_i = N$ where N is the number of all test instances. But we also have $\sum_{i=1}^{l} A_i + B_i = N$. Therefore,

$$\sum_{i=1}^{l} 2A_i + B_i + C_i = 2\sum_{i=1}^{l}(A_i + C_i) = 2N$$

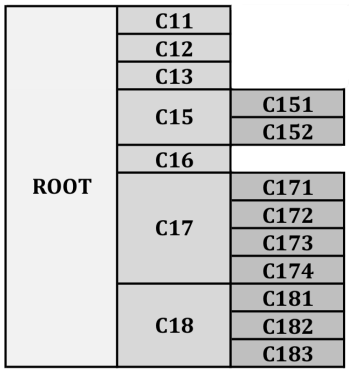
so (2.3.6) is in fact the ratio of true positives and total number of instances $microF_1 = \frac{\sum_{i=1}^{l} A_i}{N}$.

The RCV1 Dataset

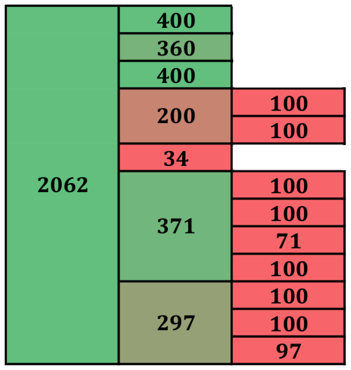
The RCV1-v2/LYRL2004 is a text categorization test collection made available from Reuters Corpus Volume I (RCV1), an archive of over 800,000 manually categorized newswire stories [43]. Each RCV1-v2 document used in our experiment has been tokenized, stopworded, stemmed and represented by 47,236 features in a final form as *ltc* weighted vectors. Those RCV1 documents are categorized with respect to the

Topics, which were organized in four hierarchical groups: CCAT, ECAT, GCAT and MCAT.

The RCV1-v2 documents are split chronologically into a training set and test set, with 23,149 training documents and 781,265 test documents. For our experiments, we focus on CCAT and MCAT two subtrees. The ECAT and GCAT subtrees are not considered because the former has a similar tree structure with CCAT and the latter display a structure that is flat rather than organized hierarchically. During preprocessing, we discard those that are not associated with any those leaf nodes as well as those that are categorized to multiple ones, requiring that each document used in the experiments is categorized to exactly one leaf node of either CCAT or MCAT. Next we discuss details.



(a) CCAT hierarchical structure. Each node in the tree is represented here as a block in the table. For example, C17 is the child of ROOT and has four children named from C171 to C174.



(b) The training data we use for classification at each node. The unbalanced data distribution problem we mentioned before is obvious here, with abundant amount of data for upper nodes yet much less at ones in the bottom.

Figure 2.3: CCAT subtree

Experiments on the CCAT Subtree We trim the CCAT *topic* group for our experiments and take out some "dumb" internal nodes, such as, those that have only one child. Reserving the main structure of CCAT, we end up with *topics* organized in a tree illustrated in Figure 2.3(a). There are altogether 17 nodes including 13 leaf nodes, 3 internal nodes and 1 root. The number of training data for each *topic* is illustrated in Figure 2.3(b) where the color shifts from green to red with more green denoting larger numbers. The total number is 2062 and most of them amass in the first hierarchy, which causes the previously mentioned unbalanced data distribution problem. In the experiments, we run including our HYBRID method all four algorithms with the same stopping criteria and same regularization parameter. The accuracy (using micro-average F_1 measure [43]) and cpu time are recorded for each node respectively and for the whole tree as well. The results are presented in Figure 2.4.

The first thing to note from Figure 2.4 is that the more second order information we take into consideration, the more accuracy we eventually achieve as expected, which can be seen by comparing the accuracy obtained by GLMNET, a second-order algorithm, with that of SCD which completely ignores the Hessian information. It can be seen that the first-order method SCD actually takes much longer than GLMNET to solve the problem, e.g., 556s compared with 17s. Having examined the detailed printouts of each algorithm, we find that even though SCD enjoys cheap per iteration cost, it takes SCD a lot more iterations than GLMNET to be able to satisfy the same stopping criteria, mostly due to the small step size SCD is forced to take each iteration. And the step size of SCD is inevitably small mainly because the lack of second order information forces it to build a globally upper bound model locally, which is thus a very loose bound with the minimizer too close to the current iterate. Hence, SCD progresses slowly, eventually resulting in expensive total computational cost even though each step only takes constant time.

On the other hand, GLMNET solves the problem very fast, e.g., nearly 35 times faster than SCD. The importance of curvature information can be further appreciated comparing the performance of GLMNET with that of SCD. Particularly, by incorporating second order information into the algorithm, GLMNET is able to achieve

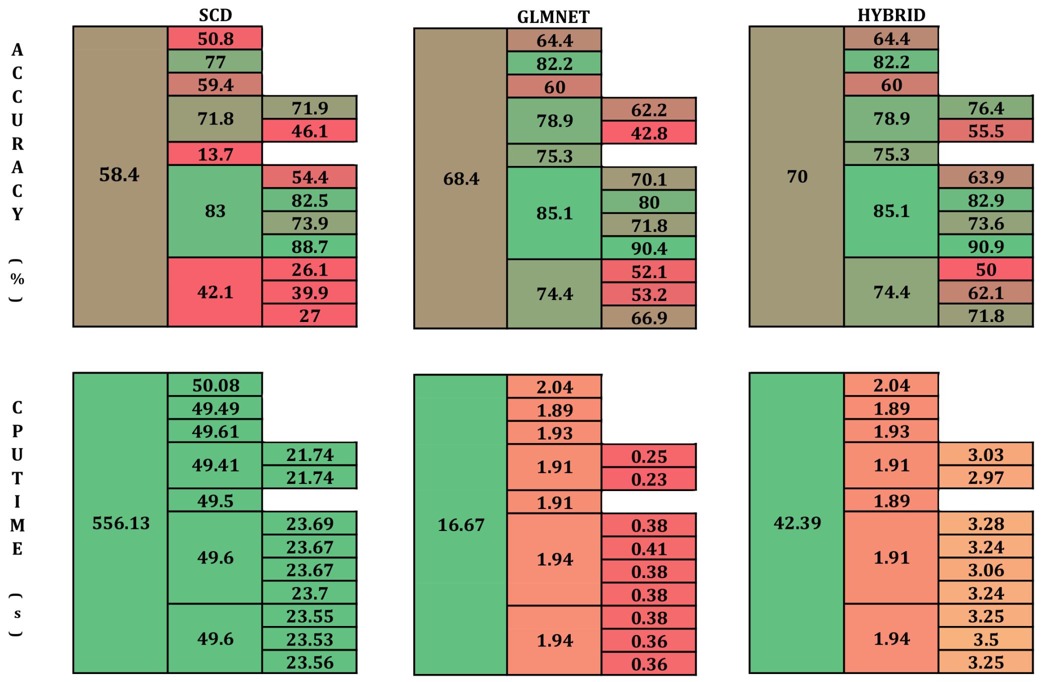
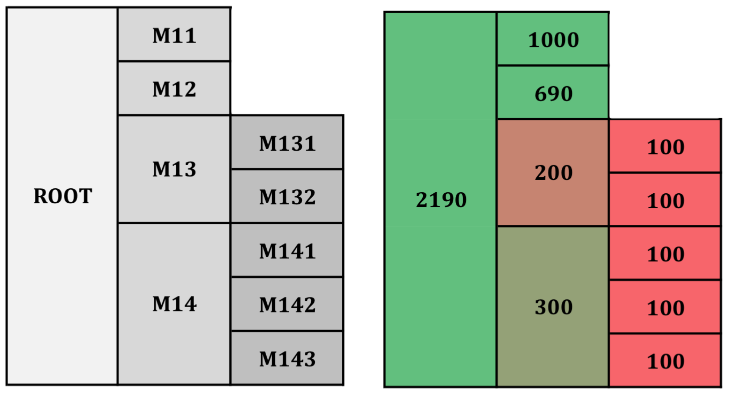


Figure 2.4: Performance comparison at each node between above mentioned algorithms. For HYBRID, we use GLMNET in the upper level and IPM at bottom. The number in the leftmost block of each table denotes the corresponding performance measure on the whole tree. For example, as shown by the rightmost upper table, 70% aggregate accuracy is achieved using the novel HYBRID method.

the fast convergence rate comparable to that of IPM; plus the cheap iteration cost similar to that of SCD, it is expected that GLMNET performs efficiently. However, GLMNET is not as accurate as IPM and although the inaccuracy can be somehow compensated by the abundance of training at the first hierarchy, the problem becomes more serious as it goes down the tree, e.g., almost 30% loss of accuracy in some cases.

Finally, by combining different algorithms we obtain a HYBRID method that has better accuracy than GLMNET. It is expected that the time HYBRID takes is over 90% less than that spent by SCD, due to not only the use of GLMNET in the first hierarchy, but also the fact that IPM at the bottom performs efficiently with limited amount of data as well. In fact, the saving can be much more when the data distribution is more severely unbalanced, in which case there are much more data in the first level and the number decreases dramatically at the bottom. As a result, the time IPM takes to solve the problem increases much faster than that of GLMNET, mainly due to IPM's per iteration complexity growing quadratically with the number of training points. The advantage of using HYBRID will become more obvious in those cases. Another thing to note is the accuracy HYBRID achieves, which is the highest of all. Again, as we discussed, the use of IPM at the bottom improves the accuracy of GLMNET and the use of GLMNET at the upper level returns a slightly better accuracy than IPM because the ability of GLMNET to efficiently process large scale of data.



(a) MCAT hierarchical structure. (b) The training data we use for classification at each node.

Figure 2.5: MCAT subtree

Experiments on the MCAT Subtree The whole experiment setting is similar to that of CCAT, except that this time we have a different hierarchical structure. The MCAT subtree contains altogether 9 nodes including 7 leaf nodes, 2 internal nodes and 1 root node, as illustrated in Figure 2.5(a). The number of training documents labelled under each *topics* is shown in Figure 2.5(b), from which we can see that over

75% of the data are accumulating on M11 and M12. We therefore expect that IPM will have a hard time at the first level where the data matrix has more than 2000 rows and 47326 columns. In fact, it does take IPM too long to solve the problem such that we can't include the results in Figure 2.6. But we are still able to use IPM in the HYBRID method because the problem size at the bottom is at most 300, which is comfortably within the range of IPM.

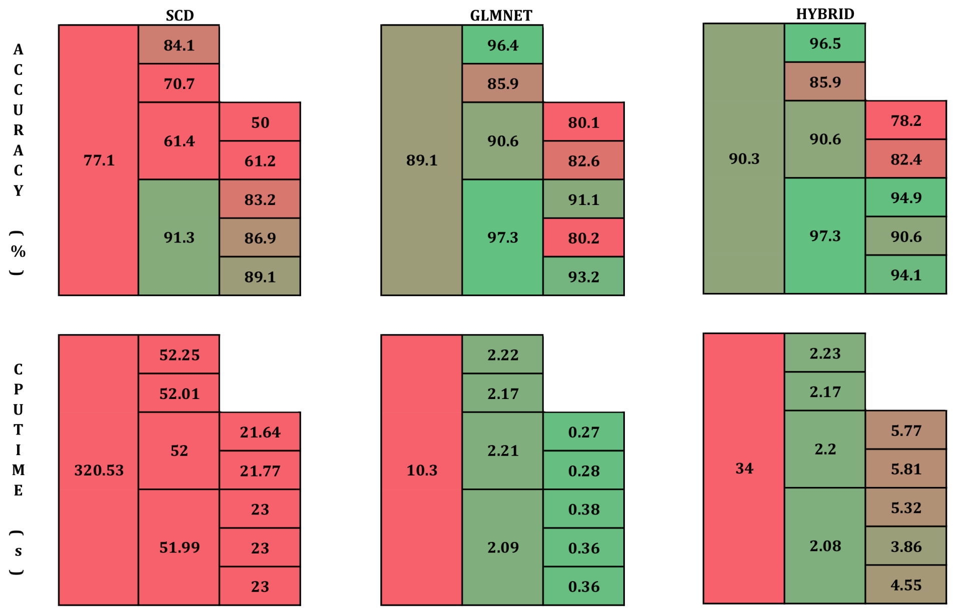


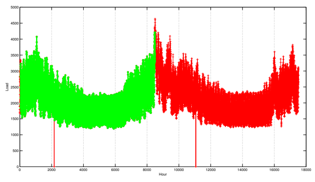
Figure 2.6: Performance comparison at each node between above mentioned algorithms.

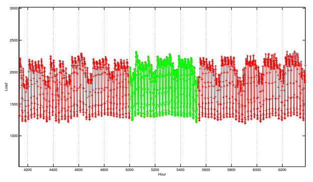
From Figure 2.6 we find that again, HYBRID outperforms other algorithm in terms of accuracy, while taking up only around one tenth of the time spent by SCD. GLMNET is still the fastest algorithm of the three, but as we discussed, lacks the high accuracy attained by most second order methods and using IPM in nodes M141, M142 and M143 helps improve the classification accuracy.

2.4 Distributed Learning for Short-term Forecasting

Short-term load forecasting (STLF), normally with a prediction time-horizon from 24 hours up to one-week ahead, has always been an important issue in efficient power system operation and decision making. STLF are not only needed for operation decisions in power system, such as scheduling of unit commitment and energy allocation, but also the key data for the electricity pricing as well. While an underestimation of the energy demand may result in inevitably high operational costs paid to procure energy in the market, an overestimation on the other hand wastes scarce resources. Hence, accurate and reliable load forecasts are vital for the energy transactions in competitive electricity markets. However, with the emergence of deregulation and free competition of modern energy markets [44, 45], the electric load is increasingly becoming difficult to forecast, thus giving rise to the need for more sophisticated forecasting tools with higher accuracy and reliability.

There are a wide variety of STLF methods proposed during the last decades, including linear or multiple regression model [46], autoregressive integrated moving average (ARIMA) [47], exponential smoothing method [48], feed-forward neural networks [49], etc. Generally speaking, all of the above mentioned methods can be categorized into two groups, the classical time series and regression methods including linear regression, ARIMA, exponential smoothing, etc., and the artificial intelligence methods containing for example artificial neural networks. While most of the classical methods are often relatively easy to implement and interpret [50], they are solely based on linear analysis, thus lacking the ability to capture the nonlinearity between the load series and the exogenous influencing factors, i.e., temperature, wind speed, etc. As a reality example illustrating the nonlinearity present in load series, we take the dataset of the North American electric utility[51, 52] and visualize them in Figure **??**. And it has been shown by many works in literature, e.g.,[45, 52–55], etc., that the forecasting performance greatly relies on how effectively the prediction algorithm captures this nonlinear nature of the time series, which is one of the main reasons why recently the so-called artificial intelligence methods have received more and more attention in solving the problem of loads forecasting. In particular, Support Vector Machine (SVM) or Support Vector Regression (SVR) [56], as a novel and powerful machine learning technique established on a theory of the structure risk minimization principle, has been known for its high generalization ability as well as resistant to the over-fitting problem, and several attempts have been made in recent literature to apply SVM to the field of load forecasting, e.g.,[45, 52, 53], etc. As the first well-known success of such attempts, authors in [53] embed different load-affecting factors into the feature space of SVR and from these features they formulate the predicted max load. They successfully detect in the beginning of their experiments a periodic component within the data set due to both the seasonal variation of consumer electricity demand, e.g., high demand in the winter while low in the summer, and the so-called "holiday" effect, e.g., demand variation between weekend and weekdays. Although they take advantage of those properties subsequently in their modeling, their method is rather preliminary – manually single out winter data, for example, to forecast loads in winter. Still, they report promising experiment results based on this idea of choosing training data by comparing the similarity between the forecasting points, which is further explored by later forecasting works. Notably, Authors in [45] propose a two-stage hybrid method combining an unsupervised network called the self-organizing map (SOM)[57] in the first stage, partitioning the non-stationary data into several subsets of stationary time-series data, with SVRs in the second stage, each defined on one of such subsets to take advantage of the similar dynamic properties. And authors in [52] introduce the locally weighted support vector regression (LWSVR), which, rather than taking in the entire data history, focuses the training on only the neighboring points around the current query point and endows a weight factor to each of the load datum. While most of those works are able to report in their experiments improvement in forecasting accuracy, several problems remain to be addressed, among which the biggest concern is the speed of the algorithm. Applying SVM to time series forecast often requires solving a constrained quadratic problem in large scale. While this alone is computationally prohibitive, combining it with the SOM in the hybrid method [45] will only make the computational complexity even worse. Similarly, for LWSVR in [52], the computation of the neighborhood before training SVM requires a traverse through the whole training set. Since that has to be done for each forecasting time point, the complexity of this preprocessing alone will be related to the product between the size of the training set and that of the forecasting set, which is impractical in most reality uses.


(a) Yearly View. One year of data is singled out by green.


(b) Weekly View. Three weeks of data is singled out by green.

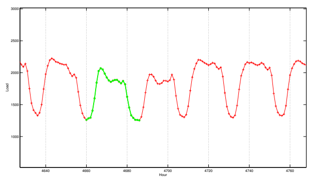

(c) Daily View. 24 hours of data is singled out by green.

Figure 2.7: Illustration of Load Series Periodicity.

In this section, we develop a time series forecasting algorithm that is specifically optimized for such large-scale forecasting problems, while at the same time incorporating this similarity-based modeling idea to achieve high accuracy in nonlinear and non-stationary time series prediction. In particular, we propose a two-phase forecasting framework that includes the training phase where forecasting models are distributively constructed, and the forecasting phase where the final forecasts are computed jointly from the outputs of all the models combined. Since the construction of forecasting models is the most computationally intensive part of the algorithm, the encapsulation of it into a separate phase makes it possible to reuse the models for different forecasting tasks. Another novelty of our algorithm is that the whole framework can be easily parallelized thanks to its distributed nature – we decouple the whole forecasting problem into several smaller separable ones at beginning of the algorithm and construct a forecasting model for each of those decoupled problems. Since each model is self-contained and trained on problem much smaller in size than the original one, the construction process will be much more efficient and can be easily parallelized, which significantly accelerate the whole training process. Finally, in the forecasting phase, the forecasts from all the models are taken into consideration simultaneously to form the final forecasting output, which we cast into a multi-class classification problem defined for each forecasting point such that the algorithm adaptively and intelligently determines the way to form the final forecast from outputs of each model.

2.4.1 Problem Setting

Let us suppose that we have an univariate time series $z(t)$ for $t = 1, 2, ..., l$, where $z(t)$ is the electric load measurement at time step t. Our goal here is to predict the value of $z(t)$ at the next time step $t = l + 1$. Let f_i denotes the ith forecasting model for $i = 1, ..., k$. Let $\mathcal{X}_i \subset \mathbb{R}^d$ be the instance domain and let $\mathcal{Y}_i$ be the set of labels. $\mathcal{S}_i = \mathcal{X}_i \times \mathcal{Y}_i$ is the training data for the forecasting model f_i. That is, for each point $(x_t, y_t) \in \mathcal{S}_i$, we have

$$y_t = f_i(x_t) \tag{2.4.1}$$

Here a standard approach to determine (x_t, y_t) is to follow Takens embedding theorem[58], which states that given any time series $z(t)$, under certain conditions for almost all t and for some finite d, m, there is a smooth map $f : \mathbb{R}^d \to \mathbb{R}$ such that

$$z(t) = f(z(t-m), z(t-2m), ..., z(t-dm)), \quad \forall t > dm \tag{2.4.2}$$

where the value of d is called the embedding dimension and the smallest value for which (2.4.2) is true is called the minimum embedding dimension. Hence, according to the above theorem, the x_t, y_t pair is given by $y_t = z(t)$ and $x_t = [z(t-m), z(t-2m), ..., z(t-dm)]'$. Note that d, m are the parameters specified by users and considered as constants in the algorithm. So once the time series $z(t), t = 1, ..., l$ is given and index t is specified, the x_t, y_t pair is uniquely determined. Hence, for the sake of clarity, we redefine the training set $\mathcal{S}_i$ of the forecasting model f_i as a index set such that $\mathcal{S}_i \subset \mathcal{N} = \{1, ..., l\}$, and for any $t \in \mathcal{S}_i$ we have $y_t = f_i(x_t)$, where $y_t = z(t)$ and $x_t = [z(t-m), z(t-2m), ..., z(t-dm)]'$.

The training phase and forecasting phase framework are described in Algorithm 3 and Algorithm 4, respectively. There are three key steps in the two frameworks, which are Step 2 data distribution and Step 4 model training in Algorithm 3, and Step 3 probability estimation in Algorithm 4. And we will discuss each of them in details next.

Algorithm 3: Distributed Training Framework

```
1 Inputs: time series $z(t), t \in \mathcal{N}$ and $k$.
2 Determine the data distribution $\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_k$ such that $\mathcal{S}_1 \cup \mathcal{S}_2 \cup ... \cup \mathcal{S}_k = \mathcal{N}$.
3 for each $i \in \{1, ..., k\}$ do
4     Train forecasting model $f_i$ on $\mathcal{S}_i$.
5 Return: forecasting models $f_i$ for $i = 1, ..., k$.
```

Algorithm 4: Joint Forecasting Framework

```
1 Inputs: forecasting models $f_i$ for $i = 1, ..., k$ and query point $x_q$
2 for $i \in \{1, ..., k\}$ do
3     Estimate probability $p_i$
4     Compute $\hat{y}_i = f_i(x_q)$.
5 Return: forecasting value $y_q = \sum_{i=1}^{k} p_i \hat{y}_i$.
```

2.4.2 Description of the Algorithm

k-Means in $\mathcal{F}$ Space

The k-means is a clustering method based on minimization of the total distance between all inputs from their respective cluster centers while at the same time maximizing the between-cluster-object dissimilarity. Here we extend the method into the high-dimensional $\mathcal{F}$ space so that it can be used to cluster data mapped to different affine segments in $\mathcal{F}$. The difficulty is that in most cases $\mathcal{F}$ space is only implicitly defined via kernels, because given any x_i in current space it is computationally infeasible to write out explicitly the expression of its counterpart $\Phi(x)$ in $\mathcal{F}$, due to both polynomial features of higher order and higher dimensionality. However, as we shall see later, the k-means algorithm can be modified so that it only depends on dot products between x_i. Hence, it suffices to know $k(x_i, x_j) := \Phi(x_i)^T\Phi(x_j)$ rather than Φ explicitly. Firstly, let us suppose we are given a set of points from original (input) d-dimensional space, $\{x_i\}_{i=1}^N \in \mathbb{R}^d$. The problem is to partition the N points' $\mathcal{F}$-space counterparts $\{\Phi(x_i)\}_{i=1}^N \in \mathcal{F}$ into k sets $(k \leq N)$ $\mathcal{X} = \{\mathcal{X}_1, \mathcal{X}_2, ..., \mathcal{X}_k\}$ such that the within-cluster sum of squares is minimized:

$$\mu^* := \arg\min_{\mu} \sum_{i=1}^{k} \sum_{\Phi(x)\in\mathcal{X}_i} ||\Phi(x) - \Phi(\mu_i)||^2 \tag{2.4.3}$$

where $\mu^* \in \mathbb{R}^{d\times k}$ with μ_i^* being the ith column in matrix μ^*; $\Phi(\mu_i) \in \mathcal{F}$ is the *centroid* of $\mathcal{X}_i$, and can be obtained by

$$\Phi(\mu_i) = \frac{\sum_{\Phi(x)\in\mathcal{X}_i} \Phi(x)}{|\mathcal{X}_i|} \tag{2.4.4}$$

Note that $\Phi(\mu_i)$ can be considered as the "best" representative of the cluster $\mathcal{X}_i$ since

$$\Phi(\mu_i) = \arg\min_{z} \sum_{\Phi(x)\in\mathcal{X}_i} ||\Phi(x) - z||^2 \tag{2.4.5}$$

Although we do not have an explicit expression of the centroid $\Phi(\mu_i)$, we can still compute the Euclidean distance between any point $\Phi(\bar{x})$ and the *centroid* $\Phi(\mu_i)$ through the kernel if $\Phi(\mu_i)$ is given by (2.4.4):

$$\begin{aligned} ||\Phi(\bar{x}) - \Phi(\mu_i)||^2 &= (\Phi(\bar{x}) - \Phi(\mu_i))^T(\Phi(\bar{x}) - \Phi(\mu_i)) \\ &= \Phi(\bar{x})^T\Phi(\bar{x}) - 2\Phi(\bar{x})^T\Phi(\mu_i) + \Phi(\mu_i)^T\Phi(\mu_i) \\ &= k(\bar{x}, \bar{x}) - 2\frac{\sum_{x\in\mathcal{X}_i} k(\bar{x}, x)}{|\mathcal{X}_i|} + \frac{\sum_{x^{(1)}\in\mathcal{X}_i}\sum_{x^{(2)}\in\mathcal{X}_i} k(x^{(1)}, x^{(2)})}{|\mathcal{X}_i|^2} \end{aligned} \tag{2.4.6}$$

or if μ_i is known:

$$\begin{aligned} ||\Phi(\bar{x}) - \Phi(\mu_i)||^2 &= \Phi(\bar{x})^T\Phi(\bar{x}) - 2\Phi(\bar{x})^T\Phi(\mu_i) + \Phi(\mu_i)^T\Phi(\mu_i) \\ &= k(\bar{x}, \bar{x}) - 2k(\bar{x}, \mu_i) + k(\mu_i, \bar{x}) \end{aligned} \tag{2.4.7}$$

The procedures of implementing k-Means in $\mathcal{F}$ Space are described in Algorithm 5. The structure of the algorithm is very simple, consisting mainly a while-loop to examine the termination conditions and two types of for-loops to compute the clusters $\mathcal{X}$, depending on which formula to use for calculating the distance between the points and the *centroids*.

Let us now analyze the per iteration cost of Algorithm 5. The main computational expense here is the for-loop to update the cluster $\mathcal{X}_i$ for each $i \in \{1, ..., k\}$. That is, for each one of the N training points x, we compute and compare its distance to the k *centroids* by (2.4.6), choosing the nearest *centroids*, say the ith one, and putting x in the corresponding cluster $\mathcal{X}_i$. So it is k times the per evaluation cost of (2.4.6) for each training point x. Suppose that we have pre-evaluated and stored the kernel matrix $K \in \mathbb{R}^{N\times N}$ with $k_{ij} = k(x_i, x_j)$. Then computing the first term in (2.4.6) takes $O(k)$ flops and the second term takes $O(\sum_i |\mathcal{X}_i|) = O(N)$ flops. For N points together, it takes $O(N^2)$ flops. Note that this estimate only takes into account the first two terms in (2.4.6). We consider the last term separately because it is fixed for each cluster $\mathcal{X}_i$, so in each iteration it is computed once, which takes $O(kN^2)$ flops for k clusters, and cached. Altogether, the per iteration complexity of Algorithm 5 is $O(kN^2)$.

Algorithm 5: k-Means in $\mathcal{F}$ Space

```
1 Set $t \leftarrow 0$.
2 Initialize the $k$ *centroids* of clusters: $\mu_1^{(t)}, \mu_2^{(t)}, ..., \mu_k^{(t)}$.
3 for $i \in \{1, ..., k\}$ do
4     Compute $\mathcal{X}_i := \{j \mid j \in \{1, ..., N\}$ and $i$ is equal to $i^*$ with $i^* :=$
      $\arg\min_{ii \in \{1,...,k\}} ||\Phi(\mu_{ii}^{(t)}) - \Phi(x_j)||^2\}$ using (2.4.7).
5 while optimality test returns false do
6     for $i \in \{1, ..., k\}$ do
7         Update $\mathcal{X}_i := \{j \mid j \in \{1, ..., N\}$ and $i$ is equal to $i^*$ with $i^* :=$
          $\arg\min_{ii \in \{1,...,k\}} ||\Phi(\mu_{ii}^{(t)}) - \Phi(x_j)||^2\}$ using (2.4.6).
8     Set $t \leftarrow t + 1$.
9 Return $\mathcal{X}$.
```

An Accelerated Kernel k-Means Algorithm Here we propose an algorithm to efficiently solve the kernel kMeans problem (2.4.3) in large scale. While the standard iterative solution to k-Means (Algorithm 5) suffers from poor scalability and is easily trapped in local minima due to the greedy nature of the update, our proposed algorithm, built on the recent advances at the interface of machine learning and optimization research, is not only designed to improve the clustering accuracy, but also specifically tuned to large-scale clustering problem. A key step in achieving such superior performance lies in the equivalent transformation of the optimization of problem (2.4.3) into a Nonnegative Matrix Factorization (NMF) problem, which can be seen, from an optimization point of view, as a nonlinear programming with a nonconvex objective and bound constraints:

$$\min_{G \geq 0} \quad ||K - GG^T||^2 \tag{2.4.8}$$

where the constraints are component-wise and the matrix norm $||G||^2 = \sum_{i,j} G_{ij}^2$ is the Frobeneus norm. To see the two problems (2.4.3) and (2.4.8) are actually equivalent, let us first define the matrix $G \in \mathbb{R}^{N\times k}$ such that

$$G_{ij} = \begin{cases} 1, & \text{if } x_i \in \mathcal{X}_j, \\ 0, & \text{otherwise.} \end{cases} \tag{2.4.9}$$

If we consider strict k-Means where every point x_i is assigned to exactly one cluster, then G is a cluster membership indicator matrix with each row containing none but one non-zero component, which can be enforced by the following two constraints

$$G \geq 0 \quad \text{and} \quad G^T G = I \tag{2.4.10}$$

Using G, we can write down the objective of k-Means (2.4.3) alternatively as

$$\min_{G\geq 0, G^TG=I} \quad \sum_{i=1}^{k}\sum_{j=1}^{N} G_{ji}||x_j - \mu_i||^2 = \sum_{i=1}^{k}\sum_{j=1}^{N} G_{ji}||x_j - \frac{\sum_{j=1}^{N} G_{ji}x_j}{\sum_{j=1}^{N} G_{ji}}||^2 \tag{2.4.11}$$

which has a compact form matrix representation

$$\min_{G\geq 0, G^TG=I} \quad \operatorname{tr}(X^TX) - \operatorname{tr}(G^TX^TXG) = -\operatorname{tr}(G^TKG) \tag{2.4.12}$$

where $K = X^TX$ is the linear kernel matrix. In nonlinear kernel mapping, k_{ij} is simply given by the inner product of $\Phi(x_i)$ and $\Phi(x_j)$. Since multiplying by a positive number or adding a constant to the objective does not alter the optimal solution, (2.4.12) can be further transformed into the following problem

$$\min_{G\geq 0, G^TG=I} ||K||^2 - 2\operatorname{tr}(G^TKG) + ||G^TG||^2 = ||K - GG^T||^2 \tag{2.4.13}$$

Relaxing the orthogonality $G^TG = I$ turns it into the NMF problem (2.4.8) presented in the beginning, and it has been shown that without the constraint the solution to the problem (2.4.8) still retains G orthogonality approximately[59].

The real challenge, however, lies in how to solve (2.4.8) for the cluster membership indicator matrix G because any such algorithm need to at least address three difficulties: 1.) retain sparsity in G for clustering interpretability. 2.) deal with the non-convexity and non-linearity in the objective to avoid trapped in local minima.

3.) design for large-scale clustering problems. There are mainly three novelties in our proposed algorithm to deal with those difficulties respectively. First of all, we transform the nonconvex problem (2.4.8) into a convex one by both adding the equality constraint $X = G$ and modifying the objective function to $||K - XG^T||^2$, so the problem now becomes bi-affine

$$\min_{G \geq 0, X = G} \quad ||K - XG^T||^2 \tag{2.4.14}$$

To deal with the equality constraint, we introduce the augmented Lagrangian function:

$$\mathcal{L}(X, G, V, \mu) = ||K - XG^T||^2 - \mathrm{tr}(V^T(X - G)) + \frac{\mu}{2}||X - G||^2 \tag{2.4.15}$$

where $V \in \mathbb{R}^{N \times k}$ is the matrix of untrstricted Lagrange multipliers for the equality constraint $X = G$ and the parameter $\mu \in \mathbb{R}$ is positive. Note that the augmented Lagrangian function can be seen as both the extension of the quadratic penalty function by the addition of Lagrangian multipliers to reduce the possibility of ill conditioning, and the extension of the standard Lagrangian by the addition of the penalty term involving μ to measure the infeasibility of the problem (2.4.14). The motivation for using the augmented Lagrangian algorithm is that, for an appropriate fixed choice (V^*, μ^*), an optimal solution (X^*, G^*) can be obtained by solving the following problem:

$$\min_{G \geq 0, X} \quad \mathcal{L}(X, G, V^*, \mu^*) \tag{2.4.16}$$

which is a bi-convex minimization problem, i.e., convex in X for each G and convex in G for each X. This motivates the alternating direction method which splits the nonconvex problem into the following two convex subproblems by alternatingly updating X and G:

$$X^{n+1} := \arg\min_X \mathcal{L}(X, G^n, V^n) \tag{2.4.17}$$

$$G^{n+1} := \arg\min_{G \geq 0} \mathcal{L}(X^{n+1}, G, V^n) \tag{2.4.18}$$

and we update the Lagrange multipliers by $V^{n+1} := V^n - \mu(X^{n+1} - G^{n+1})$. Minimizing with respect to X (2.4.17) is easy with a closed-form solution obtained by setting the gradient of $\mathcal{L}$ with respect to X to zero

$$\nabla_X \mathcal{L}(X, G^n, V^n) = -2KG^n + 2X^{n+1}(G^n)^T G^n - V^n + \mu(X^{n+1} - G^n) = 0 \tag{2.4.19}$$

$$\Rightarrow \quad X^{n+1} = (2KG^n + V^n + \mu G^n)(2(G^n)^T G^n + \mu I)^{-1} \tag{2.4.20}$$

The complexity, however, is mainly in obtaining G^{n+1} by (2.4.18), a bound-constrained nonlinear problem. A standard approach is using bound-constrained trust region method to compute the direction each iteration by solving a quadratic model of $\mathcal{L}_G$. But this would require manipulating the Hessian matrix that has as many as $O(N^2k^2)$ entries, the storage of which alone is going to be computationally prohibitive in some cases. So here we consider an accelerated gradient projection descent method (Algorithm 6), which takes advantage of a Nesterov-type extrapolation technique (step 6) to accelerate the minimization of subproblem to reach the theoretically best convergence rate of first-order method. Due to both the acceleration and the Hessian-free framework, the Algorithm 6 is designed to be scalable and can be called efficiently as a subroutine. Another important property of the Algorithm 6 is that it induces sparsity in G rather aggressively thanks to the projected gradient step 7, where the projection P is defined as follows

$$P_{ij}(X, a) = \begin{cases} x_{ij}, & \text{if } x_{ij} > a, \\ a, & \text{otherwise.} \end{cases} \tag{2.4.21}$$

Let us now analyze the optimality condition of problem (2.4.14). We first define the standard Lagrangian function

$$\mathcal{L}(X, G, V_{eq}, V_{ieq}) = ||K - XG^T||^2 - \mathrm{tr}(V_{eq}^T(X - G)) - \mathrm{tr}(V_{ieq}^T G) \tag{2.4.22}$$

where $V_{eq} \in \mathbb{R}^{N\times k}$ and $V_{ieq} \in \mathbb{R}^{N\times k}$ are the Lagrange multipliers for equality constraints $X = G$ and inequality constraints $G \geq 0$ respectively. Then the first-order

Algorithm 6: Accelerated Gradient Projection Descent Method for Subproblem (2.4.18)

```
1 INPUT: $X, V, L_G$ and $G$
2 INITIALIZATION: Set $i \leftarrow 0, G^{i-1} \leftarrow G, G^i \leftarrow G, t_{i-1} \leftarrow 1$
3 while optimality test returns false do
4     $t_i \leftarrow (1 + \sqrt{1 + 4t_{i-1}^2})/2$
5     $\omega_i \leftarrow (t_{i-1} - 1)/t_i$
6     $Z^i \leftarrow G^i + \omega_i(G^i - G^{i-1})$
7     $G^{i+1} \leftarrow P(Z^i - \nabla_G \mathcal{L}(X, Z^i, V)/L_G, 0)$
8     $i \leftarrow i + 1$
9 RETURN $G^i$.
```

optimality condition is given by

$$\nabla_{X^*}\mathcal{L}(X^*, G^*, V_{eq}^*, V_{ieq}^*) = 0 \tag{2.4.23}$$

$$\nabla_{G^*}\mathcal{L}(X^*, G^*, V_{eq}^*, V_{ieq}^*) = 0 \tag{2.4.24}$$

$$G^* \geq 0 \tag{2.4.25}$$

$$V_{ieq}^* \geq 0 \tag{2.4.26}$$

$$V_{ieq}^* \cdot G^* = 0 \tag{2.4.27}$$

which yields the optimality test

$$\nabla_X \mathcal{L} = -2KG + 2XG^TG - V_{eq} \leq \epsilon \tag{2.4.28}$$

we use to terminate and return the solution G in Algorithm 7, the final algorithmic frame for solving the kernel kMeans problem (2.4.8) .

Customized Primal SVR via Kernels

In this section we describe the algorithm to train linear regression in high-dimensional $\mathcal{F}$ space for each affine cluster $\mathcal{X}_i$, where $\mathcal{F}$ space is defined implicitly through the use of kernels. In particular, we discuss our customizations in standard Support Vector Regression (SVR) in order to learn a forecasting curve that gives a tight

Algorithm 7: Accelerated Alternating Projection Method for Kernel kMeans (2.4.8)

```
1 INPUT: kernel matrix $K$ and number of clusters $k$
2 INITIALIZATION: Set the iteration counter $n \leftarrow 1$, penalty parameter $\mu \leftarrow 10$
  and the variables $G^n, X^n, V^n \in \mathbb{R}^{N\times k}, G^n \leftarrow \mathbf{0}, X^n \leftarrow \mathbf{0}, V^n \leftarrow \mathbf{1}$,
3 while optimality test (2.4.28) returns false do
4     Compute $X^{n+1} = (2KG^n + V^n + \mu G^n)(2(G^n)^T G^n + \mu I)^{-1}$.
5     Compute $G^{n+1}$ by Algorithm 6.
6     Compute $V^{n+1} = V^n - \mu(X^{n+1} - G^{n+1})$
7     $n \leftarrow n + 1$
8 RETURN $G^n$.
```

upper bound prediction of the future load value. To solve the resulted, often large-scale, nonlinear optimization problem, we deploy online learning algorithms to ensure efficiency and scalability, which however requires working directly with the primal SVR. While kernelized SVR is often considered in its dual format, we extend the online learning algorithm to kernelized primal SVR.

To facilitate the discussion, we will suppose we have only one affine cluster $\mathcal{X}$ containing a training set $\{(x_i, y_i)\}_{i=1}^N \subset \mathcal{X} \times \mathbb{R}$. Our goal here is to learn a function f as defined in (2.4.1) by minimizing the empirical loss $L(w)$ on $\{(x_i, y_i)\}_{i=1}^N$ plus an additional regularization term $||w||^2$ to avoid overly complex hypotheses:

$$\min_{w} \quad F(w) = L(w) + \frac{\lambda}{2}||w||^2 \tag{2.4.29}$$

There are variety of loss functions $L(w)$ to choose from for regression, e.g., squared loss, Huber's robust loss, Vapnik's ϵ-insensitive loss, etc. Particularly, Vapnik's ϵ-insensitive loss function is given by $L(w) = \frac{1}{N}\sum_{i=1}^N \max(0, |y_i - f(x_i)| - \epsilon)$, which establishes a ϵ tube around the data such that no points but those outside of the tube will suffer a loss. The placement of the tube is iteratively optimized in a way that maximizes the number of points inside of the tube. Normally, this ϵ tube is always placed in symmetric with respect to the regression function f, which allows training data to deviate from f in all direction. But in certain applications like load forecasting, we would rather like the forecasting function to give a slightly overestimated prediction than a underestimated one in order to prevent a potential power outage. To that end, we modify the ϵ-insensitive loss function such that

$$L(w) = \frac{1}{N}\sum_{i=1}^{N} \max(0, |y_i - f(x_i)| - \epsilon + \operatorname{sgn}(y_i - f(x_i))\eta), \quad 0 < \eta < \epsilon \tag{2.4.30}$$

which shifts the ϵ tube upward by the amount of η that is the so-called *shifting* parameter chosen by the users. To make it more clearly the purpose of η, we can formulate the above equation alternatively in the following way

$$L(w) = \frac{1}{N}\sum_{i=1}^{N} l_i(w), \quad \text{with} \tag{2.4.31}$$

$$l_i(w) = \begin{cases} \max(0, |y_i - f(x_i)| - (\epsilon - \eta)), & \text{if } y_i - f(x_i) \geq 0 \\ \max(0, |y_i - f(x_i)| - (\epsilon + \eta)), & \text{if } y_i - f(x_i) < 0 \end{cases} \tag{2.4.32}$$

Note that when the prediction $f(x_i)$ is below the actual value y_i we suffer a loss as soon as the deviation grows larger than $\epsilon - \eta$, whereas in the case where $f(x_i)$ is over y_i, we allow the deviation to be as large as $\epsilon + \eta$ before we begin to penalize. In this way, we basically give the algorithm more space to choose f over the actual load curve, while reducing the freedom of f going under the curve.

Now let us consider the algorithm to solve (2.4.29) with the shifted loss (2.4.30). It has always been a challenge to solve such problem efficiently in large scale because of both the nonlinearity and non-smoothness present in the objective function. Here we consider the projected stochastic gradient descent methods, which enjoys inexpensive per iteration cost while still managing a superior rate of convergence among other first-order methods. We extend it, for the purpose of this paper, to SVR in kernel space. The algorithm mainly contains two steps, an update step and a projection step. In the update step, a stochastic gradient descent is performed such that

$$w_{t+\frac{1}{2}} = w_t - \alpha_t \partial F_{\mathcal{A}_t}(w_t) \tag{2.4.33}$$

where $\alpha_t = \frac{1}{\lambda t}$ is the learning rate and $-\partial F_{\mathcal{A}_t}(w_t) = \frac{1}{|\mathcal{A}_t|}\sum_{i\in\mathcal{A}_t}\Phi(x_i)\,\mathrm{sgn}(y_i - f(x_i)) - \lambda w_t$ is the negative gradient of the sub-sampled objective function which gives a descent direction in the objective; $\mathcal{A}_t$ is the set of points for which w_t suffers a non-zero loss. In the projection step, $w_{t+\frac{1}{2}}$ is projected onto the set $\mathcal{B} = \{w : ||w|| \leq 1/\sqrt{\lambda}\}$ such that

$$w_{t+1} = \min(1, \frac{1/\sqrt{\lambda}}{||w_{t+\frac{1}{2}}||})w_{t+\frac{1}{2}} \tag{2.4.34}$$

which, as analysis shows, allows the usage of a very aggressive decrease in the learning rate and yields an improved rate of convergence[60]. As simple as the algorithm seems to be, the extension to kernelized SVR is not quite straightforward. The main difficulty here is that w lies in the high dimensional space rather than the input space (recall (2.4.1)), and directly manipulating w in that space is usually intractable. We circumvent that difficulty by expressing w with dual variables a of (2.4.29) as following

$$w_t = \sum_{i=1}^{N} a_i^{(t)}\Phi(x_i) \tag{2.4.35}$$

Substituting it into (2.4.1), we obtain the formula expressing f in terms of kernel $k()$

$$f(\bar{x}) = \sum_{i=1}^{N} a_i^{(t)} k(x_i, \bar{x}) + b \tag{2.4.36}$$

Substituting (2.4.35) into (2.4.33), we obtain

$$\sum_{i=1}^{N} a_i^{(t+\frac{1}{2})}\Phi(x_i) = \sum_{i\in\mathcal{A}_t}[(1-\alpha_t\lambda)a_i^{(t)} + \frac{\alpha_t}{|\mathcal{A}_t|}\,\mathrm{sgn}(y_i - f(x_i)]\Phi(x_i) + \sum_{i\in\bar{\mathcal{A}}_t}(1-\alpha_t\lambda)a_i^{(t)}\Phi(x_i) \tag{2.4.37}$$

which yields the rules to update the dual variables

$$a_i^{(t+\frac{1}{2})} \leftarrow (1-\alpha_t\lambda)a_i^{(t)} + \frac{\alpha_t}{|\mathcal{A}_t|}\,\mathrm{sgn}(y_i - f(x_i), \quad \text{for } i \in \mathcal{A}_t \tag{2.4.38}$$

$$a_i^{(t+\frac{1}{2})} \leftarrow (1-\alpha_t\lambda)a_i^{(t)}, \quad \text{for } i \in \bar{\mathcal{A}}_t \tag{2.4.39}$$

where $f(x_i)$ can be computed by (2.4.36) and $\bar{\mathcal{A}}_t$ is the complementary set of $\mathcal{A}_t$ with respect to the index set $\mathcal{N} = \{1, 2, ..., N\}$. Similarly, we can perform the projection step by updating a instead of w. Using $||w_t||^2 = ||\sum_{i=1}^{N} a_i^{(t)} \Phi(x_i)||^2 = \sum_{i,j\in\mathcal{N}} a_i^{(t)} a_j^{(t)} k(x_i, x_j)$, we obtain

$$a_i^{(t+1)} \leftarrow a_i^{(t+\frac{1}{2})} \min(1, \frac{1/\sqrt{\lambda}}{\sqrt{\sum_{i,j\in\mathcal{N}} a_i^{(t+\frac{1}{2})} a_j^{(t+\frac{1}{2})} k(x_i, x_j)}}), \quad \text{for } i \in \mathcal{N} \tag{2.4.40}$$

Algorithm 8: online learning for kernelized primal SVR

```
1 INPUT: $T, \mathcal{N}, \lambda, s \leq |\mathcal{N}|, \{(x_i, y_i)\}_{i\in\mathcal{N}}$
2 Set $t \leftarrow 0$.
3 Initialize the dual variables $a^{(t)}$ to zeros.
4 for  $t$ = 1,2,...,$T$ do
5     Compute $\mathcal{A}_t$ such that $\mathcal{A}_t := \{i : l_i(w) \neq 0 \text{ and } |\mathcal{A}_t| = s\}$.
6     Update the dual variables by (2.4.38) and (2.4.39).
7     Project the dual variables by (2.4.40).
8 Return $a^{(t)}$.
```

Joint Forecasting via Kernel SVM

If only one regression function f is trained to model the data distribution, then the forecasted time series value $f(x)$ is uniquely defined given any query point x. But it becomes a little complicated when there are multiple regression functions $f_{\mathcal{X}_i}$ learned for each cluster $\mathcal{X}_i, i = 1, ..., k$, because now we have multiple forecasted values $f_{\mathcal{X}_1}(x), ..., f_{\mathcal{X}_k}(x)$ all at the same point x and we need to know which one (or ones) to use. Mathematically, this problem can be formulated as one finding a unit-norm probability vector

$$p(x) = [p_1(x), ..., p_k(x)]^T \text{ with } |p(x)| = 1, p_i(x) \geq 0 \ \forall i = 1, ..., k \tag{2.4.41}$$

such that the linear combination of those forecasted values $\sum_{i=1}^{k} p_i(x) f_{\mathcal{X}_i}(x)$ predicts better than any one of those value alone does. Here the critical step is to determine the probability vector $p(x)$, where each component $p_i(x)$ is the indicator of the prediction accuracy of $f_{\mathcal{X}_i}(x)$. In the extreme case, for example, $p_i(x) = 1$ implies that $f_{\mathcal{X}_i}(x)$ predicts 100 percent correct, and $p_i(x) = 0$ indicates that $f_{\mathcal{X}_i}(x)$ is irrelevant here and better not be included in the forecasting output. To determine the probability vector $p(x)$, let us consider the kernel space where embedded time series data distribute in different clusters $\mathcal{X}_i, i = 1, ..., k$ and for each of such cluster $\mathcal{X}_i$ an affine regression function $f_{\mathcal{X}_i}$ is learned to model the data distribution in that particular cluster. Each affine function $f_{\mathcal{X}_i}$ is expected to forecast accurately given the data in the corresponding cluster i, and the accuracy is expected to decay as the data point moves away from that cluster. This motivates us to measure the closeness of the query point x to each cluster and use them as a surrogate of the probability vector $p(x)$. The problem then becomes how to determine the membership of the query point x among clusters $\mathcal{X}_i, i = 1, ..., k$, which can be considered as a standard multi-class classification problem.

Chapter 3

Hierarchical Multi-label Learning

3.1 Introduction

Multi-label learning considers a class of supervised learning problems where each sample is associated with multiple labels. Hierarchical Multi-label Learning (HML), in particular, considers scenarios where labels are connected by a hierarchical structure, and consequently this structure has to be imposed on the predicted labels. Modeling and exploiting label structure has been the main focus of recent research on multi-label learning. Most work along this line is focused on *label space reduction* by using, for example, random projections, principal component analysis and canonical correlation analysis [61–67]. The overall framework connecting these approaches can be described as follows:

- **Encoding.** Given a label vector $y \in \mathcal{Y} \subset \mathbb{R}^L$, the codeword z is obtained from a linear projection $P \in \mathbb{R}^{m\times L}$:

  $$z = Py \tag{3.1.1}$$

  An m-dimensional vector $z \in \mathcal{Z} \subset \mathbb{R}^m$ is used to encode y, which is of dimension L ($m \ll L$). The key assumption here is that while the labels are correlated the vector y is often sparse, so that the dimension reduction can be used and later the full label vector can be recovered without loss of information.

- **Training.** Given the encoding matrix P and the training set $\mathcal{M} = \{x_i, Py_i\}_{i=1}^n$ where $x_i \in \mathcal{X} \subset \mathbb{R}^d \; \forall i \in \{1, ..., n\}$, a prediction model $\mathcal{H} : \mathcal{X} \to \mathcal{Z}$ is trained to predict the codeword z, i.e., a linear predictor $\mathcal{H}(\cdot) = Wx$ where $W \in \mathcal{R}^{L \times d}$.
- **Prediction.** An unknown data point x_+ is observed and the prediction model $\mathcal{H}$ is used to obtain codeword

$$z_+ \leftarrow \mathcal{H}(x_+)$$

- **Decoding.** The unknown sparse label vector y_+ is recovered from the predicted codeword z_+ by solving an optimization problem.

Hierarchical multi-label structure appears in many real-world recommendation and targeting systems. For example, consider the following problem: in display-ads targeting each user is associated with a set of labels such as sports, football, vehicles, luxury cars, etc., and each label contains ads campaign that this user is likely to interact with. It is often the case that the labels are not independent of each other, and that their interdependence can be characterized by a connected (tree) structure, e.g., football is the child of sports. The task then is to assign each user with a set of labels that both appeal to the user and satisfy the underlying taxonomy. Overlooking such structure not only undermines the predictive performance (loss of information), but also makes the results hard to interpret, e.g., predicting that a user is interested in football but not in sports.

There are two key goals of HML: to *exploit* the hierarchy structure of the labels during training and to *preserve* the same structure during prediction. The encoding step discussed above is the common way to exploit the structure of labels. However, there is no particular exploitation of the hierarchical structure, just the sparsity of y. Moreover, most methods in this field do not impose the structure when predicting the label vector. Some methods [34–37, 68] produce the structured label vector, by employing a top-down approach; i.e., they predict the part of the label vector that correspond to the upper levels of the taxonomy first, without considering the lower levels. This clearly is a suboptimal approach because the optimization is not performed over the entire vector y. A recent method proposed in [69] considers a different setting where it is mandatory for each prediction to end in a leaf node (no partial path is allowed). The method finds the hierarchical labels by maximizing the posterior probability of a given point x over the whole label space, which is formulated as a combinatorial optimization problem. By exploiting the hierarchy, the total complexity of the method is $O(h^2 L_l^2)$ where h is the height of the tree and L_l is the number of leaf nodes.

In this paper we propose a novel two-stage HML method with $O(L)$ complexity in *decoding* optimal hierarchical labels. In particular, it exploits the hierarchical label structure during the training and also imposed this structure explicitly on the label during the prediction, by solving a convex optimization problem based on structured sparse learning. Most of the existing structure learning methods address the structure of the predictors (such as sparse classifiers), but not the labels. Hence our first contribution in this paper is to propose an optimal *decoding* scheme that efficiently induces both sparsity and the hierarchical structure on predicted labels. Given a predictor W, computed during training, and a input vector x, the product Wx is typically an unstructured dense vector. The true label vector, however, is not only sparse (with the non zeros in y denoting positive labels) but has to satisfy the hierarchical structure (if a parent label is negative, then so are the labels for its children). In prior approaches a unstructured sparse vector has been recovered given Wx, for example, by using a sparsity inducting ℓ_1 norm. Here we propose the use of the so-called hierarchical sparsity-inducing norm, parameterized by weights for each subtree in the hierarchy. Hence our method finds the optimal sparse vector that respects both the taxonomy and the unstructured prediction. The hierarchical norm that we use is a convex function and is a variant of group-lasso penalty [14]. It allows us to employ an efficient convex optimization algorithm that enjoys linear complexity in the label dimension [70]. We note that while the hierarchical norm has been explored before (see, e.g., [70–72]), this is the first time, to the best of our knowledge, that it has been used in the context of inducing hierarchical structure among the output of the HML problems.

In prior work on using hierarchical norm it has been frequently observed that the outcome is strongly dependent on the choice of weights corresponding to the subtrees. Several heuristic approaches to choosing such weights have been, thus, proposed. As we will see in Section 3.3, the weight parameters in hierarchical norm are not scale-invariant to input data, which make it difficult in our case to choose the correct values for the weights independently of the input data and the classifiers. This leads to our second contribution. First, we describe an ideal learning model that jointly trains the multi-label classifier and the hierarchical weights. This model, is a bi-level optimization problem with a non smooth convex lower-level problem, which make it difficult to solve efficiently in large scale setting. We then propose a two-stage training scheme that decouples the joint learning problem into a sequence of two optimization problems, that each can be efficiently solved. In particular, we base the first-level training on the multi-label Empirical Risk Minimization (ERM) framework [15] where various convex loss functions can be employed, i.e., least squares, logistic, or hinge loss. The output of the first problem is then used as the input of the second problem, where the hierarchical weights are learned while taking into account of both the classifiers and the encoded hierarchical information present in the training data. While the second problem still is a bi-level optimization problem, its dimension is now significantly reduced to the order of the number of labels that is well under 1000 in this work. In this paper we approach the bi-level problem as a black-box optimization problem, which we solve using an efficient derivative-free optimization methods [16]. The details can be found in Section 3.4.

Finally we demonstrate the performance of HML on both 1) publicly available document classification dataset and 2) a proprietary large scale ad targeting dataset constructed from 2 million users' daily activities in a taxonomy tree with 1500 labels.

3.2 Hierarchical sparsity-inducing norms

The group-Lasso penalty, which is known to be useful in inducing structured output, is an extension of Lasso that promotes sparsity in the group level as opposed to ℓ_1-norm penalty, which promotes sparsity among individual variable. In its simplest form, the group Lasso partitions the variables into non-overlapping groups with all members of the group either being set to zero or not simultaneously. Recent studies [70–72] have used more complicated and structured groups to induce complex patterns such as tree hierarchy The idea behind the so-called hierarchical norm is based on the observation that

$$\text{a variable is zero} \Rightarrow \text{its descendants variables are zeros}$$

This gives us a natural grouping of variables which aims to impose the given hierarchical structure: any given variable and all its descendants form a group.

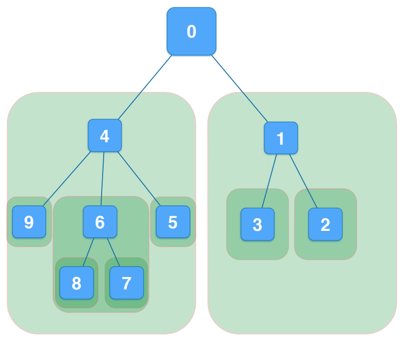
To formalize this intuition, we introduce notations on general group-Lasso penalty and hierarchical structure. Mathematically speaking, a group-Lasso penalty equipped with the Euclidean norm $||\cdot||$ can be described as follows:

$$\Omega_{\mathcal{G}}(x) = \sum_{g \in \mathcal{G}} \phi_g ||x_g||$$

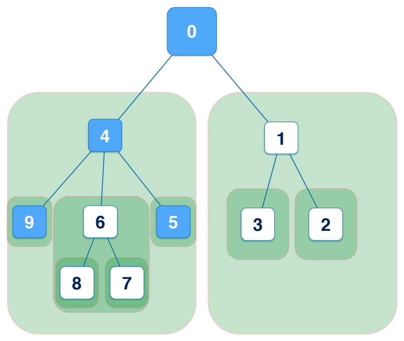
where $g \in \mathcal{G}$ forms a subset of the index set of all the coordinates in x and $\mathcal{G}$ is a collection of such index sets. If, for example, x is a vector in space $\mathbb{R}^n$, then g is a subset of $\{1, ..., n\}$ and x_g is a vector of dimension $|g|$.

In this work particularly, we are interested in the case when the set $\mathcal{G}$ and the groups $g \in \mathcal{G}$ are associated with a given tree structure, e.g., each group g consists of nodes that form a *rooted subtree* of the tree. In particular, let $\mathcal{T}$ denote a tree that consists of a set of nodes indexed by $\{1, ..., L\}$. For each node $t \in \mathcal{T}$, let $\mathcal{C}(t)$ denote the set of children of t, $\mathcal{A}(t)$ - the set of ancestors of t (including t), and $\mathcal{D}(t)$ - the set of descendants of t (including t). Then, we define the set $\mathcal{G}$ associated with such hierarchical structure $\mathcal{T}$ by

$$\mathcal{G}_{\mathcal{T}} := \{\mathcal{D}(t) : \forall\ t \in \mathcal{T}\} \tag{3.2.1}$$



(a) The tree structure is illustrated with blue rectangles as nodes and straight lines as edges. $\mathcal{G}_\mathcal{T} = \{g_t\}_{t=1}^{9}$, where $g_t = \mathcal{D}(t)$ $\forall t = 1, ..., 9$, e.g., $g_1 = \{1, 2, 3\}$, $g_4 = \{4, 5, 6, 7, 8, 9\}$.



(b) Each green rectangle and the nodes it includes represents a group and the variables it contains. Setting group g_1 and g_6 to zeros gives us a set of non-zero nodes that also satisfy the tree definition..

Figure 3.1: The tree structure is illustrated with blue rectangles as nodes and straight lines as edges. Groups g are green translucent rectangles covering nodes that are contained in the group.

and the hierarchical norm by

$$\Omega_{\mathcal{G}_\mathcal{T}}(x) = \sum_{g \in \mathcal{G}_\mathcal{T}} \phi_g ||x_g||, \quad \phi_g > 0 \ \forall g \in \mathcal{G}_\mathcal{T}$$

We use an example in Figure 3.1 to illustrate how hierarchical norm $\Omega_\mathcal{T}(\cdot)$ works. The tree we use has 9 nodes plus one root node numbered 0. The set $\mathcal{G}_\mathcal{T}$ contains 9 groups and each group g includes a rooted subtree or a singleton. In Figure 3.1(b) $\Omega_\mathcal{T}(\cdot)$ results in variables in groups g_1 and g_6 to equal zero, and the remaining non-zero variables form a tree hierarchy.

3.3 Hierarchical Multi-label Model

The model we propose for hierarchical multi-label learning problem is built on the empirical risk minimization (ERM) framework, and make use of the *hierarchical operator*:

$$P_{\mathcal{T}}(u,\phi) := \arg\min_{v} \frac{1}{2}||u-v||_2^2 + \sum_{g\in\mathcal{G}_{\mathcal{T}}} \phi_g ||v_g|| \tag{3.3.1}$$

where ϕ is a vector with each entry ϕ_g being the weight of group g in $P_{\mathcal{T}}(\cdot,\cdot)$. It then follows from the descriptions of hierarchical norms in Section 3.2 that the non-zero pattern in $P_{\mathcal{T}}(\cdot,\cdot)$ always satisfies the underlying hierarchical structure specified by $\mathcal{T}$ for any given input pair $\{u,\phi\}$, although such pattern of $P_{\mathcal{T}}(\cdot,\cdot)$ is not necessarily always the same as that of u even if the pattern of u satisfies the same hierarchical structure. In fact, the following lemma follows directly from Theorem 2 presented below:

Lemma 1. *We have $s_v \subseteq s_u$ where $s_v = \{i : P_{\mathcal{T}}(u,\phi)^{(i)} \neq 0\}$ and $s_u = \{i : u^{(i)} \neq 0\}$.*

Remark 1. *The zero pattern in $P_{\mathcal{T}}(u,\phi)$ is created from the pattern of a given vector u by setting some groups $g \in \mathcal{G}$ to zero, depending on the norm of u_g and ϕ_g. If $\phi_g \geq ||u||$, for example, where g is the group at the root which contains all variables, then all variables in g are set to zero, and $P_{\mathcal{T}}(u,\phi)$ is zero. Hence choosing correct values for $\{\phi_g\}_{g\in\mathcal{G}}$, taking the hierarchy and the vector u into account, is critical to recovering the structures in $P_{\mathcal{T}}(u,\phi)$.*

We now present the model for the HML problem. Ideally, we want to learn the parameters ϕ and W simultaneously so that W is chosen according to the penalty $\Omega_{\mathcal{G}_{\mathcal{T}}}$ (among other things) and ϕ are chosen according to the scale of Wx. Specifically, given $P_{\mathcal{T}}(u,\phi)$, any loss function $\ell(\cdot,\cdot)$ and the training set $\mathcal{M} = \{x_i, y_i\}_{i=1}^n$ where $x_i \in \mathcal{X} \subset \mathbb{R}^d, y_i \in \mathcal{Y} \subset \mathbb{R}^L \ \forall i \in \{1,...,n\}$, we consider the following ℓ_1-regularized problem:

$$\min_{\phi,W} \sum_{(y_i,x_i)\in\mathcal{M}} \frac{1}{|\mathcal{M}|}\ell(y_i, P_{\mathcal{T}}(Wx_i,\phi)) + \lambda||W||_1 \tag{3.3.2}$$

where $||W||_1 := \sum_{i=1}^{L}\sum_{j=1}^{d}|W_{ij}|$ is the element-wise ℓ_1 norm of the $L \times d$ matrix W. We use feature selection here because in most multi-label applications the data matrix X contains all features, that are relevant for the whole set of labels. it is, hence, useful to select the most relevant features for prediction of particular labels.

After obtaining W and ϕ, given an observed unknown data point x_+ we generate the predicted set of labels by the following *hierarchical projection* step:

$$y_+ = P_{\mathcal{T}}(Wx_+, \phi). \tag{3.3.3}$$

This step ensures that the output y_+ complies with the underlying taxonomy structure. Note that optimizing jointly over W and ϕ in Problem (3.3.2) is difficult given the non-smoothness of hierarchical operator $P_{\mathcal{T}}(\cdot,\cdot)$ and the often high dimensionality of W. Hence, instead, we propose to use the following decoupling scheme:

- Solve for W^* without the hierarchical operator:

  $$W^* \leftarrow \arg\min_{W} \sum_{(y_i,x_i)\in\mathcal{M}} \frac{1}{|\mathcal{M}|}\ell(y_i, Wx_i) + \lambda||W||_1 \tag{3.3.4}$$

  which is the standard ERM problem for each label, i.e., sparse logistic regression or Lasso depending on which loss function is involved.

- Solve for ϕ^* with W fixed to W^*:

  $$\phi^* \leftarrow \arg\min_{\phi} \sum_{(y_i,x_i)\in\mathcal{M}_\phi} \frac{1}{|\mathcal{M}_\phi|}\ell(y_i, P_{\mathcal{T}}(W^*x_i, \phi)) \tag{3.3.5}$$

Problem (3.3.5) minimizes the loss between the true label and the hierarchically projected point $P_{\mathcal{T}}(\cdot, \phi)$, using the input values of W^*, on the set $\mathcal{M}_\phi$, which can be chosen as a subset of the training set $\mathcal{M}$.

Problem (3.3.5) is still non-smooth, but by fixing the matrix W the dimension of the problem is greatly reduced to the number of groups in $\mathcal{G}_{\mathcal{T}}$, which is equal to the number of labels L and is several hundreds in the examples used in our experiments. We consider Problem (3.3.5) as a black-box optimization problem, which allows us to search for the optimal solution of Problem (3.3.5) without the use of the derivative information. In the next section we will see that the main effort of each step of this optimization lies in evaluating the hierarchical operator (3.3.1) that can be done efficiently by exploiting the hierarchical group structure.

3.4 Learning ϕ

In this work we make use of the basic model-based trust region derivative-free method (DFO) [73] for learning hierarchical group weights ϕ. While the DFO method and the underlying convergence theory have been designed for smooth problems, it is known to perform very well in practice on non smooth problems such as Problem (3.3.5). In particular, it tends to significantly outperform to other existing approaches, i.e., direct-search methods which applies to no smooth problems (see, e.g., [74]). In fact, we observe that in our experiments DFO achieves a significant improvement of objective function value and the resulting weights ϕ along with W obtained by solving Problem (3.3.4) give a superior prediction performance compared with traditional multi-label classifiers. Next we give a brief introduction to the basic ideas behind DFO, as well as the efficient function evaluation of Problem (3.3.5). We refer the interested readers to the book by [16] for more details about various DFO approaches.

3.4.1 Derivative-Free Optimization

DFO is based on the trust-region framework, which iteratively minimizes a local model to obtain the next step in a region around the current iterate, where the local model is believed to provide a good approximation to the objective function. The minimizer of the local model, thus, serves as the candidate for the next iterate, and depending on the quality of the minimizer, i.e., how much it decreases the objective function, the size of the region, where the local model is defined, is dynamically expanded or shrunk. The most commonly used local model is the Taylor expansion in the case when derivative is available. In the context of DFO, typically, an interpolation polynomial is constructed from a maintained set of sample points. The model is deemed sufficiently accurate if its accuracy matches the order of that of a Taylor model.

The main computational effort of DFO while minimizing Problem (3.3.5) is spent on objective function evaluations. Here, to construct a sufficiently accurate (first-order) model, it takes $O(L)$ function evaluations with L being the dimension of the objective function (number of labels in our case). After the first $O(L)$ evaluations, DFO maintains the sample set of the $O(L)$ most recently evaluated points and only updates one new point for each iteration while reusing most of the existing points. So after K iterations, the total number of function evaluations done by DFO is $O(L+K)$. While the worst case complexity of DFO is not polynomial, in practice good solutions are often obtained in $O(L)$ function evaluations. Next we discuss the complexity of each function evaluation in Problem (3.3.5).

3.4.2 Complexity of hierarchical operator

It is well known that the dual problem of $P_{\mathcal{T}}(\cdot, \phi)$ with fixed ϕ is the following:

$$\begin{aligned} &\max_{\zeta} \; -\frac{1}{2}(||u - \sum_{g \in \mathcal{G}} \zeta_g||^2 - ||u||^2) \\ &\text{s.t.}\, \forall g \in \mathcal{G} \; ||\zeta_g||_* \leq \phi_g, \zeta_g^{(j)} = 0 \; \forall j \notin g \end{aligned} \tag{3.4.1}$$

In the simplest case when $\mathcal{G}$ constitutes a partition of the index set, it is easy to see that the dual problem (3.4.1) is separable, and thus each dual variable ζ_g can be obtained independently of the others by projecting u_g onto the dual norm ball $\mathcal{B}^*_{\phi_g} = \{x : ||x||_* \leq \phi_g\}$. In general, from strong duality, we have the following optimality conditions:

$$P_{\mathcal{T}}(u, \phi) = u - \sum_{g \in \mathcal{G}} \zeta_g \tag{3.4.2}$$

$$\forall g \in \mathcal{G}, \zeta_g = \text{Proj}_{\mathcal{B}^*_{\phi_g}}(P_{\mathcal{T}}(u, \phi) + \zeta_g) \tag{3.4.3}$$

For general overlapping groups one can iteratively apply (3.4.3) to update dual variables $\{\zeta_g\}_{g\in\mathcal{G}}$ until the change in the update is below a given threshold. Then the primal solution from $\{\zeta_g\}_{g\in\mathcal{G}}$ can be recovered using (3.4.2). By exploiting the hierarchical structure in groups, however, a recursive one-pass algorithm to compute $P_{\mathcal{T}}(u, \phi)$ is available and is given in Theorem 2 which follows from result in [70].

Theorem 2. *Let $u \in \mathbb{R}^L$, $\Omega_{\mathcal{T}}(\cdot)$ be associated with ℓ_2-norm and $\preceq$ denote a total order relation such that $g_i \preceq g_j \Rightarrow \{g_i \subseteq g_j$ or $g_i \cap g_j = \emptyset\}$. Then, initializing $v \leftarrow u$ and in order $\preceq$ applying update*

$$v_g \leftarrow \max(1 - \phi_g/||v_g||_2, 0)v_g \tag{3.4.4}$$

once and for all $g \in \mathcal{G}_{\mathcal{T}}$ returns v as $P_{\mathcal{T}}(u, \phi)$. The whole computation can be done in $O(L)$ operations.

Remark 2. *The order $\preceq$ requires that updates* (3.4.4) *start from the bottom of the tree and propagate all the way to the root node. The ℓ_2-norm term appearing in the* $\max(\cdot)$ *function can thus be accumulated as the updates are propagated upwards. This can be done recursively so that the sum of squares of the variables is maintained and updated by adding one term for each node, which is the key to the linear complexity (w.r.t. number of variables) of $P_{\mathcal{T}}(u, \phi)$.*

Remark 3. *From Theorem 2 we conclude that the use of $\Omega_{\mathcal{G}_{\mathcal{T}}}(\cdot)$ indeed imposes the tree structure on v. By definition* (3.2.1) *of g, the group-thresholding effect on g ($v_g \leftarrow 0$) always applied to the whole subtree, including all groups h such that $h \subseteq g$. During the process* (3.4.4) *if a group is set to zero, then it will remain zero at the later steps, thanks to the nature of the update.*

Remark 4. *The group-thresholding effect occurs when we have $||v_g||_2 \leq \phi_g$. Hence a larger ϕ_g is more likely to set the variables in group g to zero.*

Table 3.1: Datasets statistics used in experiments. Dataset *itp* and *ads* have the same exact dimension, but are constructed differently – *itp* targets users' general interest and includes users' activities from other category-related events besides ads view or ads click.

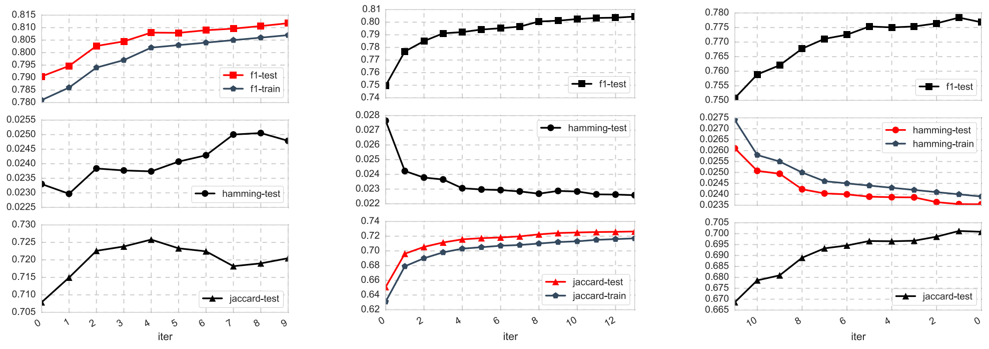
DATASETS	LABELS	FEATURES	TRAINING	TESTING
ITP	380	12,787	1,775,671	761,002
ADS	380	12,787	1,775,671	761,002
RCV	103	47,236	17,525	16,243

Table 3.2: The comparison of FLAT, HML, Leaf Reduce (LR) and Root Map (RM) on all three datasets, with squared and logistic loss function, and evaluated by precision, recall and f1 score.

		SQUARED				LOGISTIC			
DATA	EVAL	FLAT	HML	LR	RM	FLAT	HML	LR	RM
ITP	PREC	0.2216	0.2969	0.3152	0.1603	0.2221	0.2958	0.3133	0.1545
	REC	0.7343	0.4694	0.4212	0.7249	0.7383	0.4893	0.4492	0.7496
	F1	0.2956	**0.3331**	0.3230	0.2351	0.2960	**0.3354**	0.3275	0.2356
ADS	PREC	0.2065	0.2737	0.3054	0.1394	0.2079	0.2717	0.3092	0.1361
	REC	0.7013	0.4432	0.3588	0.7576	0.7065	0.4843	0.3785	0.7980
	F1	0.2732	**0.3034**	0.2926	0.2135	0.2744	**0.3197**	0.2942	0.2155
RCV	PREC	0.8737	0.7784	0.8816	0.8710	0.8664	0.8240	0.8744	0.8603
	REC	0.6772	0.7651	0.6718	0.6821	0.7648	0.8193	0.7555	0.7733
	F1	0.7382	**0.7404**	0.7357	0.7396	0.8046	**0.8157**	0.8015	0.8061

3.5 Other hierarchical methods

Let us consider two simple hierarchical-inducing methods. They are denoted by Leaf Reduce (LR) and Root Map (RM), and presented in Algorithm 11 and Algorithm 12, respectively. To describe those two methods, we also define two recursive hierarchical operations ρ-treeMap and η-treeReduce in Algorithm 9 and Algorithm 10. We will see that LR and RM can be expressed by η-treeReduce and ρ-treeMap, respectively.



(a) f1 score (b) jaccard similarity score (c) hamming loss

Figure 3.2: Testing and training performance under different metrics as learning objective on *rcv* dataset. Take (b) for example. HML equipped with jaccard similarity as the learning objective is applied on a training set of 3000 instances. Solutions after each iteration (the x-axis) of HML are recorded and later evaluated under all three metrics on the full *rcv* dataset (test set). The red curve denotes the objective values evaluated on the test set. In the same subplot along with the red curve are the objective values evaluated at different iterations on the training set. Other two subplots illustrate the same solutions evaluated by other two metrics on the same test set.

While these are two conceptually simple operations, together they are able to produce sophisticated algorithms, as demonstrated in Algorithm 13. In fact, it can be shown that Algorithm 13 is essentially equivalent to the hierarchical operator (3.3.1). This also shed light on the underlying connections between those methods, that the hierarchical operator (3.3.1) is in fact a intricate combination of LR and RM, where the two operations ρ-treeMap and η-treeReduce are integrated under a single objective of inducing hierarchy. The resulting combination enables a richer representation and can thus be more potent than either of the individual compounds alone.

In Algorithm 9 and 10 recall that for a node t its parent node is p_t and its set of children is $\mathcal{C}(t)$. We use Γ to denote binary operators such as $OR, AND, +, *$, etc., and R to denote a general function. A simple example would be ρ-treeMap$(0, +)$ that computes the depth for each node t, if we let $\rho_{p_0} = -1$ and initialize $\rho_t = 1$ for

Algorithm 9: ρ-treeMap(t, Γ)

```
1 $\rho_t \leftarrow \Gamma(\rho_t, \rho_{p_t})$
2 for $c$ in $\mathcal{C}(t)$ do
3     $\rho$-treeMap$(c, \Gamma)$
```

all $t \in \mathcal{T}$. Note also that ρ-treeMap modifies global variables $\{\rho_t\}_{t\in\mathcal{T}}$ in place and that η-treeReduce modifies both $\{\rho_t\}_{t\in\mathcal{T}}$ and $\{\eta_t\}_{t\in\mathcal{T}}$.

Algorithm 10: η-treeReduce$(t, \Gamma_1, \Gamma_2, R)$

```
1 for $c$ in $\mathcal{C}(t)$ do
2     $\eta_t \leftarrow \Gamma_1(\eta\text{-treeReduce}(c, \Gamma_1, \Gamma_2, R), \eta_t)$
3 $\rho_t \leftarrow R(\eta_t, \phi_t, u_t)$
4 return $\Gamma_2(\eta_t, \rho_t)$
```

LR is described in Algorithm 11. The main part is a call to η-treeReduce which calls itself recursively until a leaf node is reached. Then it returns 1 if $u_t \geq \phi_t$ and 0 otherwise. For all the non-leaf nodes, the return value is non-zero as long as any one of its children returns a non-zero value, or the inequality $u_t \geq \phi_t$ holds.

RM is described in Algorithm 12. Contrary to LR, RM starts at the root of the hierarchy, and sets the whole subtree to zero if $\rho_t = 0$ $(u_t < \phi_t)$, t the root of the subtree, due to the AND operator passed to ρ-treeMap.

The procedure for evaluating hierarchical operator (3.3.1) is described in Algorithm 13 using ρ-treeMap and η-treeReduce. Note that the call to ρ-treeMap is exactly the same as that appearing in RM. The difference is in the values of ρ. In RM, they are set for each t individually based on u_t and ϕ_t. In Algorithm 13, however, that is determined through a call to the η-treeReduce with a specific choice of the binary operators Γ_1, Γ_2 and function R. In particular, the value of ρ_t is returned by the function R and thus depends on the value of ϕ_t, the input parameter, and the value of η_t. And η_t is in turn determined by a recursive call to the η-treeReduce as shown in Algorithm 10. Note also that the function R returns 0 if $\phi_t > \sqrt{\eta_t}$, where the value of η_t is roughly the squared norm of the subtree vector u_{g_t} rooted at node t.

Algorithm 11: Leaf Reduce

```
1 INPUT: $u, \phi \in \mathbb{R}^L$
2 $\eta \leftarrow \mathbf{0}$
3 $R(\eta_t, \phi_t, u_t) := OR(I(u_t \geq \phi_t), \eta_t)$
4 $\eta$-treeReduce$(0, OR, OR, R)$ // 0 the root
5 return $v \leftarrow \rho$
```

Algorithm 12: Root Map

```
1 INPUT: $u, \phi \in \mathbb{R}^L$
2 $\rho_0 \leftarrow 1$
3 for $t$ in $\mathcal{T}$ do
4     $\rho_t \leftarrow I(u_t \geq \phi_t)$
5 for $t$ in $\mathcal{C}(0)$ do // children of the root
6     $\rho$-treeMap$(t, AND)$
7 return $v \leftarrow \rho$
```

Let us illustrate the difference between LR, RM and the hierarchical operator (Algorithm 13) using the following toy example. Consider a taxonomy of three nodes, indexed by 0, the root, and 1 and 2, its two children. Suppose all ϕ's are 0.5, i.e., $\phi_0 = \phi_1 = \phi_2 = 0.5$. Let us compare outputs of the three algorithms on this toy taxonomy given different values of input u. To start with, let us consider $u = (0.3, 0.9, 0.8)$. Rounding each entry individually yields $v = (0, 1, 1)$ which clearly does not respect the taxonomy. Applying RM on u we obtain $v = (0, 0, 0)$, while both LR and Algorithm 13 produce $v = (1, 1, 1)$. Both $(0, 0, 0)$ and $(1, 1, 1)$ respect the taxonomy but in this case $(1, 1, 1)$ is probably the better prediction. In fact, the two children nodes being very close to 1, i.e., $u_1 = 0.9$ and $u_2 = 0.8$, is a strong indicator that the classifier at the root probably made a mistake by producing a low score, i.e., 0.3, in u. Moreover, we find that RM is not able to correct that error because it makes the decision to set the whole taxonomy to 0 at the root without even looking at the values of its children. Now let us consider a different set of values for the input $u = (0.3, 0.3, 0.8)$, where we reduce the value for node 1 from 0.9 to 0.3. Again, rounding with no regards to the taxonomy fails to produce taxonomy-compatible solution. LR, RM, and Algorithm 13 yield $(1, 1, 1)$, $(0, 0, 0)$ and $(0, 0, 0)$, respectively. In this case, we think $(0, 0, 0)$ is the better solution because both the root and one of its children, two nodes out of three in total, receive low scores in u. We find that this time RM produces the good prediction, but LR fails to do so, assigning all nodes 1 by only looking at the children, without taking into account the value at the root. Finally, we note that in both cases Algorithm 13 yields the better solution, precisely through the combination of ρ-treeMap and η-treeReduce which enables a global, and better, decision to be made.

3.6 Numerical Experiments

In the first part of the experiments we compare HML, the Hierarchical Multi-Label model described in Section 3.3, with FLAT that treat each label as a separate binary classification problem. We also include the other two aforementioned hierarchical methods LR and RM in the comparison. The results are reported in Table 3.2. In the second part we demonstrate the learning effectiveness of HML by comparing the testing and the training performance for each iteration of HML. We also show that HML is able to directly optimize any objective such as f1, jaccard similarity, hamming loss, etc., even though the underlying optimization problem is both non-smooth and non-convex. The results are illustrated in Figure 3.2.

We consider the following two applications (related datasets are described in Table 3.1):

Online ads targeting: We created features using user activity logs collected at servers of a large internet company. The activities included visited web pages, issued search queries, as well as links and ads clicked during the period of three months.

Algorithm 13: Hierarchical Operator (3.3.1)

```
1 **INPUT:** $u, \phi, \in \mathbb{R}^L$
2 $\rho_0 \leftarrow 1$
3 for $t$ in $\mathcal{T}$ do
4     $\eta_t \leftarrow u_t^2$
5 $R(\eta_t, \phi_t, u_t) := (1 - \phi_t/\sqrt{\eta_t})_+$
6 $\eta$-treeReduce$(0, +, *, R)$
7 for $t$ in $\mathcal{C}(0)$ do
8     $\rho$-treeMap$(t, AND)$
9 return $v \leftarrow \rho$
```

All activities were classified into a in-house hierarchical taxonomy consisting of 1,500 nodes. In the experiments we used two different data sets: 1) *ads* dataset, where we predict the probability that a user will *click* on an ad from a specific category; and 2) *itp* dataset, where we predict the probability that a user will *interact* with a specific category (either by clicking on the ad from that category, or clicking on a search result from that category, or any other category-related event). In order to generate labels, we selected 380 categories for which we had more than 1,000 positive examples, defined as a user who either clicked on a categorized ad in the *ads* dataset, or interacted with category in the *itp* dataset. We retained only active users (those who had positive labels in at least two paths of the taxonomy), which resulted in a data comprising around 2 million users. 70% of the data set was used to train 380 category-specific models, optimizing for both logistic and squared loss, and the remaining 30% of the data set was used for testing.

Documents classification: Here we use the publicly-available text categorization test collection RCV1-v2/LYRL2004 made available from Reuters Corpus Volume I (RCV1) [43]. Each RCV1-v2 document used in our experiment has been tokenized, stopworded, stemmed and represented by 47,236 features in a final form as *ltc* weighted vectors. Those RCV1 documents are categorized with respect to the *Topics*, which were organized in four hierarchical groups. We go through the entire dataset and filter out those documents with less than 5 labels. We end up with $33,768$ documents in total for training and testing.

Let us look at the results presented in Table 3.2. In particular, HML is applied to maximize F1 score on the training set, hence if we only look at rows denoted by F1 score, then clearly HML outperforms FLAT and other methods in all experiments involving either squared or logistic loss functions. Note that the results are all evaluated on the full datasets. It demonstrates the generalization ability of HML, as illustrated in Figure 3.2.

The messages become less clear-cut if we look at the other two rows denoted by precision and recall. In general we notice that HML and LR achieve a higher precision while FLAT and RM has a better recall. For the three hierarchical methods, LR and RM are more extreme in terms of precision and recall, and HML strike a balance between those two, achieving better performance which in this case is defined by a higher F1 score.Finally we note that the superiority of HML is more evident in datasets *itp* and *ads*, which possesses more complicated taxonomy structure among labels and also more number of labels than *rcv* does.

Figure 3.2 includes two more metrics other than F1. The message is clear when we look at the plots with red curves. It shows that while the learning algorithm is only trained on the training set, the performance on the unseen testing set tracks closely what is observed on the smaller training set, no matter what objective is used. Optimizing one metric also improves, in general, the performance measured by other metrics. That improvement might not be monotonic as is the case of the optimized metric. While choosing which metric to optimize will depend on the particular application and other practical concerns, HML, as it shows, provides an effective learning algorithm for improving different metrics in hierarchical problems.

3.7 Conclusion

In this paper we studied multi-label learning problems with hierarchical structure present on labels. We devise a novel hierarchical *decoding* scheme that efficiently imposes hierarchical structure on predictions, and we propose an ideal bi-level optimization problem for jointly learning the classifiers and the hierarchical weights that we find are highly influential on the predictive performance. We also propose an efficient decoupling scheme for learning the weights and the classifiers in a sequential two-stage process. We demonstrate the superiority of our model on large-scale real-world datasets.

Chapter 4

Complexity of Inexact Proximal Newton Methods

4.1 Introduction

In this chapter, we are interested in the following convex optimization problem:

$$\min_{x\in\mathbb{R}^n} F(x) \equiv f(x) + g(x) \tag{4.1.1}$$

where $f, g : \mathbb{R}^n \to \mathbb{R}$ are both convex functions such that f is twice differentiable, with a bounded Hessian, and $g(x)$ is such that the following problem is easy to solve (approximately) for any $z \in \mathbb{R}^n$ and some class of positive definite matrices H, relative to minimizing $F(x)$:

$$\min_{x\in\mathbb{R}^n} \left\{ g(x) + \frac{1}{2}\|x - z\|_H^2 \right\}. \tag{4.1.2}$$

Here $\|y\|_H^2$ denotes $y^\top Hy$ (on occasion, in this chapter, we abuse the norm notation by using H matrix that is not positive definite).

This chapter is organized as follows: in Section 4.2 we describe the algorithmic framework. Then, in Section 4.3 we present some of the assumptions and discussions and, in Section 4.3.1, convergence rate analysis based on [2] and [23]. In Section 4.4

we show the new convergence rate analysis using [24–26] for exact and inexact version of our framework. We then extend the analysis for the cases where inexact solution to a subproblem is random in Section 4.5 and in particular to the randomized coordinate descent in Section 4.5.1.

4.2 Basic algorithmic framework and theoretical analysis

The following function is used throughout the paper as an approximation of the objective function $F(x)$.

$$Q(H,u,v) := f(v) + \langle \nabla f(v), u-v\rangle + \frac{1}{2}\langle u-v, H(u-v)\rangle + g(u). \qquad (4.2.1)$$

For a fixed point $\bar{x}$, the function $Q(H,x,\bar{x})$ serves as an approximation of $F(x)$ around $\bar{x}$. Matrix H controls the quality of this approximation. In particular, if $f(x)$ is smooth and $H = \frac{1}{\mu}I$, then $Q(H,x,\bar{x})$ is a sum of the prox-gradient approximation of $f(x)$ at $\bar{x}$ and $g(x)$. This particular form of H plays a key role in the design and analysis of proximal gradient methods (e.g., see [2]) and alternating direction augmented Lagrangian methods (e.g, see [8]). If $H = \nabla^2 f(\bar{x})$, then $Q(H,x,\bar{x})$ is a second order approximation of $F(x)$ [28, 75]. In this paper we assume that H is a positive definite matrix such that $\sigma I \preceq H \preceq MI$ for some positive constants M and σ.

Minimizing the function $Q(H,u,v)$ over u reduces to solving problem (4.1.2). We will use the following notation to denote the accurate and approximate solutions of (4.1.2).

$$p_H(v) := \arg\min_u Q(H,u,v), \qquad (4.2.2)$$

and

$$p_{H,\phi}(v) \text{ is a vector such that : } \begin{array}{l} Q(H,p_{H,\phi}(v),v) \le Q(H,v,v) = F(v), \text{ and} \\ Q(H,p_{H,\phi}(v),v) \le Q(H,p_H(v),v) + \phi. \end{array} \qquad (4.2.3)$$

The method that we consider in this paper computes iterates by (approximately) optimizing $Q(H, u, v)$ with respect to u using some particular H which is chosen at each iteration. The basic algorithm is described in Algorithms 14 and 15.

Algorithm 14: Proximal Quasi-Newton method

```
1 Choose  $0 < \rho \leq 1$ and $x^0$;
2 for $k = 0, 1, 2, \cdots$ do
3     Choose $0 < \bar{\mu}_k, \phi_k > 0, G_k \succeq 0$;
4     Find $H_k = G_k + \frac{1}{2\mu_k} I$ and $x^{k+1} := p_{H_k}(x^k)$
5     by applying *Prox Parameter Update* $(\bar{\mu}_k, G_k, x^k, \rho)$;
```

Algorithm 15: Prox Parameter Update $(\bar{\mu}, G, x, \rho)$

```
1 Select $0 < \beta < 1$ and set $\mu = \bar{\mu}$;
2 for $i = 1, 2, \cdots$ do
3     Define $H = G + \frac{1}{2\mu} I$ and compute $p(x) := p_H(x)$;
4     If $F(p(x)) - F(x) \leq \rho(Q(H, p(x), x) - F(x))$, then output $H$ and $p(x)$,
      **Exit** ;
5     Else $\mu = \beta^i \bar{\mu}$;
```

Algorithm 15 chooses Hessian approximations of the form $H_k = \frac{1}{\mu_k} I + G_k$. However, it is possible to consider any procedure of choosing positive definite H_k which ensures $MI \succeq H_k \succeq \sigma I$ and $F(p_{H_k}(x)) - F(x) \leq \rho(Q(H_k, p_{H_k}(x), x) - F(x))$, for a given $0 < \rho \leq 1$, - a step acceptance condition which is a relaxation of conditions used in [2] and [23].

An inexact version of Algorithm 14 is obtained by simply replacing p_{H_k} by p_{H_k,ϕ_k} in both Algorithms 14 and 15 for some sequence of ϕ_k values.

4.3 Basic results, assumptions and preliminary analysis

ISTA [2] is a particular case of Algorithm 14 with $G_k = 0$, for all k, and $\rho = 1$. In this case, the value of μ_k is chosen so that the conditions of Lemma 4 hold with $\epsilon = 0$. In other words, the reduction achieved in the objective function $F(x)$ is at least the amount of reduction achieved in the model $Q(\mu_k, p(x^k), x^k)$. It is well known that as long as $\mu_k \leq 1/L(f)$ (recall that $L(f)$ is the Lipschitz constant of the gradient) then condition (4.3.7) holds with $\rho = 1$. Relaxing condition (4.3.7) by using $\rho < 1$ allows us to accept larger values of μ_k, which in turn implies larger steps taken by the algorithm. This basic idea is the cornerstone of step size selection in most nonlinear optimization algorithms. Instead of insisting on achieving "full" predicted reduction of the objective function (even when possible), a fraction of this reduction is usually sufficient. In our experiments small values of ρ provided much better performance than values close to 1.

In the next three sections we present the analysis of convergence rate of Algorithm 14 under different scenarios. Recall that we assume that $f(x)$ is convex and smooth, in other words $\|\nabla f(x) - \nabla f(y)\| \leq L(f)\|x - y\|$ for all x and y in the domain of interest, while $g(x)$ is simply convex. In Section 4.5.1 we assume that $g(x) = \lambda\|x\|_1$. Note that we do not assume that $f(x)$ is strongly convex or that it is twice continuously differentiable, because we do not rely on any accurate second order information in our framework. We only assume that the Hessian approximations are positive definite and bounded, but their accuracy can be arbitrary, as long as sufficient decrease condition holds. Hence we only achieve sublinear rate of convergence. To achieve higher local rates of convergence stronger assumptions on $f(x)$ and on the Hessian approximations have to be made, see for instance, [27] and [28] for related local convergence analysis.

First we present a helpful lemma which is a simple extension of Lemma 2 in [23] to the case of general positive definite Hessian estimate. This lemma establishes some simple properties of an ϕ-optimal solution to the proximal problem (4.1.2). It uses the concept of the ϕ-subdifferential of a convex function a at x, $\partial_\phi a(x)$, which is defined as the set of vectors y such that $a(x) - y^T x \leq a(t) - y^T t + \phi$ for all t.

Lemma 3. *Given $\phi > 0$, a p.d. matrix H and $v \in \mathbb{R}^n$, let $p_\phi(v)$ denote the ϕ-optimal solution to the proximal problem (4.1.2) in the sense that*

$$g(p_\phi(v)) + \frac{1}{2}\|p_\phi(v) - z\|_H^2 \leq \phi + \min_{x \in \mathbb{R}^n} \left\{ g(x) + \frac{1}{2}\|x - z\|_H^2 \right\}, \tag{4.3.1}$$

where $z = v - H^{-1}\nabla f(v)$. Then there exists η such that $\frac{1}{2}\|\eta\|_{H^{-1}}^2 \leq \phi$ and

$$H(z - p_\phi(v)) - \eta \in \partial_\phi g(p_\phi(v)). \tag{4.3.2}$$

Proof. (4.3.1) indicates that $p_\phi(v)$ is an ϕ-minimizer of the convex function $a(x) := \frac{1}{2}\|x - z\|_H^2 + g(x)$. If we let $a(x) = a_1(x) + a_2(x)$ with $a_1(x) = \frac{1}{2}\|x - z\|_H^2$ and $a_2(x) = g(x)$, then

$$0 \in \partial_\phi a(p_\phi(v)) \subseteq \partial_\phi a_1(p_\phi(v)) + \partial_\phi a_2(p_\phi(v)). \tag{4.3.3}$$

For any $y \partial_\phi a_1(p_\phi(v))$ we apply the definition of ϕ-subdifferential for point $t = H^{-1}y + z$ and we get

$$\frac{1}{2}\|y + H(z - p_\phi(v))\|_{H^{-1}}^2 \leq \phi,$$

from which it follows that

$$y = \eta - H(z - p_\phi(v)) \mid \frac{1}{2}\|\eta\|_{H^{-1}}^2 \leq \phi,$$

which together with (4.3.3) proves (4.3.2).
□

We will find the following bound useful on the norm of η which follows from the above lemma.

$$\|\eta_i\| \leq \sqrt{2\lambda_{max}(H)\phi_i}, \tag{4.3.4}$$

where λ_{max} is the largest eigenvalue of H or its upper bound.

Below are the assumptions made in our analysis.

Assumptions 1.

(i) The set of optimal solutions of (4.1.1)*, X^*, is nonempty and x^* is any element of that set.*

(ii) The effective domain of F is defined as $\mathrm{dom}(F) := \{x \in \mathbb{R}^n : F(x) < \infty\}$*, and the level set of F at point $x \in \mathrm{dom}(F)$ is defined by*

$$\mathcal{X}_F(x) := \{y \in \mathrm{dom}(F) : F(y) \le F(x)\}.$$

Without loss of generality, we restrict our discussions below to the level set $\mathcal{X}_0 := \mathcal{X}_F(x^0)$ given by some $x^0 \in \mathrm{dom}(F)$, e.g., the initial iterate of the Algorithm 14.

(iii) g is convex and Lipschitz continuous with constant L_g for all $x, y \in \mathcal{X}_0$:

$$g(x) - g(y) \le L_g \|x - y\|,$$

(iv) There exists positive constants M and σ such that for all $k \ge 0$, at the k-th iteration of Algorithm 14:

$$\sigma I \preceq \sigma_k I \preceq H_k \preceq M_k I \preceq MI \tag{4.3.5}$$

(v) There exists a positive constant $D_{\mathcal{X}_0}$ such that for all iterates $\{x^k\}$ of Algorithm 14:

$$\sup_{x^* \in X^*} \|x^k - x^*\| \le D_{\mathcal{X}_0}$$

In the analysis of the proximal gradient methods Assumption $1(v)$ is removed by directly establishing a uniform bound on $\|x^k - x^*\|$ (rf. [23]). In the next section we show an outline of a proximal-gradient type analysis where this assumption is imposed for simplicity of the presentation. The proximal gradient approach in the next subsection, however, requires another, stronger, assumption on H_k. In the alternative analysis that follows we impose Assumption 1(v) but relax the assumption on H_k matrices. Note that all iterates $\{x^k\}$ fall into the level set $\mathcal{X}_0$, due to the sufficient decrease condition (4.3.7) that demands a monotonic decrease on the objective values. Assumption 1(v) thus follows straightforwardly if the level set is bounded, which often holds in real-world problems or can be easily imposed.

4.3.1 Analysis via proximal gradient approach

In this section we extend the analysis in [2] and [23] to our framework under additional assumptions on the Hessian approximations H_k. The following lemma, is a generalization of Lemma 2.3 in [2] and of a similar lemma in [23]. This lemma serves to provide a bound on the change in the objective function $F(x)$.

Lemma 4. *Given ϵ, ϕ and H such that*

$$\begin{aligned} F(p_\phi(v)) &\leq Q(H, p_\phi(v), v) + \epsilon \qquad (4.3.6)\\ Q(H, p_\phi(v), v) &\leq \min_{x \in \mathbb{R}^n} Q(H, x, v) + \phi, \end{aligned}$$

where $p_\phi(v)$ is the ϕ-approximate minimizer of $Q(H, x, v)$, then for any η such that $\frac{1}{2}\|\eta\|^2_{H^{-1}} \leq \phi$ and for any $u \in \mathbb{R}^n$,

$$2(F(u) - F(p_\phi(v))) \geq \|p_\phi(v) - u\|_H^2 - \|v - u\|_H^2 - 2\epsilon - 2\phi - 2\langle \eta, u - p_\phi(v)\rangle.$$

Proof. The proof is an easy extension of that in [2]. □

Note that if $\phi = 0$, that is the subproblems are solved accurately, then we have $2(F(u) - F(p(v))) \geq \|p(v) - u\|_H^2 - \|v - u\|_H^2 - 2\epsilon$.

If the sufficient decrease condition

$$(F(x^{k+1}) - F(x^k)) \leq \rho(Q(H_k, x^{k+1}, x^k) - F(x^k)) \qquad (4.3.7)$$

is satisfied, then

$$\begin{aligned}F(x^{k+1}) &\leq Q(H_k, x^{k+1}, x^k) - (1-\rho)\left(Q(H_k, x^{k+1}, x^k) - F(x^k)\right)\\ &\leq Q(H_k, x^{k+1}, x^k) - \frac{1-\rho}{\rho}(F(x^{k+1}) - F(x^k))\end{aligned}$$

and Lemma 4 holds at each iteration k of Algorithm 14 with $\epsilon_k = -\frac{1-\rho}{\rho}(F(x^{k+1}) - F(x^k))$.

We now establish the sub linear convergence rate of Algorithm 14 under the following additional assumption.

Assumption 2. *Let $\{x^k\}$ be the sequence of iterates generated by Algorithm 14, then there exists a constant M_H such that*

$$\sum_{i=0}^{k-1}(\|x^{i+1} - x^*\|^2_{\frac{H_{i+1}}{M_{i+1}}} - \|x^{i+1} - x^*\|^2_{\frac{H_i}{M_i}}) \leq M_H, \quad \forall k, \tag{4.3.8}$$

where M_i is the upper bound on the largest eigenvalue of H_i as defined in Assumption 1(iv).

The assumption above is not verifiable, hence it does not appear to be very useful. However, it is easy to see that a condition $\frac{H_{i+1}}{M_{i+1}} \preceq \frac{H_i}{M_i}$, for all i, easily implies (4.3.8) with $M_H = 0$. This condition, in turn, is trivially satisfied if H_{i+1} is a multiple of H_i for all i, as it is in the case of ISTA algorithm. It is also clear that Assumption 2 is a lot weaker than the condition that H_{i+1} is a multiple of H_i for all i. For example, it also hold if H_{i+1} is a multiple of H_i for all i *except for a finite number of iterations.* It also holds if H_{i+1} converges to H_i (or to its multiple) sufficiently rapidly. It is also weaker than the assumption in [30] that $H_{i+1} \preceq H_i$. Exploring different conditions on the Hessian approximations that ensure Assumption 2 is a subject of a separate study. Below we show how under this condition sub linear convergence rate is established.

Theorem 5. *Suppose that Assumptions 1 and 2 hold. Assume that all iterates $\{x^k\}$*

of inexact Algorithm 14 are generated with some $\phi_k \geq 0$ (cf. (4.2.3)), then

$$F(x^k) - F(x^*) \leq$$
$$\frac{1}{k_m}\left(\frac{1}{2}\|x^0 - x^*\|^2_{H_0/M_0} + \frac{M_H}{2} + \frac{(1-\rho)(F(x^0)-F(x^*))}{\rho\sigma} + \sum_{i=0}^{k-1}\frac{\phi_i}{M_i} + D_{\mathcal{X}_0}\sum_{i=0}^{k-1}\sqrt{\frac{2\phi_i}{M_i}}\right), \tag{4.3.9}$$

where $k_m = \sum_{i=0}^{k-1} M_i^{-1}$.

Proof. Let us apply Lemma 4, sequentially, with $u = x^*$, $p_\phi(v) = x^i$ and subproblem minimization residual ϕ_i for $i = 0, \ldots, k-1$. Adding up resulting inequalities, we obtain

$$\sum_{i=0}^{k-1}\frac{F(x^{i+1}) - F(x^*)}{M_i}$$
$$\leq \frac{1}{2}\sum_{i=0}^{k-1}\left(\|x^i - x^*\|^2_{H_i/M_i} - \|x^{i+1} - x^*\|^2_{H_i/M_i}\right) + \sum_{i=0}^{k-1}(\frac{\phi_i}{M_i} + \frac{\epsilon_i}{M_i} + \frac{\langle \eta_i, x^* - x^{i+1}\rangle}{M_i})$$
$$= \frac{1}{2}\left(\|x^0 - x^*\|^2_{\frac{H_0}{M_0}} + \sum_{i=0}^{k-1}(\|x^{i+1} - x^*\|^2_{H_{i+1}/M_{i+1}} - \|x^{i+1} - x^*\|^2_{H_i/M_i}) - \|x^k - x^*\|^2_{H_k/M_k}\right) \tag{4.3.10}$$
$$+\sum_{i=0}^{k-1}\frac{\epsilon_i}{M_i} + \sum_{i=0}^{k-1}\frac{\phi_i}{M_i} + \sum_{i=0}^{k-1}\frac{\langle \eta_i, x^* - x^{i+1}\rangle}{M_i}.$$

From Assumptions 1 and 2 we have

$$\sum_{i=0}^{k-1}\frac{F(x^{i+1}) - F(x^*)}{M_i} \leq \frac{1}{2}\|x^0 - x^*\|^2_{H_0/M_0} + \frac{M_H}{2} + \sum_{i=0}^{k-1}\frac{\epsilon_i}{M_i} + \sum_{i=0}^{k-1}\frac{\phi_i}{M_i} + D_{\mathcal{X}_0}\sum_{i=0}^{k-1}\frac{\|\eta_i\|}{M_i}.$$

From (4.3.4) and definition of M_i we know that $\|\eta_i\| \leq \sqrt{2M_i\phi_i}$. Using the already established bound $\sum_{i=0}^{k-1}\epsilon_i \leq \frac{(1-\rho)(F(x^0)-F(x^*))}{\rho}$ we obtain

$$\sum_{i=0}^{k-1}\frac{F(x^{i+1}) - F(x^*)}{M_i}$$
$$\leq \frac{1}{2}\|x^0 - x^*\|^2_{H_0/M_0} + \frac{M_H}{2} + \frac{(1-\rho)(F(x^0)-F(x^*))}{\rho\sigma} + \sum_{i=0}^{k-1}\frac{\phi_i}{M_i} + D_{\mathcal{X}_0}\sum_{i=0}^{k-1}\sqrt{\frac{2\phi_i}{M_i}}.$$

Hence,

$$
\begin{aligned}
&F(x^k) - F(x^*)\\
&\le \frac{1}{k_m}\sum_{i=0}^{k-1}\frac{F(x^{i+1}) - F(x^*)}{M_i}\\
&\le \frac{1}{k_m}\left(\frac{1}{2}\|x^0 - x^*\|^2_{H_0/M_0} + \frac{M_H}{2} + \frac{(1-\rho)(F(x^0) - F(x^*))}{\rho\sigma} + \sum_{i=0}^{k-1}\frac{\phi_i}{M_i} + D_{\mathcal{X}_0}\sum_{i=0}^{k-1}\sqrt{\frac{2\phi_i}{M_i}}\right).
\end{aligned}
$$

□

Let us consider the term $\frac{1}{k_m} = \frac{1}{\sum_{i=0}^{k-1} M_i^{-1}}$. From earlier discussions, we can see that $\frac{1}{k_m} \le \frac{M}{k}$. Moreover, if H_k are diagonal matrices, then $M = L(f)$ and, hence $\frac{1}{k_m} \le \frac{L(f)}{k}$, which established a bound similar to that of proximal gradient methods. The role of M_i is to show that if most of these values are much smaller than the global Lipschitz constant $L(f)$, then the constant involved in the sub linear rate can be much smaller than that of the proximal gradient methods. This is a well known effect of using partial second order information and it is observed in our computational results.

Lemma 6. *Let v be the initial point and $Q^* := \min_{u\in\mathbb{R}^n} Q(H, u, v)$. If v_i is the random point generated by applying i randomized coordinate descent steps to a strongly convex function Q, then for some constant $0 < \alpha < 1$ (dependent only on n and σ - the bound on the smallest eigenvalue of H) we have*

$$E[Q(H, v_i, v) - Q^*] \le \alpha^i (Q(H, v, v) - Q^*). \tag{4.3.11}$$

Next, we establish a bound on the maximal possible reduction of the model function $Q(\cdot)$ objective function value, for any positive definite matrix $H \succ 0$, and fixed point $v \in \mathbb{R}^n$.

Lemma 7. *Assume that for any $v \in \mathbb{R}^n$ such that $F(v) \le F(x^0)$, all of the the subgradients of F at v are bounded in norm by a constant κ, i.e. $\|\nabla f(v) + \gamma\| \le \kappa$ for all $\gamma \in \partial g(v)$. Then the maximum function reduction for $Q(\cdot)$ is uniformly bounded from above by*

$$Q(H, v, v) - Q^* \le R, \text{ with } R = \tfrac{M\kappa^2}{2\sigma^2}, \tag{4.3.12}$$

where M and σ are respectively the bounds on the largest and smallest eigenvalues of H and $Q^* := \min_{u\in\mathbb{R}^n} Q(H,u,v)$.

Proof. *Let $v^* = \arg\min_{u\in\mathbb{R}^n} Q(H,u,v)$ and let $\gamma_g(v^*)$ be any subgradient of $g(\cdot)$ at v^*. From the first-order optimality conditions*

$$H(v^* - v) + \nabla f(v) + \gamma_g = 0, \tag{4.3.13}$$

we can obtain an upper bound on $\|v^* - v\|$

$$\|v^* - v\| = \|H^{-1}\| \cdot \|\nabla f(v) + \gamma_g\| \leq \kappa/\sigma. \tag{4.3.14}$$

Now, we bound the reduction in the objective function in terms of $||v^* - v||$. From the convexity of g,

$$g(v) - g(v^*) \leq \langle \gamma_g, v - v^* \rangle. \tag{4.3.15}$$

Multiplying (4.3.13) with $v^* - v$, we obtain

$$-\langle \nabla f(v), v^* - v \rangle = \|v^* - v\|_H^2 + \langle \gamma_g, v^* - v \rangle. \tag{4.3.16}$$

From (4.3.16), (4.3.13), (4.3.15) and the definition of Q, we have

$$\begin{aligned} Q(H,v,v) - Q^* &= g(v) - g(v^*) - \langle \nabla f(v), v^* - v \rangle - \frac{1}{2}\|v^* - v\|_H^2 \\ &\leq \langle \gamma_g, v - v^* \rangle + \|v^* - v\|_H^2 + \langle \gamma_g, v^* - v \rangle - \frac{1}{2}\|v^* - v\|_H^2 \\ &= \frac{1}{2}\|v^* - v\|_H^2 \leq \frac{M\kappa^2}{2\sigma^2}, \end{aligned}$$

which concludes the proof of the lemma. □

It follows immediately from Lemma 7 that the subproblem optimization error ϕ_i is also bounded for all i, say by Φ, and that $\Phi \leq R$. Using this bound we now present the auxiliary result that derives the bounds on separate terms that appear on the right hand side of (4.5.2) and involve ϕ_i.

Lemma 8. *Assume that ϕ_i are nonnegative bounded independent random variables whose value lies in an interval $[0, \Phi]$. Assume also that $E[\phi_i] \leq \bar{R}\bar{\alpha}^i$ for all i, and some constants $0 < \bar{\alpha} < 1$ and $\bar{R} > 0$. Then the following inequalities hold*

$$E[\sum_{i=1}^{k} \sqrt{\phi_i}] \leq \frac{\sqrt{\bar{R}\bar{\alpha}}}{1 - \sqrt{\bar{\alpha}}}, \quad E[\sum_{i=1}^{k} \phi_i] \leq \frac{\bar{R}}{1 - \bar{\alpha}} \tag{4.3.17}$$

Proof. First we note that

$$E[\sqrt{\phi_i \phi_j}] = E[\sqrt{\phi_i}]E[\sqrt{\phi_j}], \tag{4.3.18}$$

due to independence of ϕ_i's, and

$$E[\sqrt{\phi_i}] \leq \sqrt{E[\phi_i]}, \tag{4.3.19}$$

due to Jensen inequality and the fact that the square root function is concave and $\phi_i \geq 0$. Then, given (4.3.18) and (4.3.19), we derive (4.3.17) using a bound on $E[\phi_i]$.

$$E[\sum_{i=1}^{k} \sqrt{\phi_i}] = \sum_{i=1}^{k} E[\sqrt{\phi_i}] \leq \sum_{i=1}^{k} \sqrt{E[\phi_i]} \leq \sum_{i=1}^{k} \sqrt{\bar{R}}\bar{\alpha}^{i/2} \leq \frac{\sqrt{\bar{R}\bar{\alpha}}}{1 - \sqrt{\bar{\alpha}}}.$$

And the second bound on $E[\sum_{i=1}^{k} \phi_i]$ can be established in a similar fashion. □

Theorem 9. *Assume that for each iterate $\{x^k\}$ of Algorithm 17 the following iterate $\{x^{k+1}\}$ is generated by applying $ak + b$ steps of Algorithm 16, then*

$$E[F(x^k)] - F(x^*) \leq \frac{\zeta}{k},$$

where x^* is an optimal solution of (4.1.1) and ζ is a constant.

Proof. Taking expectation on both sides of (4.5.2) in Theorem 15, we have

$$E[F(x^k)] - F(x^*) \leq \frac{1}{k}\left(\frac{\sigma}{2}E[B_k] + \frac{\sigma}{2}C + 2E[A_k^2] + \sqrt{C}E[A_k] + E[\sqrt{B_k}A_k]\right). \tag{4.3.20}$$

Next, we show how to bound $E[B_k], E[A_k], E[A_k^2]$ and $E[\sqrt{B_k}A_k]$. We first note a key observation, that, as a result of Lemma 19, Lemma 7 and Algorithm 16, the expectation of subproblem optimization error ϕ_i can be upper bounded by a geometric progression such that

$$E[\phi_i] \leq R\alpha^{ai+b} = \bar{R}\bar{\alpha}^i,$$

where $0 < \alpha < 1$ and R are specified respectively in Lemma 19 and Lemma 7 and $\bar{\alpha} = \alpha^a$ and $\bar{R} = R\alpha^b$. It immediately follows that

$$E[B_k] = \frac{2(M_\sigma + 1)}{\sigma} \sum_{i=0}^{k-1} E[\phi_i] \leq \frac{2(M_\sigma + 1)}{\sigma} \sum_{i=0}^{k-1} \bar{R}\bar{\alpha} \leq \frac{2(M_\sigma + 1)\bar{R}}{\sigma} \frac{1}{1 - \bar{\alpha}}.$$

Recalling Lemma 8, we then obtain

$$E[A_k] = \sqrt{2M} E[\sum_{i=0}^{k-1} \sqrt{\phi_i}] \leq \frac{\sqrt{2M\bar{R}}}{1 - \sqrt{\bar{\alpha}}}$$

$$E[A_k^2] = E[(\sum_{i=1}^{k} \sqrt{2M\phi_i})^2] = 2ME[(\sum_{i=0}^{k-1} \sqrt{\phi_i})^2] \leq \frac{4M\bar{R}}{(1 - \sqrt{\bar{\alpha}})^2}$$

$$E[\sqrt{B_k} A_k] = E[\sqrt{\frac{2(M_\sigma + 1)}{\sigma} \sum_{i=1}^{k} \phi_i} \sum_{i=1}^{k} \sqrt{2M\phi_i}] = \sqrt{\frac{4M(M_\sigma + 1)}{\sigma}} E[\sqrt{\sum_{i=1}^{k} \phi_i} \sum_{i=1}^{k} \sqrt{\phi_i}]$$
$$\leq \sqrt{\frac{4M(M_\sigma + 1)}{\sigma}} \frac{\bar{R}}{\sqrt{1 - \bar{\alpha}}(1 - \sqrt{\bar{\alpha}})},$$

and ζ is equal to

$$\zeta = \frac{\bar{R}(M_\sigma + 1)}{1 - \bar{\alpha}} + \frac{\sigma}{2}C + \frac{8M\bar{R}}{(1 - \sqrt{\bar{\alpha}})^2} + \sqrt{C}\frac{\sqrt{2M\bar{R}}}{1 - \sqrt{\bar{\alpha}}} + \sqrt{\frac{4M(M_\sigma + 1)}{\sigma}} \frac{\bar{R}}{\sqrt{1 - \bar{\alpha}}(1 - \sqrt{\bar{\alpha}})}.$$

□

Remark 5. *Note that the choice of the number of coordinates decent steps in each applications of RCD, in other words, the choice of constants a and b has a direct affect on the constants in the complexity bound. Clearly taking more RCD steps leads to fewer iterations of the main algorithm, as larger values of a and b lead to smaller value of ζ. On the other hand, each iterations becomes more expensive. We believe that our practical approach described in Section 5.7.3 strikes a good balance in this trade-off.*

We conclude that under Assumption 2, Algorithm 14 converges at the rate of $O(1/k)$ if $\sum_{i=0}^{k-1} \sqrt{\frac{2\phi_i}{M_i}}$ is bounded for all k. This result in similar to those obtained in [23]. It is shown how randomized block coordinate descent and other methods can be utilized to optimize subproblems $\min_u Q(H_i, u, x^i)$ so that $\sqrt{\frac{2\phi_i}{M_i}}$ decays sufficiently fast to guarantee such a bound (possibly in expectation).

In this paper, however, we focus on a different derivation of the sub linear convergence rate, which results in a more complex dependence on the constants, but, on the other hand, does not require Assumption 2 and results in a weaker assumption on ϕ_i.

4.4 Analysis of sub linear convergence

In our analysis below we will use another known technique for establishing sub linear convergence of gradient descent type methods for smooth convex functions [25]. However, due to the non smooth nature of our function the analysis requires significant extensions, especially in the inexact case. Moreover, it does not apply to the line-search algorithm, we will rely on the fact that a proximal quasi-Newton method is used in that each new iteration x^{k+1} is an approximate minimizer of the function $Q(H_k, u, x^k)$. Our analysis, hence, also applies to proximal gradient methods.

First we prove the following simple result.

Lemma 10. *Consider $F(\cdot)$ defined in (4.1.1). Let Assumptions 1(iii) hold. Then for any three points $u, v, w \in \mathrm{dom}(F)$, we have*

$$F(u) - F(w) \le \|\nabla f(u) + \gamma_{g,\phi}^v\| \|u - w\| + 2L_g \|u - v\| + 2\phi. \tag{4.4.1}$$

where $\gamma_{g,\phi}^v \in \partial_\phi g(v)$ is any ϕ-subgradient of $g(\cdot)$ at point v.

Proof. From convexity of f and g and the definition of ϕ-subgradient, it follows that for any points u, w and v,

$$f(u) - f(w) \leq \langle \nabla f(u), u - w \rangle, \tag{4.4.2}$$

$$g(v) - g(w) \leq \langle \gamma_{g,\phi}^{v}, v - w \rangle + \phi \tag{4.4.3}$$

$$g(2v - u) - g(v) \geq \langle \gamma_{g,\phi}^{v}, v - u \rangle - \phi. \tag{4.4.4}$$

Hence,

$$\begin{aligned} F(u) - F(w) &= f(u) - f(w) + g(u) - g(w) \\ &= f(u) - f(w) + g(u) - g(v) + g(v) - g(w) \\ &\leq \langle \nabla f(u), u - w \rangle + \langle \gamma_{g,\phi}^{v}, v - w \rangle + g(u) - g(v) + \phi \qquad (4.4.5) \\ &= \langle \nabla f(u), u - w \rangle + \langle \gamma_{g,\phi}^{v}, u - w \rangle + \langle \gamma_{g,\phi}^{v}, v - u \rangle + g(u) - g(v) + \phi \\ &\leq \|\nabla f(u) + \gamma_{g,\phi}^{v}\| \|u - w\| + g(2v - u) - g(v) + g(u) - g(v) + 2\phi. \end{aligned}$$

Here we applied (4.4.2), (4.4.3) and (4.4.4) to get (4.4.5). Using Assumption 1(*iii*) to bound the term $g(2v - u) - g(v) + g(u) - g(v)$ we can easily derive (4.4.1). □

4.4.1 The exact case

We now consider the exact version of Algorithm 14, i.e., $\phi_k = 0$ for all k. We have the following lemma.

Lemma 11. *Let* $x^{k+1} := \arg\min_{u \in \mathbb{R}^n} Q(H_k, u, x_k)$, *then*

$$Q(H_k, x^k, x^k) - Q(H_k, x^{k+1}, x^k) \geq \frac{\sigma_k}{2} \|x^{k+1} - x^k\|^2. \tag{4.4.6}$$

Moreover, there exists a vector $\gamma_g^{k+1} \in \partial g(x^{k+1})$ *such that the following bounds hold:*

$$\frac{1}{M_k} \|\nabla f(x^k) + \gamma_g^{k+1}\| \leq \|x^{k+1} - x^k\| \leq \frac{1}{\sigma_k} \|\nabla f(x^k) + \gamma_g^{k+1}\|. \tag{4.4.7}$$

Proof. The proof is a special case of Lemma 13, proved below. □

Theorem 12. *Let Assumptions 1 hold for all* k. *Then the iterates* $\{x^k\}$ *generated by Algorithm 14 satisfy*

$$F(x^k) - F^* \leq \frac{2M^2(D_{\mathcal{X}_0} M + 2L_g)^2}{\rho \sigma^3} \frac{1}{k}. \tag{4.4.8}$$

Proof. We will denote $F(x^k) - F^*$ by ΔF_k. Our goal is to bound ΔF_k from above in terms of $1/k$ which we will achieve by deriving a lower bound on $\frac{1}{\Delta F_k}$ in terms of k.

Let us first show that

$$F(x^k) - F(x^{k+1}) = \Delta F_k - \Delta F_{k+1} \geq c_k \Delta F_k^2, \tag{4.4.9}$$

for some constant c_k, which depends on iteration k, but will be lower bounded by a uniform constant.

First we will show that

$$\Delta F_k \leq (D_{\mathcal{X}_0} + \frac{2L_g}{\sigma_k})\|\nabla f(x^k) + \gamma_g^{k+1}\|. \tag{4.4.10}$$

This follows simply from (4.4.1) with $u = x^k, w = x^*, v = x^{k+1}$ and $\phi = 0$,

$$\Delta F_k = F(x^k) - F(x^*) \leq \|\nabla f(x^k) + \gamma_g^{k+1}\|\|x^k - x^*\| + 2L_g\|x^k - x^{k+1}\|. \tag{4.4.11}$$

Substituting the bounds on $\|x^{k+1} - x^k\|$ (cf. (4.4.7)) and $\|x^k - x^*\|$ (cf. Assumptions 1(v)) in (4.4.11) we get the desired bound (4.4.10).

Now we will show that

$$F(x^k) - F(x^{k+1}) \geq \frac{\rho\sigma_k}{2M_k^2}\|\nabla f(x^k) + \gamma_g^{k+1}\|^2. \tag{4.4.12}$$

Indeed $F(x^k) - F(x^{k+1}) \geq \rho(Q(H_k, x^k, x^k) - Q(H_k, x^{k+1}, x^k))$. And a bound on the reduction in Q can be established by combining (4.4.6) and (4.4.7),

$$Q(H_k, x^k, x^k) - Q(H_k, x^{k+1}, x^k) \geq \frac{\sigma_k}{2M_k^2}\|\nabla f(x^k) + \gamma_g^{k+1}\|^2,$$

Hence, the bound (4.4.12) holds.

Finally, combining the lower bound on $F(x^k) - F(x^{k+1})$ together with the upper bound on ΔF_k^2 we can conclude that

$$F(x^k) - F(x^{k+1}) = \Delta F_k - \Delta F_{k+1} \geq \frac{\rho\sigma_k}{2M_k^2(D_{\mathcal{X}_0} + \frac{2L_g}{\sigma_k})^2}\Delta F_k^2,$$

which establishes (4.4.30) with $c_k = \frac{\rho\sigma_k^3}{2M_k^2(D_{\mathcal{X}_0}\sigma_k+2L_g)^2}$.

Dividing both sides of the inequality above by $\Delta F_{k+1}\Delta F_k$ we have

$$\frac{1}{\Delta F_{k+1}} - \frac{1}{\Delta F_k} \geq c_k \frac{\Delta F_k}{\Delta F_{k+1}} \geq c_k.$$

Summing the above expression for $i = 0, \ldots, k-1$ we have

$$\frac{1}{\Delta F_k} \geq \sum_{i=0}^{k-1} c_i + \frac{1}{\Delta F_0} \geq \sum_{i=0}^{k-1} c_i,$$

which finally implies

$$\Delta F_k = F(x^k) - F^* \leq \frac{1}{\sum_{i=0}^{k=1} c_i} \leq \frac{2M^2(D_{\mathcal{X}_0}M + 2L_g)^2}{\rho\sigma^3}\frac{1}{k}.$$

□

Let us note that if $H_k = L(f)I$ for all k, as in standard proximal gradient methods, where $L(f)$ is the Lipschitz constant of $\nabla f(x)$, then the bound becomes

$$F(x^k) - F^* \leq \frac{2(D_{\mathcal{X}_0}L(f) + 2L_g)^2}{\rho L(f)}\frac{1}{k} \approx \frac{2D_{\mathcal{X}_0}^2 L(f)}{k},$$

if $L_g \ll D_{\mathcal{X}_0}L(f)$. This bound is similar to $\frac{2\|x^0-x^*\|^2 L(f)}{k}$ established for proximal gradient methods, assuming that $D_{\mathcal{X}_0}$ is comparable to $\|x^0 - x^*\|$.

4.4.2 The inexact case

We now analyze Algorithm 14 in the case when the computation of $p_H(v)$ is performed inexactly. In other words, we consider the version of Algorithm 14 (and 15) where we compute $x^{k+1} := p_{H_k,\phi_k}(x^k)$ and ϕ_k can be positive for any k. The analysis is similar to that of the exact case, with a few additional terms that need to be bounded. We begin by extending Lemma 11.

Lemma 13. *Let $x^{k+1} \in \mathrm{dom}(F)$ be such that the inequality holds*

$$Q(H_k, x^{k+1}, x^k) - \min_{y\in\mathbb{R}^n} Q(H_k, y, x^k) \leq \phi_k$$

for some $\phi_k \geq 0$. *Then*

$$Q(H_k, x^k, x^k) - Q(H_k, x^{k+1}, x^k) \geq \frac{\sigma_k}{2}\|x^{k+1} - x^k\|^2 - \sqrt{2M_k\phi_k}\|x^{k+1} - x^k\| - \phi_k. \tag{4.4.13}$$

Moreover there exists a vector $\gamma_{g,\phi}^{k+1} \in \partial g_{\phi_k}(x^{k+1})$ *such that the following bounds hold:*

$$\frac{1}{M_k}\|\nabla f(x^k) + \gamma_{g,\phi}^{k+1}\| - \frac{\sqrt{2M_k\phi_k}}{M_k} \leq \|x^{k+1} - x^k\| \leq \frac{1}{\sigma_k}\|\nabla f(x^k) + \gamma_{g,\phi}^{k+1}\| + \frac{\sqrt{2M_k\phi_k}}{\sigma_k}. \tag{4.4.14}$$

Proof. Recall Lemma 3, from (4.3.2), there exists a vector, which we will refer to as $\gamma_{g,\phi}^{k+1}$, such that

$$\gamma_{g,\phi}^{k+1} = H_k(x^k - x^{k+1}) - \nabla f(x^k) - \eta_k \in \partial_\phi g(x^{k+1}), \tag{4.4.15}$$

with $\frac{1}{2}\|\eta_k\|^2_{H_k^{-1}} \leq \phi_k$, which, in turn, implies $\|\eta_k\| \leq \sqrt{2M_k\phi_k}$. The following inequality follows from the definition of ϕ-subdifferential,

$$g(x^k) - g(x^{k+1}) \geq \langle \gamma_{g,\phi}^{k+1}, x^k - x^{k+1}\rangle - \phi_k. \tag{4.4.16}$$

From (4.4.15)

$$H_k(x^{k+1} - x^k) + \nabla f(x^k) + \gamma_{g,\phi}^{k+1} + \eta_k = 0, \tag{4.4.17}$$

hence we obtain

$$\|x^{k+1} - x^k\| \leq \frac{1}{\sigma_k}\|\nabla f(x^k) + \gamma_{g,\phi}^{k+1}\| + \frac{1}{\sigma_k}\|\eta_k\| \leq \frac{1}{\sigma_k}\|\nabla f(x^k) + \gamma_{g,\phi}^{k+1}\| + \frac{\sqrt{2M_k\phi_k}}{\sigma_k},$$
$$\|x^{k+1} - x^k\| \geq \frac{1}{M_k}\|\nabla f(x^k) + \gamma_{g,\phi}^{k+1}\| - \frac{1}{M_k}\|\eta_k\| \geq \frac{1}{M_k}\|\nabla f(x^k) + \gamma_{g,\phi}^{k+1}\| - \frac{\sqrt{2M_k\phi_k}}{M_k}.$$

Using (4.4.16) and (4.4.17), we derive (4.4.13) as follows,

$$
\begin{aligned}
&Q(H_k, x^k, x^k) - Q(H_k, x^{k+1}, x^k) \\
&= g(x^k) - g(x^{k+1}) - \nabla f(x^k)^T(x^{k+1} - x^k) - \frac{1}{2}\|x^{k+1} - x^k\|_{H_k}^2 \\
&\geq \langle \gamma_{g,\phi}^{k+1}, x^k - x^{k+1}\rangle - \phi_k + \|x^{k+1} - x^k\|_{H_K}^2 \\
&+ \langle \gamma_{g,\phi}^{k+1}, x^{k+1} - x^k\rangle + \langle \eta_k, x^{k+1} - x^k\rangle - \frac{1}{2}\|x^{k+1} - x^k\|_{H_k}^2 \\
&= \frac{1}{2}\|x^{k+1} - x^k\|_{H_k}^2 + \langle \eta_k, x^{k+1} - x^k\rangle - \phi_k \geq \frac{\sigma_k}{2}\|x^{k+1} - x^k\|^2 - \|\eta_k\|\|x^{k+1} - x^k\| - \phi_k \\
&\geq \frac{\sigma_k}{2}\|x^{k+1} - x^k\|^2 - \sqrt{2M_k\phi_k}\|x^{k+1} - x^k\| - \phi_k.
\end{aligned}
$$

□

Unlike the exact case, the inquality $\frac{1}{\Delta F_{k+1}} - \frac{1}{\Delta F_k} \geq c_k$ can no longer be guaranteed to hold on each iteration. The convergence rate is obtained by observing that when this desired inequality fails another inequality always holds, which bounds ΔF_k in terms of ϕ_k. Specifically, we have the following theorem.

Lemma 14. *Consider kth iteration of the inexact Algorithm 14 with* $0 \leq \phi_k \leq 1$. *Let* $\Delta F_k := F(x^k) - F(x^*)$. *Then there exists large enough positive constant* $\theta > 0$, *such that one of the following two cases must hold,*

$$\Delta F_k \leq b_k\sqrt{\phi_k}, \tag{4.4.18}$$

$$\frac{1}{\Delta F_{k+1}} - \frac{1}{\Delta F_k} \geq c_k. \tag{4.4.19}$$

where b_k *and* c_k *are given below,*

$$b_k = \theta D_{\mathcal{X}_0}\sqrt{2M_k} + \frac{2(1+\theta)L_g}{\sigma_k}\sqrt{2M_k} + 2, \tag{4.4.20}$$

$$c_k = \frac{\rho(\sigma_k^3(\theta-1)^2 - 2\sigma_k M_k^2(1+\theta) - \sigma_k^2 M_k)}{(\sqrt{2}D_{\mathcal{X}_0}\theta\sigma_k M_k + 2\sqrt{2}L_g(1+\theta)M_k + \sigma_k\sqrt{M_k})^2}. \tag{4.4.21}$$

Proof. First, applying (4.4.1) with $u = x^k, w = x^*$ and $v = x^{k+1}$ we obtain,

$$\Delta F_k = F(x^k) - F(x^*) \leq \|\nabla f(x^k) + \gamma_{g,\phi}^{k+1}\|\|x^k - x^*\| + 2L_g\|x^k - x^{k+1}\| + 2\phi_k. \tag{4.4.22}$$

We will consider two cases that are possible at each iteration k for some fixed constant $\theta > 1$ which we will specify later.

- Case 1
$$\|\nabla f(x^k) + \gamma_{g,\phi}^{k+1}\| < \theta\sqrt{2M_k\phi_k}. \tag{4.4.23}$$

- Case 2
$$\|\nabla f(x^k) + \gamma_{g,\phi}^{k+1}\| \geq \theta\sqrt{2M_k\phi_k}. \tag{4.4.24}$$

Let us assume that Case 1 holds, then from (4.4.14) and (4.4.23) it simply follows that
$$\|x^{k+1} - x^k\| \leq \frac{(1+\theta)\sqrt{2M_k\phi_k}}{\sigma_k}. \tag{4.4.25}$$

Using (4.4.23), (4.4.22), the bound on $\|x^{k+1} - x^k\|$ from (4.4.25) together with the bound on $\|x^k - x^*\|$ from Assumptions 1(v) we get
$$\begin{aligned}
\Delta F_k &\leq D_{\mathcal{X}_0}\|\nabla f(x^k) + \gamma_{g,\phi}^{k+1}\| + \frac{2(1+\theta)L_g\sqrt{2M_k\phi_k}}{\sigma_k} + 2\phi_k \quad (4.4.26)\\
&\leq (\theta D_{\mathcal{X}_0} + \frac{2(1+\theta)L_g}{\sigma_k})\sqrt{2M_k\phi_k} + 2\phi_k\\
&\leq (\theta D_{\mathcal{X}_0}\sqrt{2M_k} + \frac{2(1+\theta)L_g}{\sigma_k}\sqrt{2M_k} + 2)\sqrt{\phi_k}, \quad (4.4.27)
\end{aligned}$$
which ensures (4.4.18).

We now consider Case 2, where (4.4.24) along with (4.4.14) from Lemma 13 imply
$$\frac{\theta - 1}{\theta M_k}\|\nabla f(x^k) + \gamma_{g,\phi}^{k+1}\| \leq \|x^{k+1} - x^k\| \leq \frac{1+\theta}{\theta\sigma_k}\|\nabla f(x^k) + \gamma_{g,\phi}^{k+1}\|. \tag{4.4.28}$$

Substituting into (4.4.22) the upper bound on $\|x^{k+1} - x^k\|$ from (4.4.28) and the bound on $\|x^k - x^*\|$ from Assumptions 1(v) we now get
$$\Delta F_k \leq (D_{\mathcal{X}_0} + \frac{2L_g(1+\theta)}{\theta\sigma_k})\|\nabla f(x^k) + \gamma_{g,\phi}^{k+1}\| + 2\phi_k.$$

From $\phi_k \leq 1$ it follows that $\|\nabla f(x^k) + \gamma_{g,\phi}^{k+1}\| \geq \theta\sqrt{2M_k\phi_k} \geq \theta\sqrt{2M_k}\phi_k$. Hence we obtain
$$\Delta F_k \leq (D_{\mathcal{X}_0} + \frac{2L_g(1+\theta)}{\theta\sigma_k} + \frac{2}{\theta\sqrt{2M_k}})\|\nabla f(x^k) + \gamma_{g,\phi}^{k+1}\|. \tag{4.4.29}$$

We will show that in this case, as in the exact case, we have

$$\Delta F_k - \Delta F_{k+1} = F(x^k) - F(x^{k+1}) \geq c_k \Delta F_k^2 \tag{4.4.30}$$

for some constant c_k (different from that in the exact case). Towards that goal we will establish a lower bound on $F(x^k) - F(x^{k+1})$ in terms of $\|\nabla f(x^k) + \gamma_{g,\phi}^{k+1}\|^2$ as before. We still have $F(x^k) - F(x^{k+1}) \geq \rho(Q(H_k, x^k, x^k) - Q(H_k, x^{k+1}, x^k))$. We now use the bounds (4.4.13) from Lemma 13, (4.4.24) and (4.4.28) and obtain

$$\begin{aligned}&Q(H_k, x^k, x^k) - Q(H_k, x^{k+1}, x^k)\\ &\geq \frac{\sigma_k}{2}(\frac{\theta-1}{\theta M_k})^2\|\nabla f(x^k) + \gamma_{g,\phi}^{k+1}\|^2 - \frac{1+\theta}{\theta^2\sigma_k}\|\nabla f(x^k) + \gamma_{g,\phi}^{k+1}\|^2 - \frac{1}{2\theta^2 M_k}\|\nabla f(x^k) + \gamma_{g,\phi}^{k+1}\|^2\end{aligned}$$

By selecting a sufficiently large θ we can ensure that

$$F(x^k) - F(x^{k+1}) \geq \rho t_k \|\nabla f(x^k) + \gamma_{g,\phi}^{k+1}\|^2, \tag{4.4.31}$$

for $t_k = \frac{\sigma_k}{2}(\frac{\theta-1}{\theta M_k})^2 - \frac{1+\theta}{\theta^2\sigma_k} - \frac{1}{2\theta^2 M_k} > 0$. Finally, combining the lower bound (4.4.31) on $F(x^k) - F(x^{k+1})$ together with the upper bound (4.4.29) on ΔF_k^2 we can conclude that (4.4.30) holds with

$$c_k = \frac{\rho t_k}{(D_{\mathcal{X}_0} + \frac{2L_g(1+\theta)}{\theta\sigma_k} + \frac{2}{\theta\sqrt{2M_k}})^2} = \frac{\rho(\sigma_k^3(\theta-1)^2 - 2\sigma_k M_k^2(1+\theta) - \sigma_k^2 M_k)}{(\sqrt{2}D_{\mathcal{X}_0}\theta\sigma_k M_k + 2\sqrt{2}L_g(1+\theta)M_k + 2\sigma_k\sqrt{M_k})^2}.$$

Finally, (4.4.19) follows from (4.4.30) divided by $\Delta F_k \Delta F_{k+1}$ and using the fact that $\frac{\Delta F_k}{\Delta F_{k+1}} \geq 1$.
□

Remark 6. *Let us discuss the result of the above lemma. The lemma applies for any value of θ for which t_k, and hence, c_k is positive for all k. It is easy to see that large values of θ imply potentially larger values of c_k. On the other hand, large θ is likely to cause Case 1 to hold (i.e.,* (4.4.18)*) instead of Case 2 (i.e.,* (4.4.19)*) on any given iteration, with b_k also growing with the size of θ. As we will show below the overall rate of convergence of the algorithm is derived using the two bounds -* (4.4.18)*, where the rate is controlled by the rate of $\phi_k \to 0$ and* (4.4.19)*, which is* *similar to the bound in the exact case. The overall bound, thus, will depend on the upper bound on b_k's and the inverse of the lower bound on c_k's. If, again, we assume that $\sigma_k = M_k = L(f)$ for all k, then $\theta \geq 5$ is sufficient to ensure that $c_k > 0$ and this results in $b_k \leq O(D_{\mathcal{X}_0} L(f))$ and $1/c_k \geq O(D^2_{\mathcal{X}_0} L(f))$, thus again, we obtain a bound which is comparable to that of proximal gradient methods, although with more complex constants.*

We now derive the overall convergence rate, under the assumption that ϕ_k decays sufficiently fast.

Theorem 15. *Suppose that Assumption 1 holds. Assume that all iterates $\{x^k\}$ of inexact Algorithm 14 are generated with some $\phi_k \geq 0$ that satisfy*

$$\phi_k \leq \frac{a^2}{k^2}, \text{ with } 0 < a \leq 1. \tag{4.4.32}$$

Let θ be chosen as specified in Lemma 14. Then for any k

$$F(x^k) - F(x^*) \leq \frac{\max\{ba, \frac{1}{c}\}}{k-1} \tag{4.4.33}$$

with b, c given as follows,

$$b = \theta D_{\mathcal{X}_0}\sqrt{2M} + \frac{2(1+\theta)L_g}{\sigma}\sqrt{2M} + 2,\ c = \frac{\rho\left(\frac{\sigma^2}{M^2}(1-\theta^{-1})^2 - (2\theta^{-1} + 3\theta^{-2})\right)}{2(D_{\mathcal{X}_0} + \frac{2L_g(1+\theta^{-1})}{\sqrt{\sigma}} + \frac{2}{\sqrt{2}}\theta^{-1})^2}. \tag{4.4.34}$$

Proof. Consider all iterations before a particular iteration k. From Lemma 14, it follows that either (4.4.18) or (4.4.19) must hold for each prior iteration. Let $k_1 < k$ denote the index of the last iteration before k, for which (4.4.18) holds. If no such k_1 exists, then (4.4.19) holds for all k and without loss of generality we can consider $k_1 = 0$. From (4.4.18) and from the fact that the function value never increases

$$\Delta F_{k_1+1} = F(x^{k_1+1}) - F(x^*) \leq F(x^{k_1}) - F(x^*) = \Delta F_{k_1} \leq b_{k_1}\sqrt{\phi_{k_1}} \leq \frac{ab_{k_1}}{k_1} \tag{4.4.35}$$

which is the same as

$$\frac{1}{\Delta F_{k_1+1}} \geq \frac{k_1}{ab_{k_1}}. \tag{4.4.36}$$

For each iteration from $k_1 + 1$ to $k - 1$, (4.4.19) gives

$$\frac{1}{\Delta F_{k_1+i+1}} - \frac{1}{\Delta F_{k_1+i}} \geq c_i, \ \forall i = 1, ..., k - k_1 - 1$$

Summing up the above inequalities and using (4.4.36) we obtain the following bound on ΔF_k,

$$\frac{1}{\Delta F_k} \geq \sum_{i=1}^{k-k_1-1} c_i + \frac{1}{\Delta F_{k_1+1}} \geq \sum_{i=1}^{k-k_1-1} c_i + \frac{k_1}{ab_{k_1}}. \tag{4.4.37}$$

To derive a simple uniform bound on ΔF_k we will use b and c - uniform upper and lower bounds, respectively, for b_k and c_k given in (4.4.20), i.e., $b_k \leq b$, $c_k \geq c, \forall k \geq k_0$. From Assumptions 1$(iv)$ we can derive the expressions for c as follows,

$$\begin{aligned} c_k &= \frac{\rho\left(\frac{\sigma_k}{2}(\frac{\theta-1}{\theta M_k})^2 - \frac{1+\theta}{\theta^2\sigma_k} - \frac{1}{2\theta^2 M_k}\right)}{(D_{\mathcal{X}_0} + \frac{2L_g(1+\theta)}{\theta\sigma_k} + \frac{2}{\theta\sqrt{2M_k}})^2} \geq \frac{\rho\left(\frac{\sigma}{2}(\frac{\theta-1}{\theta M})^2 - \frac{1+\theta}{\theta^2\sigma} - \frac{1}{2\theta^2\sigma}\right)}{(D_{\mathcal{X}_0} + \frac{2L_g(1+\theta)}{\theta\sigma} + \frac{2}{\theta\sqrt{2\sigma}})^2} \\ &\geq \frac{\rho\left(\frac{\sigma^2}{2M^2}(\frac{\theta-1}{\theta})^2 - \frac{1+\theta}{\theta^2} - \frac{1}{2\theta^2}\right)}{(D_{\mathcal{X}_0} + \frac{2L_g(1+\theta)}{\theta\sqrt{\sigma}} + \frac{2}{\theta\sqrt{2}})^2} \end{aligned}$$

Bound b can be obtained in a similar fashion.

Substituting bounds b and c in (4.5.5) we get

$$\frac{1}{\Delta F_k} \geq (k - k_1 - 1)c + \frac{k_1}{ab} \geq \min\{c, \frac{1}{ab}\}(k-1),$$

which is the same as (4.5.2).

□

Remark 7. *It follows that the inexact version of Algorithm 14 has sublinear convergence rate if* $\phi_i \leq a^2/i^2$ *for some* $a < 1$ *and all iterations* $i = 0, \ldots, k$*. In contrast, the bounds in [23] and in Section 4.3.1 require that* $\sum_{i=0}^{\infty} \sqrt{\phi_i}$ *is bounded. This bound* *on the overall sequence is clearly tighter than $\phi_i \leq a^2/i^2$, since $\sum_{i=0}^{\infty} \frac{a}{i} = \infty$. On the other hand, it does not impose any particular requirement on any given iteration, except that each ϕ_i is finite, which our bound on ϕ_i is assumed to hold at each iteration, so far. In Section 4.5 we will relax this assumption.*

4.4.3 Complexity in terms of subproblem solver iterations

Let us discuss conditions on the sequence $\{\phi_i\}$ established above and how they can be ensured. Firstly, let us note that condition $a < 1$ in Theorem 15 can easily be removed. We introduced it for the sake of brevity, to ensure that $\phi_i \leq 1$ on each iteration. Clearly, an arbitrarily large a can be used and in that case $\phi_i \leq 1$ for all $i \geq a$. Moreover, the condition $\phi_i \leq 1$ is only needed to replace ϕ_i with $\sqrt{\phi_i}$ in Lemma 14 in inequality (4.4.26). Instead we can use bound (4.4.24) and upper bounds on $\nabla f(x^i)$ and $\gamma_{g,\phi}^{i+1}$ to replace ϕ_k with a constant multiple of $\sqrt{\phi_i}$. In conclusion, it is sufficient to solve the subproblem on the i-th iteration to accuracy $O(1/i^2)$.

The question now is: what method and what stopping criterion should be used for subproblem optimization, so that sufficient accuracy is achieved and no excessive computations are performed, in other word, how can we guarantee the bound on ϕ_i, while maintaining efficiency of the subproblem optimization? It is possible to consider terminating the optimization of the i-th subproblem once the duality gap is smaller than the required bound on ϕ_i. However, checking duality gap can be computationally very expensive. Alternatively one can use an algorithm with a known convergence rate. This way it can be determined apriori how many iterations of such an algorithm should be applied to the i-th subproblem to achieve the desired accuracy. In particular, we note that the objective functions in our subproblems are all σ-strongly convex, so a simple proximal gradient method, or some of its accelerated versions, will enjoy linear convergence rates when applied to these subproblems. Hence, after i iterations of optimizing Q_i, such a method will achieve accuracy ϕ_i that decays geometrically, i.e., $\phi_i = C\delta^i$, for some constants $C > 0$ and $0 < \delta < 1$, hence $\sum_{i=0}^{\infty} \sqrt{\phi_i}$ is bounded. Note that the same property holds for any linearly convergent method, such as the proximal gradient or a semi-smooth Newton method. Also, it is easy to see that $\phi_i \leq a^2/i^2$ holds for some $a > 0$ for all i. One can also use FISTA [2] to optimize Q_i which will ensure $\phi_i \leq a^2/i^2$ for some $a > 0$ but will not guarantee $\sum_{i=0}^{\infty} \sqrt{\phi_i}$. The advantage of using FISTA and its resulting rate is that it does not depend on the strong convexity constant, hence the subproblem complexity does not depend on σ - the lower bound on the smallest eigenvalues of the Hessian approximations. In conclusion, we have the following overall bounds.

Theorem 16. *Suppose that Assumptions 1 hold and that for all k, at the k-th iteration of inexact Algorithm 14 function $Q(H_k, u, x^k)$ is approximately minimized, to obtain x^{k+1} by applying $l(k) = \alpha k + \beta$ steps of any algorithm which guarantees that $Q(H_k, x^{k+1}, x^k) \leq Q(H_k, x^k, x^k)$ and whose convergence rate ensures the error bound $\phi_k \leq a^2/(\alpha k + \beta)^2$ for some $a > 0$. Then accuracy $F(x^t) - F(x^*) \leq \epsilon$ is achieved after at most*

$$K = \beta t + \frac{\alpha}{2}(t-1)t \tag{4.4.38}$$

inner iterations (of the chosen algorithm), with $t = \lceil \frac{\max\{ba, \frac{1}{c}\}}{\epsilon} + 1 \rceil$ and b, c given in Theorem 15.

Proof. A proof follows trivially from Theorem 15.

Theorem 17. *Suppose that Assumptions 1 hold and that or all k at the k-th iteration of inexact Algorithm 14 function $Q(H_k, u, x^k)$ is approximately minimized, to obtain x^{k+1} by applying $l(k)$ steps of an algorithm, which guarantees that $Q(H_k, x^{k+1}, x^k) \leq Q(H_k, x^k, x^k)$ and whose convergence rate ensures the error bound $\phi_k \leq \delta^{l(k)} M_Q$, for some constants $0 < \delta < 1$ and $M_Q > 0$. Then, by setting $l(k) = 2log_{\frac{1}{\delta}}(k)$, accuracy $F(x^t) - F(x^*) \leq \epsilon$ is achieved after at most*

$$K = \sum_{k=0}^{t} 2\log_{\frac{1}{\delta}}(k) \leq 2t\log_{\frac{1}{\delta}}(t) \tag{4.4.39}$$

inner iterations (of the chosen algorithm), with $t = \lceil \frac{\max\{ba, \frac{1}{c}\}}{\epsilon} + 1 \rceil$ and b, c given in Theorem 15.

Proof. A proof follows trivially from Theorem 15.

Remark 8. *The total complexity in terms of the inner iterations should not be viewed as a summary of the whole complexity of Algorithm 14. A key step of the algorithm is the computation of $F(x^k)$ and $\nabla f(x^k)$ at each iteration. In big data applications this is often the most extensive step, hence the main complexity is defined by the number of function and gradient computations. Due to backtracking via proximal parameter update to satisfy sufficient decrease condition, the number of function and gradient computation may be larger than the number of iterations of Algorithm 14, however, it does not exceed this number by more than a logarithmic factor. In practice, only several initial iterations contain backtracking steps, hence Theorem 15 provides the bound on the complexity in terms of function and gradient computations.*

In the next section we extend our convergence rate results to the case of solving subproblems via randomized coordinate descent, where ϕ is random and hence does not satisfy required bounds on each iteration.

4.5 Analysis of the inexact case under random subproblem accuracy

As we pointed out in the introduction, the most efficient practical approach to subproblem optimization, in the case when $g(x) = \lambda\|x\|_1$, seems to be the coordinate descent method. One iteration of a coordinate descent step can be a lot less expensive than that of a proximal gradient or a Newton method. In particular, if matrix H is constructed via the LBFGS approach, then one step of a coordinate decent method takes a constant number of operations, m (the memory size of LBFGS, which is typically 10-20). On the other hand, one step of proximal gradient takes $O(mn)$ operations and Newton method takes $O(nm^2)$.

Unfortunately, cyclic (Gauss-Seidel) coordinate descent, does not have deterministic complexity bounds, hence it is not possible to know when the work on a particular subproblem can be terminated to guarantee the desired level of accuracy. However, a randomized coordinate descent has probabilistic complexity bounds, which can be used to demonstrate the linear rate of convergence in expectation.

We have the following probabilistic extension of Theorem 15.

Theorem 18. *Suppose that Assumption 1 holds. Assume that for all k iterates $\{x^k\}$ of inexact Algorithm 14 are generated with some $\phi_k \geq 0$ that satisfy*

$$P\{\phi_k \leq \frac{a^2}{k^2}\} \geq 1-p, \text{ for some } 0 < a \leq 1 \text{ and } 0 \leq p < 1, \tag{4.5.1}$$

conditioned on the past. Let θ, b and c be as specified in Theorem 15. Then for any k

$$E(F(x^k) - F(x^*)) \leq \frac{\max\{ba, \frac{1}{c}\}(2-p)}{(1-p)(k-1)}. \tag{4.5.2}$$

Proof. As in the proof of Theorem 15 consider all iterations before a particular iteration k. From Lemma 14, it follows that either (4.4.18) or (4.4.19) must hold for each prior iteration. Let $k_1 < k$ denote the index of the last iteration before k, for which (4.4.18) holds. Let k_2 denote the index of the second to last such iteration and so on, hence k_i is the index of the last iteration such that there are exactly i iterations between k_i and $k-1$ for which (4.4.18) holds. Without loss of generality we can assume that k_i exists for each i, because if it does not - we can set $k_i = 0$ and obtain a better bound. Let us now assume for a given i that $\phi_{k_i} \leq \frac{a^2}{{k_i}^2}$ holds, but that $\phi_{k_j} > \frac{a^2}{{k_j}^2}$ for all $j = 1, \ldots, i-1$ (if $i = 1$ we have the case analyzed in the proof of Theorem 15). The analysis in the proof of Theorem 15 extends easily to the case $i > 1$ by observing that

$$\frac{1}{\Delta F_{l+1}} - \frac{1}{\Delta F_l} \geq c_l$$

holds for any $k_i + 1 \leq l \leq k-1$, $l \neq k_j$, $j = 1 \ldots, i-1$, that is all the iterations for which or (4.4.18) does not hold. We also have

$$\frac{1}{\Delta F_{l+1}} - \frac{1}{\Delta F_l} \geq 0,$$

for $k_i+1 \leq l \leq k-1$, $l = k_j$, $j = 1\ldots, i-1$, simply from the fact that function values never increase. Summing up the above inequalities and using the fact that $\phi_{k_i} \leq \frac{a^2}{{k_i}^2}$ we obtain ΔF_k,

$$\frac{1}{\Delta F_k} \geq \sum_{l=1}^{k-k_i-i} c_l + \frac{k_l}{ab_{k_l}} \geq (k-k_i-i)c + \frac{k_i}{ab}, \tag{4.5.3}$$

and finally, we have

$$\Delta F_k \leq \frac{\max\{ba, \frac{1}{c}\}}{k-i}. \tag{4.5.4}$$

Now, recall that $P\{\phi_{k_i} \leq \frac{a^2}{{k_i}^2}\} \geq 1-p$, for any iteration i, independently of the other iteration. This means that the probability that $\{\phi_{k_i} \leq \frac{a^2}{{k_i}^2}\}$ and $\{\phi_{k_j} > \frac{a^2}{{k_j}^2}\}$ for all $j = 1\ldots, i-1$ is $(1-p)p^i$. This implies

$$E(\Delta F_k) \leq \sum_{i=1}^{k-1} \frac{\max\{ba, \frac{1}{c}\}}{k-i}(1-p)p^{i-1} \leq \frac{\max\{ba, \frac{1}{c}\}(1-p)}{k-1} \sum_{i=1}^{k-1}(p^{i-1} + \frac{i-1}{k-i}p^{i-1}). \tag{4.5.5}$$

To bound the term $\sum_{i=1}^{k-1}(p^{i-1} + \frac{i-1}{k-i}p^{i-1})$ observe that $\frac{i-1}{k-i}p^{i-1} \leq (i-1)p^{i-1}$ and hence

$$\sum_{i=1}^{k-1}(p^{i-1} + \frac{i-1}{k-i}p^{i-1}) \leq \frac{1}{(1-p)} + \frac{1}{(1-p)^2} = \frac{2-p}{(1-p)^2},$$

which gives us the final bound on the expected error. □

We note that the (2-p) factor is the result of an overestimate of the weighted geometric series and a tighter bound should be possible to obtain.

Below we show that randomized coordinate descent can guarantee sufficient accuracy for subproblem solutions and hence maintain the sub linear convergence rate (in expectation) of Algorithm 14. Moreover, we show in Section 5.7.3, that the randomized coordinate descent is as efficient in practice as the cyclic one.

4.5.1 Analysis of Subproblem Optimization via Randomized Coordinate Descent

In randomized coordinate descent the model function $Q(\cdot)$ is iteratively minimized over one randomly chosen coordinate, while the others remain fixed. The method is presented in Algorithm 16 and is applied for l steps, with l being an input parameter.

Algorithm 16: Randomized Coordinate Descent for optimizing Model Function $Q(H, v, x)$ over v: *RCD* $(Q(H, v, x), x, l)$

```
1 Set $p(x) \leftarrow x$;
2 for $i = 1, 2, \cdots, l$ do
3     Choose $j \in \{1, 2, ..., n\}$ with probability $\frac{1}{n}$;
4     $z^* = \arg\min_z Q(H, p(x) + ze_j, x)$;
5     $p(x) \leftarrow p(x) + z^* e_j$;
6 Return $p(x)$.
```

Here we will show how properties of coordinate descent can be used together with the analysis in Section 4.4. Combination of coordinate descent with the weaker analysis presented can be found in Section 4.3.1.

Our analysis is based on Richtarik and Takac's results on iteration complexity of randomized coordinate descent [31]. In particular, we make use of Theorem 7 in [31], which we restate below without proof, while adapting it to our context.

Lemma 19. *Let v be the initial point and $Q^* := \min_{u \in \mathbb{R}^n} Q(H, u, v)$. If v_l is the random point generated by applying l randomized coordinate descent steps to a strongly convex function Q, then for some constant we have*

$$P\{Q(H, v_l, v) - Q^* \geq \phi\} \leq p,$$

as long as

$$i \geq n(1 + \mu(H)) \log(\frac{Q(H, v, v) - Q^*}{\phi p}),$$

where $\mu(H)$ is a constant that measures conditioning of H along the coordinate directions and in the worst case is at most M/σ - the condition number of H.

Let us now state the version of Algorithms 14 and 15 which is close to what is implemented in practice and for which the complexity bound can be applied.

Algorithm 17: Proximal Quasi-Newton method using randomized coordinate descent

```
1 Choose $0 < \rho \leq 1$, $a, b > 0$ and $x^0$;
2 for $k = 0, 1, 2, \cdots$ do
3     Choose $0 < \bar{\mu}_k, \theta_k > 0, G_k \succeq 0$.;
4     Find $H_k = G_k + \frac{1}{2\mu_k} I$ and $x^{k+1} := p_{H_k,\phi_k}(x^k)$
5     by applying Prox Parameter Update with RCD $(\bar{\mu}_k, G_k, x^k, \rho, a, b)$.;
```

Algorithm 18: Prox Parameter Update with RCD $(\bar{\mu}, G, x, \rho, a, b)$

```
1 Select $0 < \beta < 1$ and set $\mu = \bar{\mu}$;
2 for $i = 1, 2, \cdots$ do
3     Define $H = G + \frac{1}{2\mu} I$, and compute $p(x) := p_{H,\phi}(x)$
4     by applying RCD $(Q(H, v, x), x, l(k))$;
5     If $F(p(x)) - F(x) \leq \rho(Q(H, p(x), x) - F(x))$, then output $H$ and $p(x)$,
      Exit ;
6     Else $\mu = \beta^i \bar{\mu}$;
```

The conclusion of Lemma 19 in application to Algorithms 16-18 is that when applying $l(k)$ steps of randomized coordinate descent approximately to optimize $Q(H_k, u, x^k)$, $\phi_k \leq M_Q \delta^{l(k)}$, with probability p, where M_Q is an upper bound on $\frac{Q(H_k,x^k,x^k) - Q(H_k,p_H(x^k),x^k)}{p}$ and $\delta = e^{-\frac{1}{n(1+\mu(H_k))}}$. This, together with Theorem 18 implies that to solve subproblem on iteration k it is sufficient to set $l(k) = O(n(1 + \mu(H_k)) \log(kp/M_Q))$ and the resulting convergence rate will then obey Theorems 15 and 17. However, it is necessary to know constants M_Q and $\mu(H)$ to be able to construct efficient expression $l(k)$. In practice, a successful strategy is to select a slow growing linear function of k, $l(k) = ak + b$. This certainly guarantees convergence rate of the outer iteration as in Theorem 18. In terms of overall rate this gives inferior complexity, however, we believe that the real difference in terms of the workload appears only in the limit, while in most cases the algorithm successfully terminated before our practical formula $l(k) = ak + b$, described in the next section, significantly exceeds, the theoretical bound $O(n(1 + \mu(H))log(kp/M_Q))$ with appropriate constants. Moreover, as noted earlier, the number of function and gradient evaluations may be the dominant complexity in the big data cases, hence it may be worthwhile to increase workload of coordinate descent in order to reduce the constants in the bound in Theorem 18.

Finally, we note that when using ISTA method for the subproblem, instead of randomized coordinate descent, the number of inner iteration does not need to depend on the dimension n. However, it depends on σ_k/M_k and the cost per iteration is roughly n times bigger than that of coordinate descent (with LBFGS matrices). Hence the overall complexity of using coordinate descent is better than that of ISTA if $\mu(H_k) \ll \sigma_k/M_k$. This indeed happens often in practical problems, as is discussed in [31] and other works on coordinate descent.

Chapter 5

Adaptive Active-set in Proximal Quasi-Newton Method for Large Scale Sparse Optimization

5.1 Introduction

We consider convex sparse unconstrained minimization problem of the following general form

$$\min_{x} \; F(x) = \lambda\|x\|_1 + L(x) \tag{5.1.1}$$

where $L : \mathbb{R}^p \to \mathbb{R}$ is convex and twice differentiable and $\lambda > 0$ is the regularization parameter that controls the sparsity of x. More generally, the regularization term $\lambda\|x\|_1$ can be extended to $\|\lambda \circ x\|_1 = \sum_{i=1}^{p} \lambda_i |x_i|$ to allow for different regularization weights on different entries, e.g., when there is a certain sparsity pattern desired in the solution x. We will focus on the simpler form as in (5.1.1) in this work for the sake of simplicity of presentation, as the extension to the general form is straightforward.

To keep the current article short and focused, we leave the theoretical analysis of a general inexact quasi-Newton proximal scheme out of this chapter. This chapter is organized as follows. In Section 5.2 we overview the algorithm and define basic terminologies. Section 5.3 we introduce compact LBFGS and its low-rank structure, paying particular attention to the mutable matrix type. Section 5.4 discusses active-set strategies and the convergence guarantee. Section 5.6 discusses different termination conditions for the inner loop and their implications on the global convergence performance. In Section 5.7 we present numerical comparisons of different active-set methods and termination conditions on algorithm running time.

5.2 The proposed algorithm

Two-loop scheme. At k-th iteration of the algorithm, the descent step is computed from minimizing a quadratic function $Q_{H_k,x^k}(\cdot)$ that serves as an approximation to (5.1.1) around the current iterate x^k,

$$Q_{H_k,x^k}(d) := L(x^k + d) + \langle \nabla L(x^k), d\rangle + \frac{1}{2}\langle d, H_k d\rangle + \lambda \|x^k + d\|_1.$$

Solving the *subproblem*, or the minimization of Q, with a general matrix H can sometimes be as hard as solving the original problem (5.1.1) [76], and often requires taking iterative steps, which we denote as *inner iterations*.

Low-rank structure. The difficulty of the subproblem can vary a lot with different choice of H. Sometimes it is as easy as computing a closed-form solution, e.g., with H to be a multiple of identity matrix. But such Q does not necessarily serve as a good approximation to (5.1.1), hence may not result in a good step for reducing (5.1.1). In this work, we propose to use a compact representation of L-BFGS as the matrix H, which, as we will see, strike a balance between the easiness of minimizing Q and the goodness of the approximation it provides. Moreover, such H possesses a nice low-rank structure, which can be exploited to facilitates the minimization of Q.

Self-adaptive active-set selection. At every outer iteration, we divide the variables into two sets, the *free* set $\mathcal{I}$ and the *active* set $\mathcal{A}$. We only update those belonging in the free set during inner iterations, i.e., Q is minimized in a subspace formed by those variables in the free set,

$$\hat{d}^k := \arg\min_d \ \{Q_{H_k,x^k}(d), \text{s.t.}\, d_i = 0 \ \forall i \in \mathcal{A}_k\}. \tag{5.2.1}$$

This allows us to construct a smaller H matrix, solve a smaller subproblem and in some cases optimize the memory usage to be more cache-friendly since everything now is small enough to fit within a cache line. We use *active-set* method to denote the strategy of constructing the two sets. For problem (5.1.1), the active set is often selected such that it approximates the set of zero entries in the optimal solution x^*. For example, if this set is known in advance, then the optimal solution of (5.1.1) can be obtained by only minimizing over the variables not in the free set while fixing those in the active set to zero.

Inexactness. The definition in (5.2.1) implies that $\hat{d}^k$ is the exact minimizer of the subproblem. In practice, however, the subproblems are almost never optimized exactly (and rightfully so), unless in cases when H is a multiple of diagonal so that closed-form solutions exist. Hence we also define the inexact minimizer, $\hat{d}^k_\phi$, where the inexactness is captured by a nonnegative scalar $\phi_k \geq 0$,

$$Q_{H_k,x^k}(\hat{d}^k_\phi) \leq Q_{H_k,x^k}(\hat{d}^k) + \phi_k \text{ and } Q_{H_k,x^k}(\hat{d}^k_\phi) \leq Q_{H_k,x^k}(0). \tag{5.2.2}$$

We use *inexact* conditions to denote the precision the subproblems need to be minimized to. As in other classic optimization algorithms, e.g., inexact Newton methods, inexact conditions plays an important role both in convergence rate analysis and in practical performance, e.g., running time, number of function and gradient evaluations. A common intuition demands that subproblems be solved more accurately as the objective approaches optimality. In this work we will describe a particular inexact conditions we propose on ϕ, the simple stopping criteria that follows and its effect on the algorithm performance.

Globalization. Finally, to drive global convergence, the following sufficient decrease condition, a relaxation of what is used in [2] and [23], is validated on every (inexact) minimizer of the subproblem

$$F(x^k + d^k) - F(x^k) \leq \rho(Q_{H_k,x^k}(d^k) - F(x^k)) \tag{5.2.3}$$

where ρ is a constant between 0 and 1. We use d^k to denote the $\hat{d}^k_\phi$ that satisfies the condition, and we update the iterate by $x^k \leftarrow x^k + d^k$. In particular, we let H be parameterized by μ,

$$H_k \leftarrow \frac{\mu}{2} + B_k$$

Let μ iterate through sequence $\{\mu_0, \mu_0\beta, \mu_0\beta^2, ...\}$ with β as a constant larger than 1. As μ becomes larger, we have a slightly different subproblem with a larger strongly convex parameter, and the step $\hat{d}$ computed from minimizing the subproblem will eventually be a descent direction for the objective. Once we find the smallest μ that produces such $\hat{d}^k_\phi$ satisfying the sufficient decrease condition, we return from the procedure with $d^k \leftarrow \hat{d}^k_\phi$.

The general framework described above is summarized in Algorithm 19. Next we discuss how to construct matrix B_k, the active set $\mathcal{A}_k$, the algorithm for minimizing Q and the criteria for terminating the inner iteration.

Algorithm 19: LHAC for sparse optimization

```
1 Input: $x^0, \mu_0, \beta$
2 for $k = 1, 2, ...$ do // outer loop
3     compute $\nabla L(x^k), B_k, \mathcal{A}_k$
4     $\mu \leftarrow \mu_0/\beta, H_k \leftarrow B_k, d^k \leftarrow 0$
5     while condition (5.2.3) is not satisfied do
6         $\mu \leftarrow \mu\beta, H_k \leftarrow \frac{\mu}{2}I + B_k$
7         for $i = 1, 2, ...$ do // inner loop
8             update $d^k$ towards the minimizer of $Q_{H_k,x^k}(\cdot)$
9     $x^{k+1} \leftarrow x^k + d^k$
```

5.3 Mutable matrix type in low-rank hessian approximation

We make use of a specific form of the low-rank Hessian estimates, i.e., $B_k = B_0 - GRG^T$. The matrices B_0, G, R are defined in Theorem 20, which, adopted from [21], shows that B_k is a compact representation of L-BFGS method. In Section 5.5 we will see how to exploit this structure in coordinate descent to facilitate the subproblem minimization. In this section we will focus on discussing a mutable matrix type we implement which is specifically designed for efficiently and repeatedly mutating matrices without consuming extra memory. This new data type will be particularly useful for implementing coordinate descent on the low-rank Hessian approximation, as we will see in the second part of this section. In the first part we briefly describe Theorem 20 which defines $S \in \mathbb{R}^{n\times m}, T \in \mathbb{R}^{n\times m}, L \in \mathbb{R}^{m\times m}, D \in \mathbb{R}^{m\times m}$ which are the main matrices we use this data type to instantiate.

Theorem 20. *Let B_0 be symmetric and positive definite, and assume that the k vector pairs $\{s_i, t_i\}_{i=0}^{k-1}$ satisfy $s_i^T t_i > 0, s_i = x_{i+1} - x_i$ and $t_i = \nabla L_{i+1} - \nabla L_i$. Let B_k be obtained by applying k BFGS updates with these vector pairs to B_0, using the formula:*

$$B_k = B_{k-1} - \frac{B_{k-1}s_{k-1}s_{k-1}^T B_{k-1}}{s_{k-1}^T B_{k-1} s_{k-1}} + \frac{y_{k-1}y_{k-1}^T}{y_{k-1}^T s_{k-1}}. \tag{5.3.1}$$

From the correction pairs $\{s_i, t_i\}$ define matrices $S_k, T_k \in \mathbb{R}^{p\times k}, L_k, D_k \in \mathbb{R}^{k\times k}$:

$$S_k = [s_0, ..., s_{k-1}], \qquad (L_k)_{i,j} = \begin{cases} s_{i-1}^T t_{j-1} & \text{if } i > j \\ 0 & \text{otherwise,} \end{cases} \tag{5.3.2}$$

$$T_k = [t_0, ..., t_{k-1}], \qquad D_k = \operatorname{diag}[s_0^T t_0, ..., s_{k-1}^T t_{k-1}].$$

Construct $G \in \mathbb{R}^{p\times 2k}$ and $R \in \mathbb{R}^{2k\times 2k}$ from B_0, S_k, T_k, L_k, D_k in the follow manner:

$$G = \begin{bmatrix} B_0 S_k & T_k \end{bmatrix}, \ R = \begin{bmatrix} S_k^T B_0 S_k & L_k \\ L_k^T & -D_k \end{bmatrix}^{-1}. \tag{5.3.3}$$

We then have that

$$B_k = B_0 - GRG^T. \tag{5.3.4}$$

For large-scale problems the matrices S_k and T_k are augmented by one column every iteration according to (5.3.2). The update (5.3.4) become inefficient as k grows, if all the previous pairs $\{s_i, t_i\}$ are used. Hence BFGS method is often used in the limited-memory setting, known as the L-BFGS method [77].

In the limited-memory setting, we maintain the set $\{s_i, t_i\}$ with the m most recent correction pairs by removing the oldest pair and adding the latest one. When the number of iteration k exceeds m, the representation of the matrices S_k, T_k, L_k, D_k needs to be slightly modified to reflect the changing nature of $\{s_i, t_i\}$: S_k and T_k are maintained as the $p \times m$ matrices with the i-th column of S_k and T_k being $s_{k-1-m+i}$ and $t_{k-1-m+i}$, respectively ($i = 1, 2, ..., m$). Also, L_k and D_k are now both the $m \times m$ matrices: D_k is a diagonal matrix with the i-th diagonal entry to be $s^T_{k-1-m+i} t_{k-1-m+i}$ and L_k is a strictly lower triangular matrix with the i-th row given by the $i-1$ leftmost entries of $s^T_{k-1-m+i} T_k$.

5.3.1 Efficient matrix mutating operations

As shown in Theorem 20, the compact representation requires efficiently mutating matrices such as $S \in \mathbb{R}^{n\times m}, T \in \mathbb{R}^{n\times m}, L \in \mathbb{R}^{m\times m}, D \in \mathbb{R}^{m\times m}$. The mutations, however, follow certain patterns that we shall exploit, i.e., (5.3.2), to define the data type `LMatrix`. For example, operations that are performed most often are the deletion of the first column and the addition of the new one to the end; although row mutations are also required, they are only needed for L, which is a much smaller matrix than S, i.e., m by m where m is a small chosen constant. We summarized in Table 5.1 the complexity of four `LMatrix` mutating operations required in obtaining the compact representation. It can be seen from the table that column operations can be implemented using just one built-in library function call. For row operations sometimes we need m separate function calls but again m is very small so the cost is negligible.

We also discuss implementing matrix-vector product on `LMatrix`, which will be used for maintaining S^TS. We will see that for a n by m matrix a naive implementation requires m separate lv1 BLAS function calls. But after careful arrangement through a maintained permutation matrix we are able to improve that to a single function call to lv2 BLAS function, which enjoys better multi-core usage and is more cache-friendly. Maintaining the permutation matrix can also be done efficiently, as efficient as one pass through a linear array of m entries which fits within a single cache line of most personal computers. Finally, we will see that, besides the matrix-vector product, the same mutating operations in Table 5.1, are needed again on the maintained matrix S^TS. Hence we also use `LMatrix` to instantiate matrix S^TS.

	Row Append	Row Pop	Column Append	Column Pop
Complexity	$O(m)$	$O(mn)$	$O(n)$	$O(m)$
Implementations	m `copy`	m `memmove`	1 `memcpy`	1 `memmove`

Table 5.1: Row and column operations on `LMatrix` type $M \in \mathbb{R}^{n\times m}$. Note that while one `memcpy` or `memmove` on $O(n)$ data has a complexity of $O(n)$, the constant in front of n can be much smaller than 1 (hence much faster than normal n flops), due to the use of cache optimizations and specialized processor instructions such as SIMD in those built-in library functions.

Mutable matrix type `LMatrix`

Column operations `LMatrix` is implemented such that two mutating operations on the matrix, *column append* (CA) and *column pop* (CP), are optimized for maximum possible efficiency. Here CA means inserting a column vector to the right, i.e., the inserted vector becomes the last column of the matrix; and CP means delete the first column on the left. By "maximum efficiency" we mean the complexity of both operations is linear to one dimension of the matrix. For example, assume that we initiate an instance of type `LMatrix` with a n by m matrix. Then CA can be done by one call to `memcpy` which copies $O(n)$ elements, and CP is done by one call to `memmove` which copies $O(m)$ elements while taking into account overlapping. We consider these complexities to be "optimal" because we need at least $O(n)$ to insert (and copy) a n-dimensional vector. The $O(m)$ complexity for CP is also efficient, especially in our case where we have $n \gg m$.

Matrix-vector product To achieve the optimal complexity for CP and CA, `LMatrix` uses two arrays internally, *ref* and *body*, where *body* stores the nm entries of the matrix in column major and *ref* stores pointers to each column. In particular, *ref* keeps the pointers sorted to reflect the column order of the matrix so that we do not need to move around nm entries in *body* every time the matrix is mutated. Hence for any `LMatrix` instance, a naive implementation of matrix-vector product would be a loop iterating through the *ref* array and performing a lv1 BLAS operation to compute vector-vector product for each column. To achieve better cache uses and parallel performance, here we discuss how to utilize lv2 BLAS library and combine it with permutation to implement the matrix-vector product in `LMatrix`.

Let us use $\hat{M} \in \mathbb{R}^{n\times m}$ to denote the matrix given by accessing *body* in column major, and $M \in \mathbb{R}^{n\times m}$ to denote the one given by *ref*. M and $\hat{M}$ will have exactly the same mn entries but different column order. For example, the i-th entry *ref*(i) points to $M(:,i)$, the i-th column in M, which might be in the j-th column of $\hat{M}$. This connection between M and $\hat{M}$ can be captured by a permutation matrix $P \in \mathbb{R}^{m\times m}$, i.e., $P(j,:) = e_i^T$ if $M(:,i) = \hat{M}(:,j)$. Computing the product $x = M^T * b$ can thus be done by two calls to the BLAS function dgemv, the first call to compute $\hat{x} = \hat{M}^T * b$ and the second to obtain the final result by permuting x, i.e., $x = P * \hat{x}$.

P is computed on the fly, from a linear array $p \in \mathbb{R}^m$ where $p(i) = j$ if $M(:,i) = \hat{M}(:,j)$, because maintaining p can be done more efficiently. For that we will exploit the fact that CA and CP are always performed *together* in Algorithm 20. To be more specific, at the beginning, we initialize p such that $p(i) = i$, i.e., P is an identity matrix and $M = \hat{M}$. Later whenever CA and CP are performed, we update p, i.e., for each entry of p either decreasing it by 1, or replacing it by m if that entry is 1 (assuming that the first column of M is $M(:,1)$ instead of $M(:,0)$).

The extra complexity of such arrangement consists of one $O(m)$ for updating p and another $O(m)$ for the matrix-vector multiplication $P * \hat{x}$, counting only the non-zero operations. Notice that once p is computed, it is valid to use until when `LMatrix` is mutated again. The complexity of matrix-vector product with `LMatrix` $M \in \mathbb{R}^{n\times m}$ is thus $O(m(n+1)) \approx O(mn)$ when m is small, which is roughly the same as the naive implementations but is much more cache-friendly.

Maintain $S_k^T S_k$

We maintain the small matrix $S_k^T S_k \in \mathbb{R}^{m\times m}$ and update it every iteration by a matrix-vector product which costs $O(nm)$ operations, rather than the required $O(m^2 n)$ if we compute the matrix from scratch. Consider S_{k-1} and S_k,

$$S_{k-1} = \begin{bmatrix} s_{k-m-1} & C \end{bmatrix}, \; S_k = \begin{bmatrix} C & s_{k-1} \end{bmatrix},$$

where they share the same sub-matrix denoted by C. Hence,

$$S_{k-1}^T S_{k-1} = \begin{bmatrix} s_{k-m-1}^T s_{k-m-1} & s_{k-m-1}^T C \\ C^T s_{k-m-1} & C^T C \end{bmatrix}, \; S_k^T S_k = \begin{bmatrix} C^T C & C^T s_{k-1} \\ s_{k-1}^T C & s_{k-1}^T s_{k-1} \end{bmatrix}.$$

Let us contain $S_{k-1}^T S_{k-1}$ in a `LMatrix` type structure M_{k-1} and consider the complexity of updating M_{k-1} to M_k such that M_k contains $S_k^T S_k$. Besides CP and CA, we introduce two mutating operations, *row pop* (RP) and *row append* (RA), to denote the corresponding row-wise operations. The complexity of the four mutations is summarized in Table 5.1. Updating M_{k-1} to M_k can then be expressed by the following operation,

$$M_k \leftarrow CA(RA(CP(RP(M_{k-1})), s_{k-1}^T C), S_k^T s_{k-1}) \tag{5.3.5}$$

where $RA(M, v)$ appends v to the last row of M and $CA(M, v)$ appends v to the last column of M. Note that $s_{k-1}^T C$ is simply the first $m-1$ sub-vector of $S_k^T s_{k-1}$. Hence the update (5.3.5) involves a `LMatrix`-vector product $S_k^T s_{k-1}$ followed by the four operations applied in sequence. The complexity of maintaining $S_k^T S_k$ is then $O(m^2 + m(m-1) + m - 1 + m - 1 + m) = O(m^2)$.

Reduced $Q \in \mathbb{R}^{n\times 2m}$ and $\hat{Q} \in \mathbb{R}^{2m\times n}$

A reduced matrix $Q_{\mathcal{I}} \in \mathbb{R}^{|\mathcal{I}|\times 2m}$ indexed by set $\mathcal{I}$ contains only a subset of rows in the full matrix Q. We have $Q_{\mathcal{I}} = Q$ when the free set $\mathcal{I}$ contains all the coordinates. In sparse optimization, $\mathcal{I}$ is constructed such that only a small subset of coordinates are included to be optimized. It is thus inefficient to construct the full matrix Q while only a small number of rows in Q are actually needed in the computation. Given $Q_{\mathcal{I}}$, the matrix $\hat{Q}$ can also be reduced to

$$\hat{Q}_{\mathcal{I}} = RQ_{\mathcal{I}}^T \text{ with } R \in \mathbb{R}^{2m\times 2m}$$

which is now a simpler linear system to solve with a RHS of a much smaller size.

5.4 Self-adaptive greedy active-set selection

In this section we describe the strategy to construct the active set $\mathcal{A}$. In the discussion that follows we will consider any point $x \in \mathrm{dom}(F)$, thus removing the superscript in x^k. We first define four index sets, which form a partition of the set $\mathcal{P} = \{1, ..., n\}$,

$$\begin{aligned}
\mathcal{Z}_1(x) &:= \{i \in \mathcal{P} \mid x_i \neq 0 \text{ and } |\nabla_i L(x)| \geq \lambda\},\\
\mathcal{Z}_2(x) &:= \{i \in \mathcal{P} \mid x_i \neq 0 \text{ and } |\nabla_i L(x)| < \lambda\},\\
\mathcal{Z}_3(x) &:= \{i \in \mathcal{P} \mid x_i = 0 \text{ and } |\nabla_i L(x)| > \lambda\},\\
\mathcal{Z}_4(x) &:= \{i \in \mathcal{P} \mid x_i = 0 \text{ and } |\nabla_i L(x)| \leq \lambda\}.
\end{aligned} \tag{5.4.1}$$

We denote as $P_i(x)$ the i-th coordinate of the least norm subgradient of F at x:

$$P_i(x) = \begin{cases} \nabla_i L(x) + \lambda & x_i > 0,\\ \nabla_i L(x) - \lambda & x_i < 0,\\ \max\{|\nabla_i L(x)| - \lambda, 0\} & x_i = 0 \end{cases} \tag{5.4.2}$$

The four sets can thus be expressed using $P(x)$ as follows:

$$\begin{aligned}
\mathcal{Z}_1(x) \cup \mathcal{Z}_2(x) \cup \mathcal{Z}_3(x) &= \{i \mid P_i(x) \neq 0\}\\
\mathcal{Z}_4(x) &= \{i \mid P_i(x) = 0\}
\end{aligned} \tag{5.4.3}$$

Also, from the optimality conditions of (5.1.1) it follows that,

$$||P(x)|| \to 0 \text{ as } w \to x^*. \tag{A1}$$

We denote the active set and free set at optimality as $\mathcal{A}^*$ and $\mathcal{I}^*$, respectively. We define $\mathcal{A}^*$ as the index set of zero entries in x^*, and $\mathcal{I}^*$ as that of nonzero entries, i.e.,

$$\mathcal{A}^* := \{i \in \mathcal{P} | x_i = 0\}, \ \mathcal{I}^* := \{i \in \mathcal{P} | x_i \neq 0\}.$$

From optimality condition we have,

$$\mathcal{A}^* = \mathcal{Z}_4(x^*), \ \mathcal{I}^* = \mathcal{Z}_1(x^*) \cup \mathcal{Z}_2(x^*), \ \mathcal{Z}_3(x^*) = \emptyset.$$

Hence the goal of any active-set method discussed here is to construct $\mathcal{A}$ such that $\mathcal{A} \to \mathcal{A}^*$ as $x \to x^*$ (assuming $\mathcal{I}$ is the complement of $\mathcal{A}$). Let us start with a "standard" active-set method (STD) which simply set $\mathcal{A}$ to $\mathcal{Z}_4(x)$. A nice property can then be established about STD, which shows that optimizing over the subspace formed by STD is equivalent to optimizing over the full space with a block-diagonal H matrix.

Lemma 21. *Let $x, g \in \mathbb{R}^n$, $\mathcal{I} := \{i \mid x_i \neq 0 \text{ or } |g_i| > \lambda\}$ and its complement $\mathcal{A} := \{i \mid x_i = 0 \text{ and } |g_i| \leq \lambda\}$. Let $H_{\mathcal{I}}$ be any p.d. matrix of dimension $|\mathcal{I}| \times |\mathcal{I}|$. The vector $d^* \in \mathbb{R}^n$ defined below,*

$$d^*_{\mathcal{A}} := \boldsymbol{0}, \ d^*_{\mathcal{I}} := \arg \min_{d \in \mathbb{R}^{|\mathcal{I}|}} \{\langle g_{\mathcal{I}}, d\rangle + \frac{1}{2}\langle d, H_{\mathcal{I}} d\rangle + \lambda \|x_{\mathcal{I}} + d\|_1\}. \tag{5.4.4}$$

is also the solution for the following problem,

$$\min_{d \in \mathbb{R}^n} \{\langle g, d\rangle + \frac{1}{2}\langle d, Hd\rangle + \lambda \|x + d\|_1\}, \tag{5.4.5}$$

where $H = \begin{bmatrix} H_{\mathcal{I}} & \boldsymbol{0} \\ \boldsymbol{0} & tI \end{bmatrix}$ is a n by n matrix with $t > 0$ as a fixed constant.

Proof. Note that (5.4.5) is block-separable with one block corresponding to set $\mathcal{A}$ and the other to set $\mathcal{I}$. The solution of $\mathcal{I}$ block is exactly $d^*_{\mathcal{I}}$. And applying the definition of $\mathcal{A}$ and the optimality conditions it is easy to see that the solution of $\mathcal{A}$ block is $\mathbf{0}$. □

In STD the free set $\mathcal{I}$ is constructed as $\mathcal{Z}_1(x) \cup \mathcal{Z}_2(x) \cup \mathcal{Z}_3(x)$ which eventually converges to $\mathcal{I}^* = \mathcal{Z}_1(x^*) \cup \mathcal{Z}_2(x^*)$ as $\mathcal{Z}_3(x)$ is diminishing at optimality. The idea is that for $\mathcal{I}$ to converge to $\mathcal{Z}_1(x^*) \cup \mathcal{Z}_2(x^*)$ we have to at lease include these two sets $\mathcal{Z}_1(x)$ and $\mathcal{Z}_2(x)$. But only including them would not converge because *for an active-set strategy to work it has to allow both directions – nonzero to turn to zero and the other way*, and obviously optimizing over $\mathcal{Z}_1(x) \cup \mathcal{Z}_2(x)$ only allows variables to change in one direction – from non-zero to zero. So STD also includes set $\mathcal{Z}_3(x)$.

In this work we propose another active-set method, which we denote as GRDY. The method computes two sets $\mathcal{Z}_2(x)$ and $\mathcal{Z}_1(x) \cup \mathcal{Z}_3(x)$, and construct $\mathcal{I}$ differently based on the difference of the two sets. In particular, at the beginning it sorts $\mathcal{Z}_1(x) \cup \mathcal{Z}_3(x)$, pick the first $|\mathcal{Z}_2(x)|$ and join it with $\mathcal{Z}_2(x)$; later when $|\mathcal{Z}_1(x) \cup \mathcal{Z}_3(x)|$ is close to $|\mathcal{Z}_2(x)|$, it includes all entries $\mathcal{Z}_1(x) \cup \mathcal{Z}_2(x) \cup \mathcal{Z}_3(x)$ in $\mathcal{I}$.

To explain the idea behind GRDY, let us consider how each of those sets is changing as x approaches optimality, i.e., $w \to x^*$. A key observation is that when x is far from x^*, $\mathcal{Z}_3(x)$ and $\mathcal{Z}_4(x)$ contain few entries as there are only a few zero elements in x. The set $\mathcal{Z}_2(x)$ is also small because most gradient elements are larger than λ which is often chosen to be a small number (less than one for example). In fact, $\mathcal{Z}_1(x)$ is often the largest set at this point. Note that the size of $\mathcal{I}$ is only $2|\mathcal{Z}_2(x)|$ by GRDY. As a comparison, the $\mathcal{I}$ by STD contains all entries from $\mathcal{Z}_1(x) \cup \mathcal{Z}_2(x) \cup \mathcal{Z}_3(x)$, which includes almost all the variables. Later as the algorithm advances, the size of $\mathcal{Z}_1(x)$ is decreasing while the other sets are expanding. Eventually condition (A1) demands that $\mathcal{Z}_4(x)$ contain all of the zero components in x and $\mathcal{Z}_3(x)$ contains none. One notable observation here is that at the neighborhood of x^*, i.e., $w \in \mathcal{N}_\sigma(x^*)$, we have the sizes of two sets $\mathcal{Z}_1(x)$ and $\mathcal{Z}_2(x)$ "converging" to each other:

$$|\mathcal{Z}_1(x)| \approx |\mathcal{Z}_2(x)| \approx |\mathcal{I}^*|/2$$

which implies that $|\mathcal{Z}_1(x) \cup \mathcal{Z}_3(x)|$ is close to $|\mathcal{Z}_2(x)|$. At this point GRDY will automatically include in $\mathcal{I}$ all entries from $\mathcal{Z}_1(x) \cup \mathcal{Z}_2(x) \cup \mathcal{Z}_3(x)$.

5.5 Fast coordinate descent on low-rank structure

At k-th iteration given the current iterate x^k, we apply coordinate descent method to the piecewise quadratic subproblem to obtain the direction d^k. Suppose jth coordinate in d^k is to be updated, i.e., $d^k \leftarrow d^k + z^* e_j$ (e_j is the j-th vector of the identity), where z^* is determined by solving the following one-dimensional problem,

$$z^* = \arg\min_z \ \frac{1}{2}(H_k)_{jj} z^2 + ((\nabla L_k)_j + (H_k d)_j) z + \lambda |x_j^k + d_j + z|. \tag{5.5.1}$$

Closed-form solutions exist and can be obtained by one soft-thresholding step [17, 78]:

$$z^* = -c + S(c - b/a, \lambda/a) \tag{5.5.2}$$

with a, b, c chosen to be $a = (H_k)_{jj}, b = (\nabla L_k)_j + (H_k d^k)_j$, $c = x_j^k + d_j^k$ and the shrinkage function as $S(u, r) := \operatorname{sgn}(u)\max(|u| - r, 0)$.

As mentioned above, the special low-rank update of H_k provides us an opportunity to accelerate the coordinate descent process. To see how, let us recall that $H_k = \frac{\mu}{2} I + B_k = \frac{\mu}{2} I + B_k^0 - GRG^T$. The basic matrix B_0 in (5.3.4) may vary from iteration to iteration, so here we use B_k^0 to denote the basic matrix in k-th iteration. A popular choice of B_k^0 in practice is to set it to $\frac{t_{k-1}^T t_{k-1}}{t_{k-1}^T s_{k-1}} I$, which is proved effective in ensuring that the search direction is well-scaled so that less time is spent on line search. Letting $\gamma_k = \frac{\mu}{2} + \frac{t_{k-1}^T t_{k-1}}{t_{k-1}^T s_{k-1}}$ and $\hat{G} = RG^T$, we have,

$$H_k = \gamma_k I - G\hat{G}.$$

where $G \in \mathbb{R}^{p \times 2m}$, $\hat{G} \in \mathbb{R}^{2m \times p}$ and m chosen as a small constant. Clearly we do not need to explicitly store or compute H_k. Instead, since H_k is only used through $(H_k)_{ii}$ and $(H_k d)_i$ when applying soft-thresholding steps to updating each coordinate i, we can only store the diagonal elements of H_k and compute $(H_k d)_i$ on the fly whenever it is needed. Specifically,

$$(H_k)_{ii} = \gamma_k - g_i^T \hat{g}_i \tag{5.5.3}$$

where g_i is the ith row of the matrix G and $\hat{g}_i$ is the ith column vector of the matrix $\hat{G}$. To compute $(H_k d)_i$, we maintain a $2m$ dimensional vector $d_G := \hat{G}d$, then

$$(H_k d^k)_i = \gamma_k d_i^k - g_i^T d_G \tag{5.5.4}$$

which takes $O(2m)$ flops, instead of $O(p)$ if we multiply d by the ith row of H_k. Notice though, that after taking the soft-thresholding step d_G has to be updated each time by $d_G \leftarrow d_G + z_i \hat{g}_i$. This update requires little effort given that $\hat{g}_i$ is a vector with $2m$ dimensions where m is often chosen to be between 3 and 20. However, we need to use extra memory for caching $\hat{G}$ and d_G which takes $O(2mp + 2m)$ space. With the other $O(2p + 2mp)$ space for storing the diagonal of H_k, G and d, altogether we need $O(4mp + 2p + 2m)$ space, which can be written as $O(4mp)$ when $p \gg m$.

So far we have discussed most missing details from Algorithm 19, and the results are summarized in Algorithm 20. Particularly, in Step 8 of Algorithm 20 we use a set function $\Gamma(\mathcal{S}, s)$ which returns a new set composed of s elements from $\mathcal{S}$. It returns $\mathcal{S}$ itself if $s \geq |\mathcal{S}|$. To determine which s elements should be chosen, greedy methods as described above or randomization can be used. In Step 9 we compute and store the diagonal entries in Hessian approximation B_k with indices contained in $\mathcal{I}_k$. In Step 11 we divided μ_0 by β so that we have $\mu = \mu_0$ for the first while iteration. In Step 14 we apply coordinate descent, as discussed in Section 5.5, to the subproblem constructed from $\mathcal{I}_k$. Finally, we introduce a function $l(k)$ in Step 14, which denote the number of inner iterations we apply at the k-th outer iteration. We will discuss $l(k)$ in more details next.

5.6 Inner loop termination criteria and global convergence

The inner loop of Algorithm 20 is terminated after $l(k)$ iterations, hence the termination criteria we use is simply the number computed by the function l with respect to the outer iteration counter k. Alternatively, one could consider terminating the

Algorithm 20: Low rank **H**essian **A**pproximation in **A**ctive-set **C**oordinate descent (LHAC)

```
1  Input: $x^0$ initial iterate, $m > 0$ the LBFGS parameter,
   $\mu_0 > 0, \eta > 1, \beta > 1, 1 \geq \rho \geq 0$
2  Set $x^1 \leftarrow x^0, \theta_1 \leftarrow 1, G \leftarrow 0, \hat{G} \leftarrow 0$
3  for $k = 1, 2, ...$ do // outer loop
4      Compute $\mathcal{Z}_2(x^k)$ and $\mathcal{Z}_1(x^k) \cup \mathcal{Z}_3(x^k)$ by (5.4.1)
5      if $|\mathcal{Z}_1(x^k) \cup \mathcal{Z}_3(x^k)| \leq \eta |\mathcal{Z}_2(x^k)|$ then
6          $\mathcal{I}_k = \mathcal{Z}_2(x^k) \cup \mathcal{Z}_1(x^k) \cup \mathcal{Z}_3(x^k)$
7      else
8          $\mathcal{I}_k = \Gamma(\mathcal{Z}_1(x^k) \cup \mathcal{Z}_3(x^k), |\mathcal{Z}_2(x^k)|) \cup \mathcal{Z}_2(x^k)$
9      for coordinate $j \in \mathcal{I}_k$ do
10         Compute $(B_k)_{jj} = \theta_k - g_j^T \hat{g}_j$
11     Set $d_G \leftarrow 0, d^k \leftarrow 0, \mu \leftarrow \mu_0/\beta$
12     while condition (5.2.3) is not satisfied do
13         $\mu \leftarrow \mu\beta, \gamma_k \leftarrow \frac{\mu}{2} + \theta_k$
14         for $i = 1, 2, ..., l(k)$ do // inner loop
15             for coordinate $j \in \mathcal{I}_k$ do
16                 Compute $a = \frac{\mu}{2} + (B_k)_{jj}, b = \nabla_j L(x^k) + \gamma_k d_j^k - g_j^T d_G$ and
                   $c = x_j^k + d_j^k$
17                 Compute $z = -c + \text{sgn}(c - b/a)\max(|c - b/a| - \lambda/a, 0)$
18                 Set $d^k \leftarrow d^k + z e_j$, $d_G \leftarrow d_G + z\hat{g}_j$
19     Set $x^{k+1} \leftarrow x^k + d^k$, $t_k \leftarrow \nabla L(x^{k+1}) - \nabla L(x^k)$, $s_k \leftarrow x^{k+1} - x^k$, $\theta_{k+1} \leftarrow \frac{t_k^T t_k}{t_k^T s_k}$
20     Set $S_{k+1}, T_{k+1}, D_{k+1}, L_{k+1}$ according to Theorem 20
21     Set $G \leftarrow \begin{bmatrix} \theta_{k+1} S_{k+1} & T_{k+1} \end{bmatrix}$, $\hat{G} \leftarrow \begin{bmatrix} \theta_{k+1} S_{k+1}^T S_k & L_{k+1} \\ L_{k+1}^T & -D_{k+1} \end{bmatrix}^{-1} G^T$.
```

inner loop once the duality gap is smaller than a required bound on ϕ_k. Checking duality gap, however, can be computationally more expensive than simply counting the number of steps. It is also not clear how often the check needs to be performed. In Algorithm 20, for instance, each coordinate descent step is so cheap that it is inefficient to perform the expensive check after each such cheap step.

Let us now discuss the functional forms $l(k)$ can take. l can be as simple as a constant, for instance. That is, the subproblems are always minimized to a fixed accuracy proportional to the value of the constant, e.g., a large constant is equivalent to exact minimization of the subproblems. In general, however, we would like l to be a monotonically increasing function so that the subproblem is minimized more accurately as outer iteration moves closer to the optimality, i.e., ϕ_k is a monotonically decreasing function of k which approaches 0 as $k \to \infty$. This function $k \to \phi_k$ is determined by the outer convergence rate. For example, if the outer iteration convergences with k less than 10, then it means that ϕ_k has to also go to zero in 10 iterations. Given these two functions, $k \to l(k)$ and $k \to \phi_k$, there is a third one. Since ϕ_k and $l(k)$ are both monotonic function of k, ϕ_k can be alternatively expressed using l. And this function, $l \to \phi_k$, is predefined by the convergence rate of the inner algorithm. Knowing any two of the three functions, $k \to \phi_k$, $k \to l(k)$ and $l \to \phi_k$, we are able to derive the third one. In other words, how fast l should be increasing will depend on the relation between $l(k)$ and ϕ_k, and how fast ϕ_k should be approaching zero will depend on the rate of the outer iteration. Hence the actual form l takes will be the result of taking both into account. Theorem 22, which we adopt from [20], expresses it formally and precisely.

Theorem 22. *Let $l(k)$ denote the number of inner iterations taken to approximately minimize Q_{H_k,x^k} in Algorithm 19, ϕ_k denote the minimization error as defined in* (5.2.2)*, and K denote the number of inner iterations (of the chosen algorithm) to achieve ϵ-accuracy on F after t outer iteration, i.e., $F(x^t) - F(x^*) \leq \epsilon$. The relations between $l(k), \phi_k$ and K are stated in the following two cases, with $t = \lceil \frac{\max\{ba, \frac{1}{c}\}}{\epsilon} + 1 \rceil$ and $b, c > 0$ as fixed constants*[1]*,*

1. *if $\phi_k \leq a^2/l(k)^2$ for some $a > 0$, then,*

$$l(k) = \alpha k + \beta \text{ and } K = \beta t + \frac{\alpha}{2}(t-1)t;$$

[1] b and c are defined in [20] and depends on the bounds of eigenvalues of H_k.

2. if $\phi_k \leq \delta^{l(k)} M_Q$, for some constants $0 < \delta < 1$ and $M_Q > 0$, then,

$$l(k) = 2\log_{\frac{1}{\delta}}(k) \text{ and } K = \sum_{k=0}^{t} 2\log_{\frac{1}{\delta}}(k) \leq 2t\log_{\frac{1}{\delta}}(t).$$

Theorem 22 shows that l has to increase at least as fast as a linear function of k, if the inner algorithm converges as fast as FISTA [2], for example, and that if the rate of the inner algorithm is linear, then l can slow down to a logarithmically-growing function. In particular, we note that the objective functions in our subproblems are all σ-strongly convex, so a simple proximal gradient method, or some of its accelerated versions, will enjoy linear convergence rates when applied to these subproblems. In other words, if we replace the coordinate descent in Algorithm 20 with proximal gradient methods, it is sufficient for $l(x)$ to increase logarithmically because of the linear rate of proximal gradient on the strongly convex subproblems. The next question is what would $l(x)$ be if we keep Randomized Coordinate Descent (RCD) as it is in Algorithm 20? The major distinction is that now the relation between ϕ_k and $l(k)$ does not always hold as required by Theorem 22, due to the non-deterministic nature of RCD. We could repeatedly apply RCD until the relation is verified, but the verification tends to be computationally cumbersome and thus defeat the purpose of using $l(k)$ in the first place. The next Theorem 23 [20] we present extends Theorem 22 to the above probabilistic case. It allows the relation between ϕ_k and $l(k)$ to hold with certain probability. And the price we pay is a constant factor depending on the magnitude of that probability. Finally, we conclude that at k-th iteration $O(\log(k))$ passes of coordinates are desired in the inner loop of Algorithm 20.

Theorem 23. *Suppose that the coordinate in Step 15 of Algorithm 20 is picked randomly with probability $\frac{1}{|\mathcal{I}_k|}$. Let p denote the probability that $\phi_k \leq M_Q e^{-\frac{l(k)}{|\mathcal{I}_k|(1+\mu(H_k))}}$ holds at any iteration k, $\mu(H)$ denote a constant that measures conditioning of H along the coordinate directions and M_Q denote an upper bound on $\frac{Q_{H_k,x^k}(0)-Q^*_{H_k,x^k}}{p}$. Then at any iteration k the number of coordinate descent steps we need is given by,*

$$l(k) = O(|\mathcal{I}_k|(1+\mu(H_k))\log(kp/M_Q)).$$

And ϵ-accuracy is achieved in expectation, i.e., $E[F(x^k)-F(x^*)] \leq \epsilon$, after $O(\frac{1}{\epsilon}\log(\frac{1}{\epsilon}))$ coordinate descent steps.

5.7 Numerical experiments

We report results on training sparse logistic regression (5.7.1) with various datasets listed in Table 5.2.

$$\min_{w\in\mathbb{R}^p} \; F(w) = \lambda\|w\|_1 + \frac{1}{N}\sum_{n=1}^{N}\log(1+\exp(-y_n \cdot w^T x_n)), \tag{5.7.1}$$

Data	p	N	#non-zeros	Description
a9a	123	32,561	451,592	'census Income' dataset.
epsilon	2000	100,000	200,000,000	PASCAL challenge 2008.
gisette	5000	6,000	29,729,997	handwritten digit recognition.
mnist8m	784	1,000,000	199,401,957	handwritten digit recognition.
slices	385	53,500	20,597,500	CT slices location prediction.

Table 5.2: Data sets used in the experiments.

5.7.1 Active-set methods

In this experiment we compare the strategy used in Algorithm 20, denoted as GRDY-AD, with six other active-set strategies. The standard approach STD described in Section 5.4 is also included in the comparison. The rest are variants derived from GRDY-AD and STD, all trying to maintain a smaller free set than STD so that the subproblems can be minimized more efficiently. A smaller free set leads to cheap iteration, but also often suffers from slow convergence. An extreme case would be coordinate descent where the free set is a singleton, which takes lots of iteration/coordinate updates to converge. Hence, many of the strategies here are greedy-based, updating only those variables with greatest optimality violations, in the hope that even though other variables are fixed there are still enough "useful" information in the subproblem for determining a good descent direction. The next thing to consider is how many variables to be included in the free set. It is required that the free set converge to the non-zero subspace in the optimal solution, hence a strategy fixing the size of the free set would not work, such as coordinate descent where that size is fixed to 1. Strategies included in the experiments all come with different adaptive selection criteria.

Finally, we also introduce "inexactness" into the strategies, i.e., an "inexact" version of $\mathcal{Z}_1(x), \mathcal{Z}_2(x)$ and $\mathcal{Z}_3(x)$. Together they make up a subset of (5.4.3) containing those indices i such that $|P_i(x)|$ is above a small threshold $\epsilon > 0$,

$$\begin{aligned}
\mathcal{Z}_1^\epsilon(x) &:= \{i \mid x_i \neq 0 \text{ and } |\nabla_i L(x)| - \lambda > \epsilon\} \\
\mathcal{Z}_2^\epsilon(x) &:= \{i \mid x_i \neq 0 \text{ and } \lambda - |\nabla_i L(x)| > \epsilon\} \\
\mathcal{Z}_3^\epsilon(x) &:= \{i \mid x_i = 0 \text{ and } |\nabla_i L(x)| - \lambda > \epsilon\}
\end{aligned}$$

Similar to (5.4.3), it can be shown by definition that the following relation holds

$$\mathcal{Z}_1^\epsilon(x) \cup \mathcal{Z}_2^\epsilon(x) \cup \mathcal{Z}_3^\epsilon(x) = \{i \mid |P_i(x)| \geq \epsilon\}.$$

We describe each strategy below. Note that a set of indices/variables that are *selected* means that they are to be updated in the inner loop, i.e., those included in the free set $\mathcal{I}$, and others that are not *selected* are fixed through the inner loop.

- **GRDY**: a greedy strategy that selects $\mathcal{Z}_1(x) \cup \mathcal{Z}_2(x) \cup \mathcal{Z}_3(x)$ and sort the indices based on their corresponding value $|P_i(x)|$, i.e., $||\nabla_i L(x)| - \lambda|$.

- **STD**: selects $\mathcal{Z}_1(x) \cup \mathcal{Z}_2(x) \cup \mathcal{Z}_3(x)$

- **GRDY-CTZ**: a greedy strategy that selects $\mathcal{Z}_1(x) \cup \mathcal{Z}_2(x) \cup \mathcal{Z}_3(x)$, sorts the indices and pick the first $\mathcal{Z}_1(x) \cup \mathcal{Z}_2(x)$ that have the largest value in P.

- **GRDY-EP** : a greedy strategy that selects and sorts $\mathcal{Z}_1(x) \cup \mathcal{Z}_2(x) \cup \mathcal{Z}_3^{\epsilon}(x)$ for some small $\epsilon > 0$ (we choose $\epsilon = 0.01$ in this experiment).
- **GRDY-AD**: the strategy used in Algorithm 20 – it selects indices from both $\mathcal{Z}_2(x)$ and a fraction of $\mathcal{Z}_1(x) \cup \mathcal{Z}_3(x)$. I.e.,, if $|\mathcal{Z}_1(x) \cup \mathcal{Z}_3(x)|$ is close to that of $|\mathcal{Z}_2(x)|$, then it selects all indices from $\mathcal{Z}_1(x) \cup \mathcal{Z}_2(x) \cup \mathcal{Z}_3(x)$; otherwise, it sorts $\mathcal{Z}_1(x) \cup \mathcal{Z}_3(x)$ and pick the first $|\mathcal{Z}_2(x)|$ and join it with $\mathcal{Z}_2(x)$.
- **STD-EPA**: selects $\mathcal{Z}_1(x) \cup \mathcal{Z}_2(x) \cup \mathcal{Z}_3^{\epsilon}(x)$ with aggressively chosen ϵ (0.5 is used here).
- **STD-EP**: selects $\mathcal{Z}_1(x) \cup \mathcal{Z}_2(x) \cup \mathcal{Z}_3^{\epsilon}(x)$ with ϵ equal to 0.01.

All the strategies can be implemented very efficiently. Consider **GRDY-AD** for example. We maintain a *vector* of size n and initialize two counters α and β with 0. The free set is then computed as follows. For each coordinate i, if condition $|\nabla_i L(x)| > \lambda$ is true, then we increase the counter α by one and insert i into the end of *vector* and make sure that the last α entries remain sorted; if the condition is false, then we check x_i – if it is non-zero, then we put i at the front of *vector* and increase the counter β by one. After going through all the indices, the last α entries of the *vector* are those indices with $|\nabla_i L(x)| > \lambda$, and the first β are those with both $|\nabla_i L(x)| \leq \lambda$ and $x_i \neq 0$. That is,

$$\alpha = |\mathcal{Z}_1(x) \cup \mathcal{Z}_3(x)| \text{ and } \beta = |\mathcal{Z}_2(x)|,$$

according to definitions in (5.4.1). We compare the number α and β. If $\alpha \gg \beta$, then we copy the last β entries of *array*, put them behind the first β elements and set the size of $\mathcal{I}$ to 2β; if $\alpha \approx \beta$, then we copy all α entries to the front and set size of active set to $\alpha + \beta$. Note that since we maintain the last α entries to be sorted in descending order, this is a greedy method that always includes those indices from $\mathcal{Z}_1(x) \cup \mathcal{Z}_3(x)$ that have the largest dual constraints violations.

The main results are presented in Table 5.3. We first point out the two strategies that do not work well – STD-EPA and STD-EP. The former does not finish within

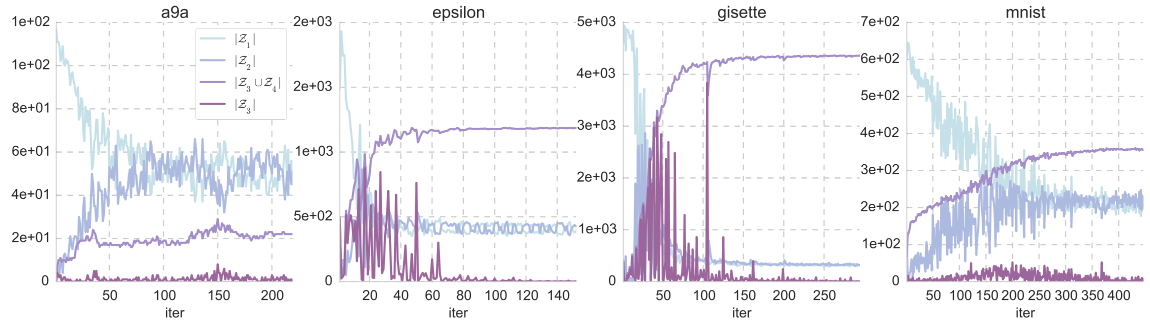


Figure 5.1: Illustrate the dynamics of the four index sets (5.4.1) as $x \to x^*$. Note how $|\mathcal{Z}_1(x)|$ and $|\mathcal{Z}_2(x)|$ start from the top and the bottom, respectively, and come together in the end, each containing (roughly) half the size of non-zero elements in x^*. Also note that $|\mathcal{Z}_3(x)|$ vanishes at optimality.

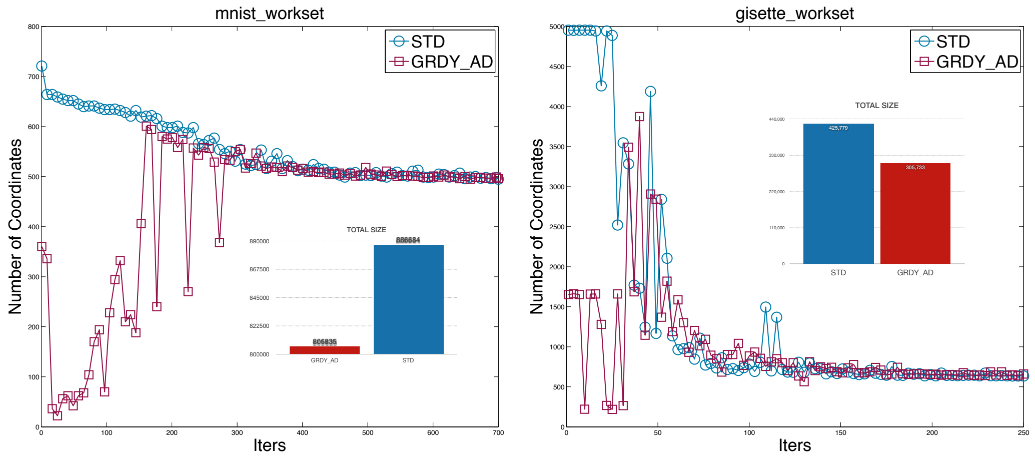


Figure 5.2: Evolution of free sets in STD and GRDY-AD. *Left:* dataset mnist ($p = 784, N = 1,000,000$). *Right:* dataset gisette ($p = 5000, N = 6000$). The line plots are the size of free set against iterations and the bars denote the total sum of free set sizes over all iterations

the given computing budget (1500 iterations) in four out of five experiments, and the latter fails once and performs poorly in the other four experiments. They both selects $\mathcal{Z}_1(x) \cup \mathcal{Z}_2(x) \cup \mathcal{Z}_3^{\epsilon}(x)$ but differ in the actual value of ϵ (0.01 v.s. 0.5). Here $\mathcal{Z}_3^{\epsilon}(x)$ leaves out those indices with small $P_i(x)$ so that the corresponding x_i remains

Table 5.3: Compare the performance of different active-set strategies.

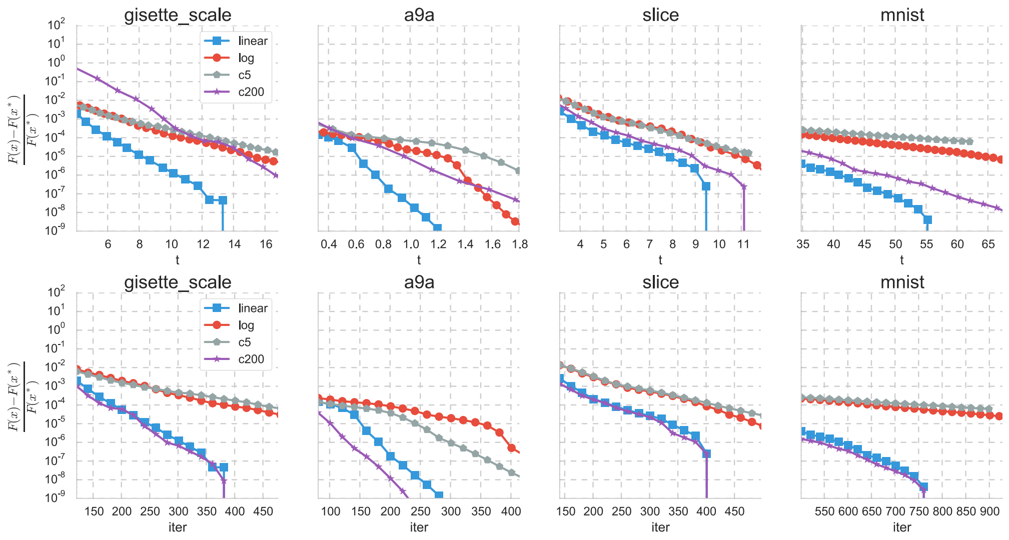
	A9A		SLICE		GISETTE		EPSILON		MNIST	
	TIME (S)	ITERS	TIME (S)	ITERS	TIME (S)	ITERS	TIME (S)	ITERS	TIME (S)	ITERS
GRDY	2.3663	360	9.9324	450	13.2527	420	29.1595	330	64.0209	810
STD	2.8944	420	10.5150	480	14.8445	480	**28.8553**	330	64.8016	840
GRDY-CTZ	2.6271	390	10.0439	450	12.3715	420	29.4759	330	59.9645	**780**
GRDY-EP	2.3768	360	9.9739	450	13.2448	420	29.2295	330	63.2239	810
GRDY-AD	**2.0126**	**330**	**8.4519**	**390**	**12.1911**	**390**	29.4915	330	**59.0973**	**780**
STD-EPA	59.9110	–	32.5708	–	51.4633	–	156.2294	–	62.1497	810
STD-EP	2.7650	420	10.9344	480	12.9109	420	792.5668	–	63.0899	810

zero and untouched although they violates the optimality conditions. It is, however, worth pointing out that GRDY-EP using the similar strategy but in a greedy manner performs quite well and even beats STD in most of the experiments. In fact, all four greedy-based strategies in general outperform STD.

The clear winner of all is GRDY-AD, which outperforms others in four out of five experiments and by measure of both time and number of iterations. Compared to other methods, GRDY-AD not only generates smaller free set but also requires fewer number of iterations to reach the same accuracy. In other words, GRDY-AD iterates fast because of smaller subproblems to minimize, without suffering from slow convergence! Figure 5.2 plots the size of the free set at each iteration and it compares GRDY-AD against STD. For STD, the size of $\mathcal{I}$ starts high at the top in the beginning, and slide gradually as the algorithm progresses. As a comparison, that of GRDY-AD takes another path – starts from the middle, and most of the time runs below the blue line while oscillating up and down. In total GRDY-AD requires less number of coordinate updates than STD does, as shown by the bar plots in Figure 5.2.

The observations that motivate GRDY-AD is illustrated in Figure 5.1. The plot shows the evolution patterns of the four index sets introduced in Section 5.4. It can be seen from the plot that $|\mathcal{Z}_1(x)|$ and $|\mathcal{Z}_2(x)|$ are converging to each other, each containing roughly half the entries from $\mathcal{I}^*$ in the end. But at the beginning

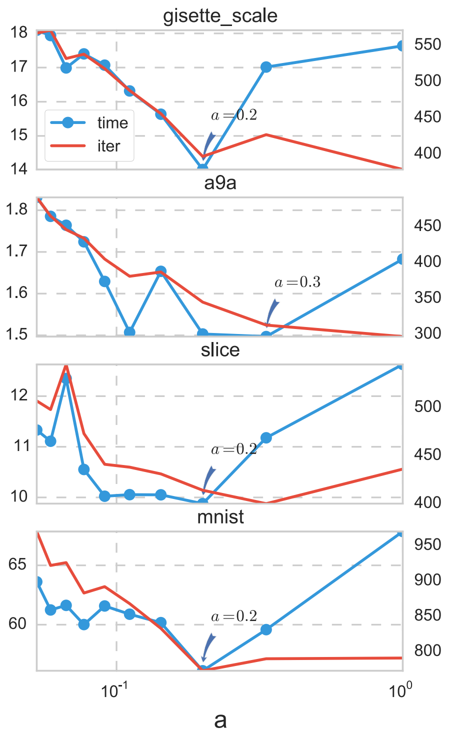
Figure 5.3: Study different inner loop termination criteria and its effect on algorithm performance. Time (top) and iterations (bottom) comparison for different $l(k)$: log, linear, constant 5 and constant 200.



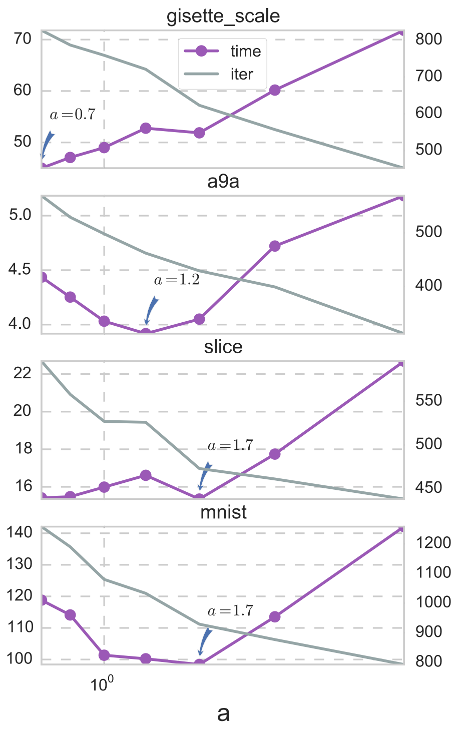
they are quite different such that the initial $|\mathcal{Z}_1(x)|$ is almost the size of the problem while $|\mathcal{Z}_2(x)|$ is close to zero. Also included in the plots are the number of zero entries $|\mathcal{Z}_3(x) \cup \mathcal{Z}_4(x)|$ and $|\mathcal{Z}_3|$, the number of zero entries that violate the optimality conditions. It can be seen that $|\mathcal{Z}_3(x) \cup \mathcal{Z}_4(x)|$ is moving upwards in general while $|\mathcal{Z}_3|$ oscillates, quite a bit in the beginning, but dies down in the end.

5.7.2 Inner loop termination and convergence

In the first experiment (see Figure 5.3) we study different termination functions $l(k)$, e.g., log, linear, constant etc., and its impact on overall running time and number of outer iterations required to converge. We will see that a faster $l(k)$ decreases the total number of outer iterations, but only to a certain point. After that the number of outer iterations will stay roughly the same. In other words, solving the subproblems more accurately, up to a point, does not further improve the convergence of the outer iteration. This corresponds well to the convergence theory proposed in [20]. But



(a) Coordinate descent



(b) Proximal gradient

Figure 5.4: Consider linear function $l(k) = ak + b$ and study the effect of a on running time and iterations for different inner algorithm, i.e., coordinate descent (a) and proximal gradient (b).

solving the subproblems more accurately does affect the overall running time. In fact, choosing a proper $l(k)$ is critical to the performance of the algorithm, since a too slow or too fast $l(k)$ either takes too many iterations, hence more function and gradient evaluations, or spends too much time on solving the subproblems without making progress on the actual objective. The experiments show that a linearly-increasing $l(k)$ performs the best in the context of Algorithm 20.

In the second experiment (see Figure 5.4(a) and Figure 5.4(b)) we study $l(k)$ and its relations with the inner algorithm in the context of linear functions $l(k) = ak + b$. We apply both coordinate descent and proximal gradient to the subproblems and we find that the best a for coordinate descent is around 0.2 and that for proximal gradient is around 1.2. In other words, at any given iteration k we need over six times more proximal gradient iterations on the subproblem than the number of passes we need if applying coordinate descent, to achieve either the best overall running time or the optimal number of outer iterations.

Results are reported in Figure 5.4. Figure 5.3 plots Algorithm 20 with four different $l(k)$ on various datasets in Table 5.2. The value of $\log(\frac{F(x)-F(x^*)}{F(x^*)})$ against both iteration and CPU time is reported, where $F(x^*)$ is estimated by pre-solving the problem to an accuracy of $1e^{-10}$. The four functions $l(k)$ are $\log(k)+1, 0.2k+1, 5$ and 200, where subproblems are exactly minimized with $l(k) = 200$ and are inexactly minimized, to a fixed accuracy, with $l(k) = 5$. The iteration plots show that $l(k) = 200$ achieves the optimal outer iteration convergence. But note that the performances of $l(k) = 200$ and $l(k) = 0.2k+1$ are very close and are almost identical in three out of four cases. Let us compare in terms of the total number of inner iterations. Summing $l(k) = 0.2k+1$ for k from 1 to 400, for instance, gives 16,041 inner iterations, while 400 times $l(k) = 200$ yields $80,000$. In other words, with the same outer convergence rate, $l(k) = 0.2k+1$ requires the number of inner iterations 5 times less than $l(k) = 200$ does! This also explains the difference between these two in the time plots.

Let us now compare $l(k) = \log(k)+1$ with $l(k) = 0.2k+1$. The total inner iteration in the log case is 1769 with k from 1 to 400. That is almost 10 times less than the 16,041 required by the linear $l(k)$. But the linear function is still much more efficient, as shown in the time plots. The reason can be read from the iteration plots. It shows that the objective value converges much faster on the outer iteration in the linear case. In fact, a superlinear convergence can be observed for $l(k) = 0.2k+1$, while the log case exhibits a merely linear trend, hence much more time spent on function and gradient evaluations.

Finally, let us look at the results in Figure 5.4(a) and Figure 5.4(b). Here we plot the final iteration number and running time against different choices of a in

$l(k) = ak + b$. As can be observed from both plots, iteration number decreases as a increases, and running time increases after decreasing to a point first. The best value for a is where running time reaches its minimum value. For coordinate descent that is $a = 0.2$ and for proximal gradient it is $a = 1.2$. While both achieves linear rate on the strongly convex subproblems, it is known in theory that proximal gradient can be a constant factor slower than coordinate descent if the condition number of the subproblems along the coordinate direction is better than the overall condition number. Hence the experiment results show that $l(k)$ needs to increase fast if a slow-rate algorithm is used in the inner loop.

5.7.3 Performance comparison with other algorithms

The aim of this section is to provide validation for our general purpose algorithm, but not to conduct extensive comparison of various inexact proximal Newton approaches. In particular, we aim to demonstrate a) that using the exact Hessian is not necessary in these methods, b) that backtracking using prox parameter, based on sufficient decrease condition, which our theory uses, does in fact work well in practice and c) that randomized coordinate descent is at least as effective as the cyclic one, which is standardly used by other methods.

LHAC, for **L**ow rank **H**essian **A**pproximation in **A**ctive-set **C**oordinate descent, is a C/C++ package that implements Algorithms 16- 17 for solving general ℓ_1 regularization problems. We conduct experiments on two of the most well-known ℓ_1 regularized models – Sparse Inverse Covariance Selection (SICS) and Sparse Logistic Regression (SLR). The following two specialized C/C++ solvers are included in our comparisons:

- QUIC: the quadratic inverse covariance algorithm for solving SICS described in [17].
- LIBLINEAR: an improved version of GLMNET for solving SLR described in [4, 5].

Note that both of these packages have been shown to be the state-of-the-art solvers in their respective categories (see e.g. [4, 17, 18, 79]).

Both QUIC and LIBLINEAR adopt line search to ensure function reduction. We have implemented line search in LHAC as well to see how it compares to the updating of prox parameter proposed in Algorithm 15. In all the experiments presented below use the following notation.

- LHAC: Algorithm 17 with backtracking on prox parameter.
- LHAC-L: Algorithm 17 with Armijo line search procedure described below in (5.7.5).

Experimental Settings For all of the experiments we choose the initial point $x_0 = \mathbf{0}$, and we report running time results in seconds, plotted against log-scale relative objective function decrease given by

$$\log(\frac{F(x) - F^*}{F^*}) \tag{5.7.2}$$

where F^* is the optimal function value. Since F^* is not available, we compute an approximation by setting a small optimality tolerance, specifically 10^{-7}, in QUIC and LIBLINEAR. All the experiments are executed through the MATLAB mex interface. We also modify the source code of LIBLINEAR in both its optimization routine and mex gateway function to obtain the records of function values and the running time. We note that we simply store, in a double array, and pass the function values which the algorithm already computes, so this adds little to nothing to LIBLINEAR's computational costs. We also adds a function call of *clock()* at every iteration to all the tested algorithms, except QUIC, which includes a "trace" mode that returns automatically the track of function values and running time, by calling *clock()* iteratively. For both QUIC and LIBLINEAR we downloaded the latest versions of the publicly available source code from their official websites, compiled and built the software on the machine on which all experiments were executed, and which uses 2.4GHz quad-core Intel Core i7 processor, 16G RAM and Mac OS.

The optimal objective values F^* obtained approximately by QUIC and LIBLINEAR are later plugged in LHAC and LHAC-L to terminate the algorithm when the following condition is satisfied

$$\frac{F(x) - F^*}{F^*} \leq 10^{-8} \tag{5.7.3}$$

In LHAC we chose $\bar{\mu} = 1, \beta = 1/2$ and $\rho = 0.01$ for sufficient decrease (see Algorithm 18), and for LBFGS we use $m = 10$.

When solving the subproblems, we terminate the RCD procedure whenever the number of coordinate steps exceeds

$$(1 + \lfloor \frac{k}{m} \rfloor)|\mathcal{I}_k| \tag{5.7.4}$$

where $|\mathcal{I}_k|$ denotes the number of coordinates in the current working set. Condition (5.7.4) indicates that we expect to update each coordinate in $\mathcal{I}_k$ only once when $k < m$, and that when $k > m$ we increase the number of expected passes through $\mathcal{I}_l$ by 1 every m iterations, i.e., after LBFGS receives a full update. The idea is not only to avoid spending too much time on the subproblem especially at the beginning of the algorithm when the Hessian approximations computed by LBFGS are often fairly coarse, but also to solve the subproblem more accurately as the iterate moves closer to the optimality. Note that in practice when $|\mathcal{I}_k|$ is large, the value of (5.7.4) almost always dominates k, hence it can be lower bounded by $l(k) = ak + b$ with some reasonably large values of a and b, which, as we analyzed in Section 4.5.1, guarantees the sub linear convergence rate. We also find that (5.7.4) works quite well in practice in preventing from "over-solving" the subproblems, particularly for LBFGS type algorithms. In 5.6 and 5.8 we plot the data with respect to the number of RCD iterations. In particular Figures 5.6(a) and 5.8(a) show the number of RCD steps taken at the k-th iteration, as a function of k. Figures 5.6(b) and 5.8(b) show convergence of the objective function to its optimal value as a function of the total number of RCD steps taken so far (both values are plotted in logarithmic scale). Note that RCD steps are not the only component of the CPU time of the algorithms, since gradient computation has to be performed at least once per iteration.

In LHAC-L, a line search procedure is employed, as is done in QUIC and LIBLINEAR, for the convergence to follow from the framework by [80]. In particular, the Armijo rule chooses the step size α_k to be the largest element from $\{\beta^0, \beta^1, \beta^2, ...\}$ satisfying

$$F(x_k + \alpha_k d_k) \leq F(x_k) + \alpha_k \sigma \Delta_k \tag{5.7.5}$$

where $0 < \beta < 1, 0 < \sigma < 1$, and $\Delta_k := \nabla f_k^T d_k + \lambda \|x_k + d_k\|_1 - \lambda \|x_k\|_1$. In all the experiments we chose $\beta = 0.5, \sigma = 0.001$ for LHAC-L.

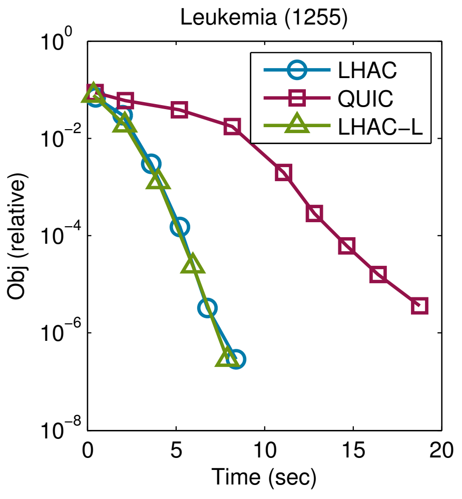
Sparse Inverse Covariance Selection The sparse inverse covariance selection problem is defined by

$$\min_{X \succ 0} \quad F(X) = -\log\det X + \mathrm{tr}(SX) + \lambda ||X||_1 \tag{5.7.6}$$

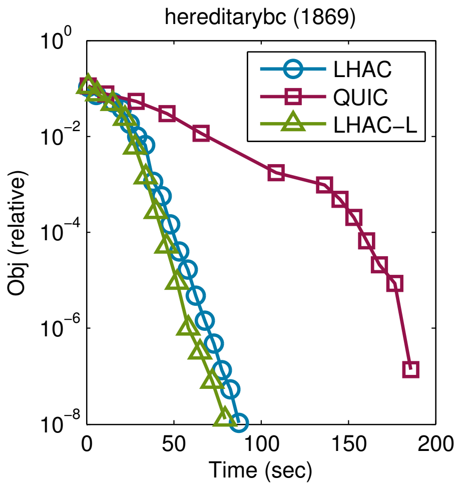
where the input $S \in \mathbb{R}^{p \times p}$ is the sample covariance matrix and the optimization is over a symmetric matrix $X \in \mathbb{R}^{p \times p}$ that is required to be positive definite.

For SICS we report results on four real world data sets, denoted as *ER_692*, *Arabidopsis*, *Leukemia* and *hereditarybc*, which are preprocessed from breast cancer data and gene expression networks. We refer to [81] for detailed information about those data sets.

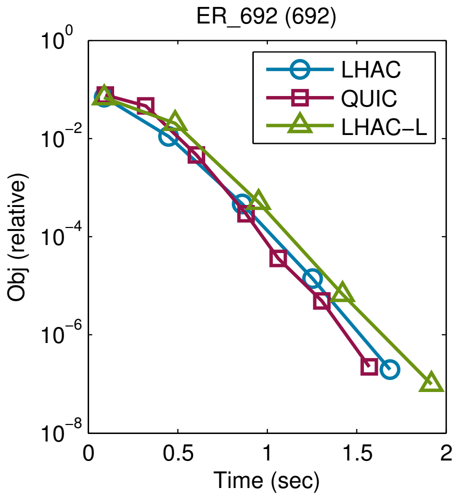
We set the regularization parameter $\lambda = 0.5$ for all experiments as suggested in [81]. The plots presented in Figure 5.5 show that LHAC and LHAC-L is almost twice as fast as QUIC, in the two largest data sets Leukemia and hereditarybc (see Figure 5.5(c) and 5.5(d)). In the other two smaller data sets the results are less clear-cut, but all of the methods solve the problems very fast and the performance of LHAC is comparable to that of QUIC. The performances of LHAC and LHAC-L are fairly similar in all experiments. Again we should note that with the sufficient decrease condition proposed in Algorithm 15 we are able to establish the global convergence rate, which has not been shown in the case of Armijo line search.



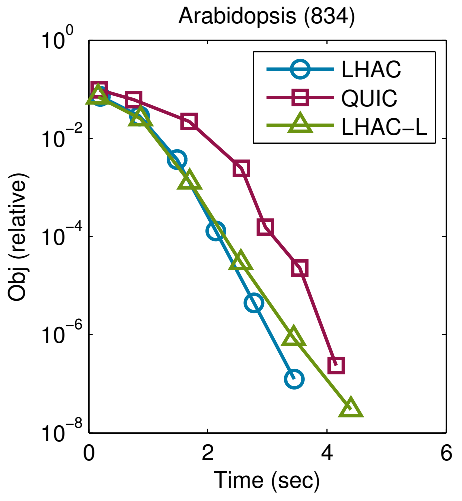
(a) $S : 1255 \times 1255$



(b) $S : 1869 \times 1869$

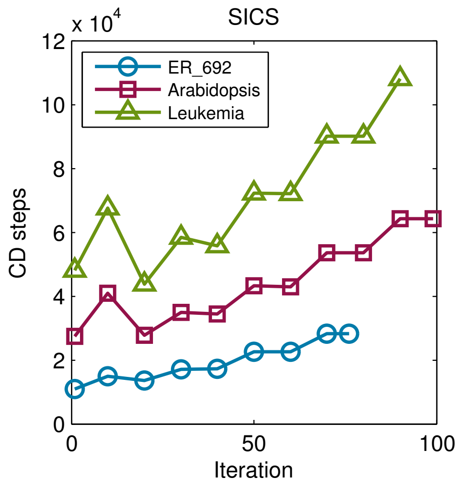


(c) $S : 692 \times 692$

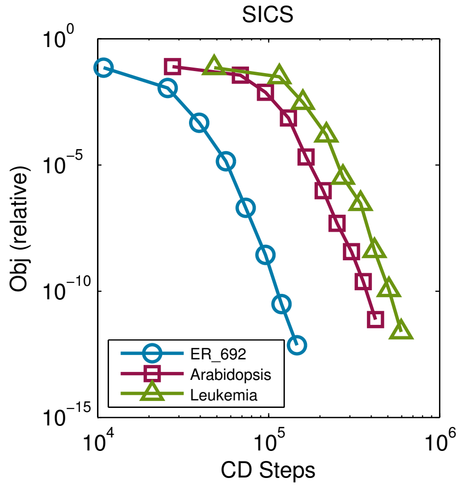


(d) $S : 834 \times 834$

Figure 5.5: Convergence plots on SICS (the y-axes on log scale).



(a) The number of coordinate descent steps on subproblems increases as iterates move towards optimality



(b) Both axes are in log scale. Change of objective w.r.t. the number of coordinate descent steps.

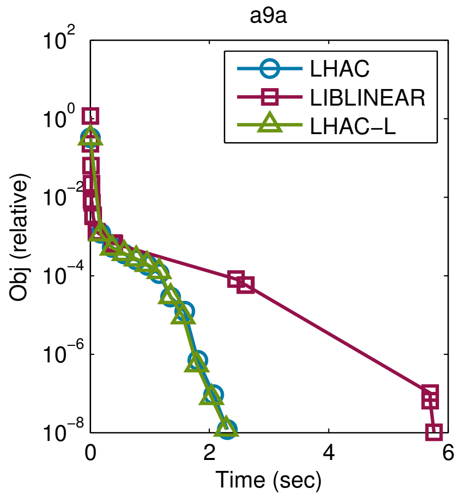
Figure 5.6: RCD step count of LHAC on different SICS data sets.

Data set	#features p	#instances N	#non-zeros	Description
a9a	123	32561	451592	'census Income' dataset.
epsilon	2000	100000	200000000	PASCAL challenge 2008.
gisette	5000	6000	29729997	handwritten digit recognition.
slices	385	53500	20597500	CT slices location prediction.

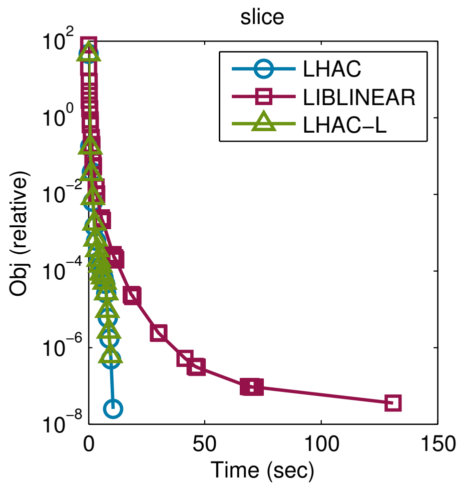
Table 5.4: Data statistics in sparse logistic regression experiments.

Sparse Logistic Regression The objective function of sparse logistic regression is given by

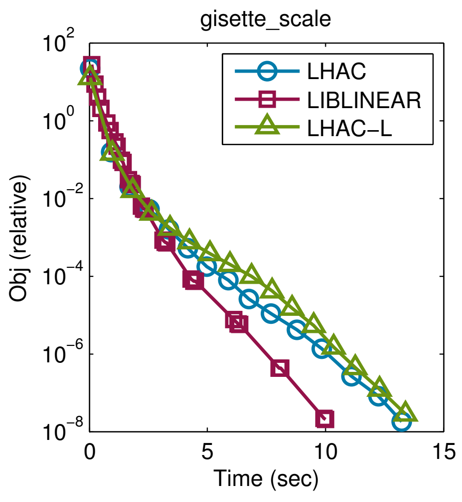
$$F(w) = \lambda\|w\|_1 + \frac{1}{N}\sum_{n=1}^{N}\log(1+\exp(-y_n \cdot w^T x_n))$$



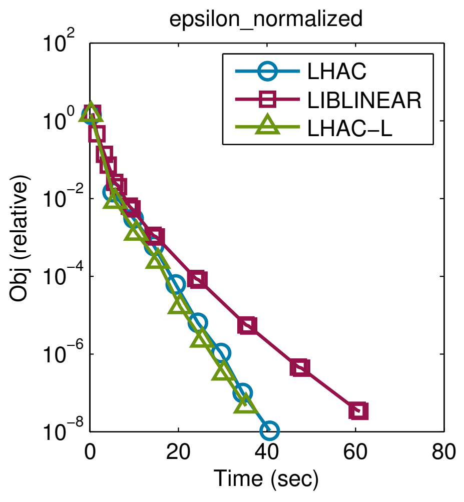
(a) a9a ($p = 123, N = 32561$)



(b) slices ($p = 385, N = 53500$)



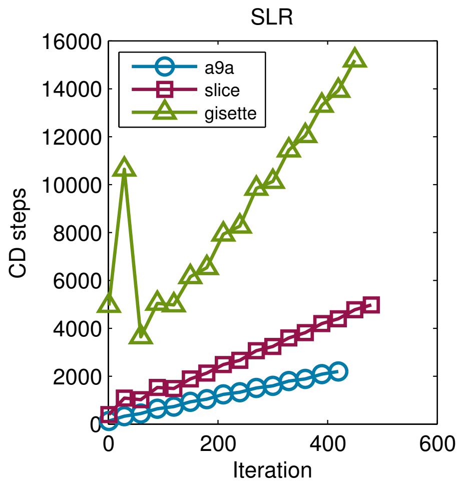
(c) gisette ($p = 5000, N = 6000$)



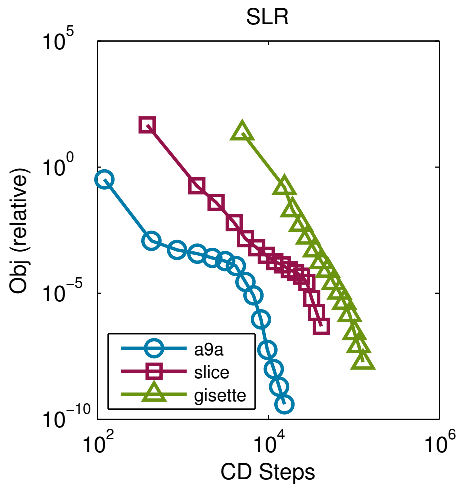
(d) epsilon ($p = 2000, N = 100000$)

Figure 5.7: Convergence plots on SLR (the y-axes on log scale).

where $L(w) = \frac{1}{N}\sum_{n=1}^{N}\log(1+\exp(-y_n \cdot w^T x_n))$ is the average logistic loss function and $\{(x_n, y_n)\}_{n=1}^{N} \in (\mathbb{R}^p \times \{-1, 1\})$ is the training set. The number of instances in the training set and the number of features are denoted by N and p respectively. Note that the evaluation of F requires $O(pN)$ flops and to compute the Hessian requires



(a) The number of coordinate descent steps on subproblems increases as iterates move towards optimality



(b) Both axes are in log scale. Change of objective w.r.t. the number of coordinate descent steps.

Figure 5.8: RCD step count of LHAC on different SLR data sets.

$O(Np^2)$ flops. Hence, we chose such training sets for our experiment with N and p large enough to test the scalability of the algorithms and yet small enough to be completed on a workstation.

We report results of SLR on four data sets downloaded from UCI Machine Learning repository [82], whose statistics are summarized in Table 5.4. In particular, the first data set is the well-known UCI Adult benchmark set *a9a* used for income classification, determining whether a person makes over \$50K/yr or not, based on census data; the second one we use in the experiments is called *epsilon*, an artificial data set for PASCAL large scale learning challenge in 2008; the third one, *slices*, contains features extracted from CT images and is often used for predicting the relative location of CT slices on the human body; and finally we consider *gisette*, a handwritten digit recognition problem from NIPS 2003 feature selection challenge, with the feature set of size 5000 constructed in order to discriminate between two confusable handwritten digits: the four and the nine.

The results are shown in Figure 5.7. In most cases LHAC and LHAC-L outperform LIBLINEAR. On data set *slice*, LIBLINEAR experiences difficulty in convergence which results in LHAC being faster by an order of magnitude. On the largest data set *epsilon*, LHAC and LHAC-L is faster than LIBLINEAR by about one third and reaches the same precision. Finally we note that the memory usage of LIBLINEAR is more than doubled compared to that of LHAC and LHAC-L, as we observed in all the experiments and is particularly notable on the largest data set *epsilon*.

Chapter 6

Conclusions

Machine learning has attracted lots of interests in recent years due to both advances in the active research community and wide applications through the industry. While much progress has been made, designing scalable and efficient algorithms for complex and large-scale learning, with theoretically sound guarantee, is still a challenging problem. Optimization thus serves a critical role to the success of machine learning in the real-world application, as the general trend moving towards big data applications, e.g., in distributed system, and complex learning objectives, e.g., non-smoothness in structured learning, deep neural network, etc.

This work contributes to the latest research on the analysis of global convergence rate of inexact proximal quasi-Newton framework that is applicable to a broad class of machine learning and signal processing problems. The work shows that randomized coordinate descent, among other subproblem methods, can be used effectively to find inexact quasi-Newton directions, which guarantee sub linear convergence rate of the algorithm, in expectation. This is the first global convergence rate result for an algorithm that uses coordinate descent to inexactly optimize subproblems at each iteration. This globally convergent framework enables, as proposed in the work, a fast, memory- and communication-efficient algorithm that directly addresses the big data cases when both N (samples) and n (features) are large and that combines first- and second-order information within one single optimization algorithm for fast convergence in practice.

The framework does not rely on or exploit the accuracy of second order information, and hence we do not obtain fast local convergence rates. We also do not assume strong convexity of our objective function, hence a sublinear conference rate is the best global rate we can hope to obtain. In [30] an accelerated scheme related to our framework is studied and an optimal sublinear convergence rate is shown, but the assumptions on the Hessian approximations are a lot stronger in [30] than in this work and appear to be impractical, hence the accelerated method is not as widely applicable. The framework studied in this work covers several existing efficient algorithms for large scale sparse optimization. However, to provide convergence rates we had to depart from some standard techniques, such as line-search, replacing it instead by a prox-parameter updating mechanism with a trust-region-like sufficient decrease condition for acceptance of iterates. We also use randomized coordinate descent instead of a cyclic one. We demonstrated that this modified framework is, nevertheless, very effective in practice and is competitive with state-of-the-art specialized methods.

Bibliography

[1] Nesterov, Y. Gradient methods for minimizing composite objective function. *CORE report* (2007).

[2] Beck, A. & Teboulle, M. A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems. *SIAM Journal on Imaging Sciences* **2**, 183–202 (2009).

[3] Wright, S. J., Nowak, R. D. & Figueiredo, M. A. T. Sparse reconstruction by separable approximation. *Trans. Sig. Proc.* **57**, 2479–2493 (2009).

[4] Yuan, G.-X., Ho, C.-H. & Lin, C.-J. An improved GLMNET for l1-regularized logistic regression and support vector machines. *National Taiwan University* (2011).

[5] Friedman, J., Hastie, T. & Tibshirani, R. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software* **33**, 1–22 (2010).

[6] Friedman, J., Hastie, T. & Tibshirani, R. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics Oxford England* **9**, 432–41 (2008).

[7] Scheinberg, K. & Rish, I. SINCO - a greedy coordinate ascent method for sparse inverse covariance selection problem. *tech. rep.* (2009).

[8] Scheinberg, K., Ma, S. & Goldfarb, D. Sparse inverse covariance selection via alternating linearization methods. *NIPS* (2010).

[9] Shalev-Shwartz, S. & Tewari, A. Stochastic methods for l1 regularized loss minimization. *ICML* 929–936 (2009).

[10] Koh, K., Kim, S. & Boyd, S. An interior-point method for large-scale l1-regularized logistic regression. *Journal of Machine learning research* **8**, 1519–1555 (2007).

[11] Tang, X., Qin, Z., Akrotirianakis, I. & Chakraborty, A. Hiclass - an efficient hierarchical classification system. Tech. Rep. (2012).

[12] Tang, X., Akrotirianakis, I. & Chakraborty, A. Distributed learning algorithm for short-term load forecasting. Tech. Rep. (2012).

[13] Qin, Z., Tang, X., Akrotirianakis, I. & Chakraborty, A. Hipad - a hybrid interior-point alternating direction algorithm for knowledge-based svm and feature selection. In Pardalos, P. M., Resende, M. G., Vogiatzis, C. & Walteros, J. L. (eds.) *Learning and Intelligent Optimization*, Lecture Notes in Computer Science, 324–340 (Springer International Publishing, 2014). URL `http://dx.doi.org/10.1007/978-3-319-09584-4_28`.

[14] Yuan, M. & Lin, Y. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* **68**, 49–67 (2006).

[15] Vapnik, V. N. & Vapnik, V. *Statistical learning theory*, vol. 1 (Wiley New York, 1998).

[16] Conn, A. R., Scheinberg, K. & Vicente, L. N. *Introduction to derivative-free optimization*, vol. 8 (Siam, 2009).

[17] Hsieh, C.-J., Sustik, M., Dhilon, I. & Ravikumar, P. Sparse inverse covariance matrix estimation using quadratic approximation. *NIPS* (2011).

[18] Olsen, P. A., Oztoprak, F., Nocedal, J. & Rennie, S. J. Newton-Like Methods for Sparse Inverse Covariance Estimation (2012).

[19] Byrd, R., Chin, G., Nocedal, J. & Oztoprak, F. A family of second-order methods for convex l1-regularized optimization. *tech. rep.* (2012).

[20] Scheinberg, K. & Tang, X. Practical inexact proximal quasi-newton method with global complexity analysis. *arXiv preprint arXiv:1311.6547* (2013). URL `http://arxiv.org/abs/1311.6547`. `arXiv:1108.0775v2`.

[21] Nocedal, J. & Wright, S. J. *Numerical Optimization*. Springer Series in Operations Research (Springer, New York, NY, USA, 2006), 2nd edn.

[22] Schmidt, M. W., Berg, E., Friedlander, M. P. & Murphy, K. P. Optimizing costly functions with simple constraints: A limited-memory projected quasi-newton algorithm. In *International Conference on Artificial Intelligence and Statistics*, None (2009).

[23] Schmidt, M., Roux, N. L. & Bach, F. Supplementary material for the paper convergence rates of inexact proximal-gradient methods for convex optimization. In *NIPS* (2011).

[24] Nesterov, Y. E. & Polyak, B. T. Cubic regularization of Newton method and its global performance. *mathprog* **108**, 177205 (2006).

[25] Cartis, C., M., G. N. I. & L., T. P. Evaluation complexity of adaptive cubic regularization methods for convex unconstrained optimization. *Optimization Methods and Software* **27**, 197–219 (2012).

[26] Nesterov, Y. E. Introductory lectures on convex optimization **87**, xviii+236 (2004). A basic course.

[27] Byrd, R., Nocedal, J. & Oztoprak, F. An inexact successive quadratic approximation method for convex l-1 regularized optimization. Tech. Rep. (2013).

[28] Lee, J. D., Sun, Y. & Saunders, M. A. Proximal newton-type methods for minimizing convex objective functions in composite form. In *SIAM Journal on Optimization* (2014).

[29] Becker, S. & Fadili, J. A quasi-newton proximal splitting method. In Pereira, F., Burges, C., Bottou, L. & Weinberger, K. (eds.) *Advances in Neural Information Processing Systems 25*, 2618–2626 (Curran Associates, Inc., 2012). URL `http://papers.nips.cc/paper/4523-a-quasi-newton-proximal-splitting-method.pdf`.

[30] Jiang, K., Sun, D. & Toh, K.-C. An inexact accelerated proximal gradient method for large scale linearly constrained convex SDP. *SIAM Journal on Optimization* **3**, 10421064 (2012).

[31] Richtárik, P. & Takáč, M. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming* (2012). URL `http://link.springer.com/10.1007/s10107-012-0614-z`.

[32] Yuan, G. & Ho, C. An improved GLMNET for l1-regularized logistic regression and support vector machines. *National Taiwan University* (2011).

[33] Tseng, P. & Yun, S. A coordinate gradient descent method for nonsmooth separable minimization,. *Mathematical Programming* **117**, 387–423 (2009).

[34] Dumais, S. & Chen, H. Hierarchical classification of web content. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, 256–263 (ACM, 2000).

[35] Cai, L. & Hofmann, T. Hierarchical document categorization with support vector machines. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, 78–87 (ACM, 2004).

[36] McCallum, A., Rosenfeld, R., Mitchell, T. & Ng, A. Improving text classification by shrinkage in a hierarchy of classes. In *Proceedings of the Fifteenth International Conference on Machine Learning*, 359–367 (Citeseer, 1998).

[37] Zimek, A., Buchwald, F., Frank, E. & Kramer, S. A study of hierarchical and flat classification of proteins. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* **7**, 563–571 (2010).

[38] Crammer, K. & Singer, Y. On the algorithmic implementation of multiclass kernel-based vector machines. *The Journal of Machine Learning Research* **2**, 265–292 (2002).

[39] Dekel, O., Keshet, J. & Singer, Y. Large margin hierarchical classification. In *Proceedings of the twenty-first international conference on Machine learning*, 27 (ACM, 2004).

[40] Zhou, D., Xiao, L. & Wu, M. Hierarchical classification via orthogonal transfer (2010).

[41] Cesa-Bianchi, N., Gentile, C. & Zaniboni, L. Hierarchical classification: combining bayes with svm. In *Proceedings of the 23rd international conference on Machine learning*, 177–184 (ACM, 2006).

[42] Barutcuoglu, Z., Schapire, R. & Troyanskaya, O. Hierarchical multi-label prediction of gene function. *Bioinformatics* **22**, 830 (2006).

[43] Lewis, D. D., Yang, Y., Rose, T. & Li, F. Rcv1: A new benchmark collection for text categorization research. *JMLR* **5**, 361–397 (2004).

[44] Hahn, H., Meyer-Nieberg, S. & Pickl, S. Electric load forecasting methods: Tools for decision making. *European Journal Of Operational Research* **199**, 902–907 (2009). URL `http://linkinghub.elsevier.com/retrieve/pii/S0377221709002094`.

[45] Fan, S. & Chen, L. Short-Term Load Forecasting Based on an Adaptive Hybrid Method. *IEEE Transactions on Power Systems* **21**, 392–401 (2006). URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1583738`.

[46] Hor, C. L., Watson, S. J. & Majithia, S. Analyzing the Impact of Weather Variables on Monthly Electricity Demand. *IEEE Transactions on Power Systems* **20**, 2078–2085 (2005). URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1525139`.

[47] Espinoza, M., Joye, C., Belmans, R. & DeMoor, B. Short-Term Load Forecasting, Profile Identification, and Customer Segmentation: A Methodology Based on Periodic Time Series. *IEEE Transactions on Power Systems* **20**, 1622–1630 (2005). URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1490617`.

[48] Taylor, J. W., De Menezes, L. M. & McSharry, P. E. A comparison of univariate methods for forecasting electricity demand up to a day ahead. *International Journal of Forecasting* **22**, 1–16 (2006). URL `http://eprints.maths.ox.ac.uk/281/`.

[49] González-Romera, E., Jaramillo-Morán, M. A. & Carmona-Fernández, D. Monthly Electric Energy Demand Forecasting Based on Trend Extraction. *IEEE Transactions on Power Systems* **21**, 1946–1953 (2006).

[50] Bruhns, A., Deurveilher, G. & Roy, J.-s. A Non-Linear Regression Model for Mid-Term Load Forecasting and Improvements in Seasonality. In *15th Power Systems Computation Conference*, August, 22–26 (Citeseer, 2005). URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.124.891&amp;rep=rep1&amp;type=pdf`.

[51] Ferreira, V. & Alves da Silva, A. Toward estimating autonomous neural network-based electric load forecasters. *Power Systems, IEEE Transactions on* **22**, 1554 –1562 (2007).

[52] Elattar, E. E., Goulermas, J. & Wu, Q. H. Electric Load Forecasting Based on Locally Weighted Support Vector Regression (2010). URL `http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5406166`.

[53] Chen, B.-j., Chang, M.-w. & Lin, C.-j. Load Forecasting Using Support Vector Machines : A Study on EUNITE Competition 2001. *Power* **19**, 1821–1830 (2004). URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1350819`.

[54] Bodyanskiy, Y., Popov, S. & Rybalchenko, T. Multilayer Neuro-fuzzy Network for Short Term Electric Load Forecasting. In Hirsch, E. A., Razborov, A. A., Semenov, A. & Slissenko, A. (eds.) *Third International Computer Science Symposium CSR2008*, 339–348 (Springer, 2008). URL `http://www.springerlink.com/content/y46125v1651tt212/`.

[55] Lin, C.-J. L. C.-J., Chen, C.-H. C. C.-H. & Lin, C.-T. L. C.-T. A Hybrid of Cooperative Particle Swarm Optimization and Cultural Algorithm for Neural Fuzzy Networks and Its Prediction Applications (2009). URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4717248`.

[56] Vapnik, V. N. *The Nature of Statistical Learning Theory*, vol. 8 of *Statistics for Engineering and Information Science* (Springer, 1995). URL `http://portal.acm.org/citation.cfm?id=211359`.

[57] Kohonen, T. The self-organizing map. *Proceedings of the IEEE* **78**, 1464–1480 (1990). URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=58325`.

[58] Takens, F. Detecting strange attractors in turbulence. *Dynamical systems and turbulence Warwick 1980* **898**, 366–381 (1981). URL `http://www.springerlink.com/index/B254X77553874745.pdf`.

[59] Ding, C., He, X. & Simon, H. D. On the equivalence of nonnegative matrix factorization and spectral clustering. *Proc SIAM Data Mining Conf* **44**, 606610 (2005). URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.76.9476&rep=rep1&type=pdf`.

[60] Shalev-shwartz, S., Singer, Y., Srebro, N. & Cotter, A. Pegasos: Primal estimated sub-gradient solver for svm. *Corpus* **127**, 807–814 (2007). URL `http://en.scientificcommons.org/56847611`.

[61] Sun, L., Ji, S. & Ye, J. Canonical correlation analysis for multilabel classification: A least-squares formulation, extensions, and analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **33**, 194–200 (2011).

[62] Chen, Y. & Lin, H. Feature-aware Label Space Dimension Reduction for Multi-label Classification. *Advances in Neural Information Processing Systems* 1538–1546 (2012). URL `http://machinelearning.wustl.edu/mlpapers/paper_files/NIPS2012_0728.pdf`.

[63] Hsu, D., Kakade, S., Langford, J. & Zhang, T. Multi-Label Prediction via Compressed Sensing. *NIPS* 1–14 (2009). URL `http://arxiv.org/abs/0902.1284http://papers.nips.cc/paper/3824-multi-label-prediction-via-compressed-sensing-supplemental.zip`. `0902.1284`.

[64] Zhang, Y. & Schneider, J. Maximum margin output coding. *Proceedings of the 29th International Conference on Machine Learning* 1575–1582 (2012). URL `http://arxiv.org/abs/1206.6478`.

[65] Bi, W. & Kwok, J. Efficient Multi-label Classification with Many Labels. *... of the 30th International Conference on ...* **28** (2013). URL `http://jmlr.org/proceedings/papers/v28/bi13.pdfhttp://machinelearning.wustl.edu/mlpapers/papers/icml2013_bi13`.

[66] Gu, H., Gartrell, M., Zhang, L., Lv, Q. & Grunwald, D. Anchormf: Towards effective event context identification. In *Proceedings of the 22Nd ACM International Conference on Conference on Information &#38; Knowledge Management*, CIKM '13, 629–638 (ACM, New York, NY, USA, 2013). URL `http://doi.acm.org/10.1145/2505515.2505548`.

[67] Gu, H., Hang, H., Lv, Q. & Grunwald, D. Fusing text and frienships for location inference in online social networks. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2012 IEEE/WIC/ACM International Conferences on*, vol. 1, 158–165 (IEEE, 2012).

[68] Cesa-Bianchi, N., Gentile, C. & Zaniboni, L. Incremental algorithms for hierarchical classification. *The Journal of Machine Learning Research* **7**, 31–54 (2006).

[69] Bi, W. & Kwok, J. Mandatory leaf node prediction in hierarchical multilabel classification. *Advances in Neural Information Processing Systems* 10–12 (2012).

[70] Jenatton, R., Mairal, J., Obozinski, G., Bach, F. & Fr, I. Proximal Methods for Sparse Hierarchical Dictionary Learning. *Proceedings of the 27th International Conference on Machine Learning (ICML-10)* 487–494 (2010). URL `http://www.icml2010.org/papers/416.pdf`. `arXiv:0909.0844`.

[71] Zhao, P., Rocha, G. & Yu, B. The composite absolute penalties family for grouped and hierarchical variable selection. *The Annals of Statistics* 3468–3497 (2009).

[72] Kim, S. & Xing, E. Tree-guided group lasso for multi-task regression with structured sparsity. *Proceedings of the 27th International ...* (2010). URL `http://arxiv.org/abs/0909.1373http://machinelearning.wustl.edu/mlpapers/paper_files/icml2010_KimX10.pdf`. `0909.1373`.

[73] Conn, A. R., Scheinberg, K. & Toint, P. L. On the convergence of derivative-free methods for unconstrained optimization. *Approximation theory and optimization: tributes to MJD Powell* 83–108 (1997).

[74] Moré, J. J. & Wild, S. M. Benchmarking derivative-free optimization algorithms. *SIAM Journal on Optimization* **20**, 172–191 (2009).

[75] Schmidt, M., Kim, D. & Sra, S. Projected newton-type methods in machine learning. *Optimization for Machine Learning* 305 (2012).

[76] Tibshirani, R. Regression shrinkage and selection via the lasso. *J. Royal. Statist. Soc B.* **58**, 267–288 (1996).

[77] Liu, D. C. & Nocedal, J. On the limited memory BFGS method for large scale optimization. *Mathematical Programming, Series B* **45**, 503–528 (1989).

[78] Donoho, D. De-noising by soft-thresholding. *Information Theory, IEEE Transactions on* **41**, 613 –627 (1995).

[79] Yuan, G.-X., Chang, K.-W., Hsieh, C.-J. & Lin, C.-J. A comparison of optimization methods and software for large-scale l1-regularized linear classification. *JMLR* **11**, 3183–3234 (2010).

[80] Tseng, P. & Yun, S. A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming* **117**, 387–423 (2009).

[81] Li, L. & Toh, K.-C. An inexact interior point method for L1-regularized sparse covariance selection. *Mathematical Programming* **2**, 291–315 (2010).

[82] Bache, K. & Lichman, M. UCI machine learning repository (2013). URL `http://archive.ics.uci.edu/ml`.

Biography

Xiaocheng Tang obtained his PhD in the area of optimization from Department of Industrial and Systems Engineering (ISE) at Lehigh University. During his study at Lehigh, he has been actively engaged in analyzing and designing novel optimization solutions for large-scale problems arising from various applications including Machine Learning and Data Mining. His work in proximal quasi-newton type methods was presented at the NIPS Optimization workshop and was submitted to Mathematical Programming A. Apart from conducting research at school, he was also engaged in solving real-world problems with collaborations from various industrial research institutes including Siemens Research, IBM Research and Yahoo Labs, and his work there has turned into several papers and patents. Prior to study at Lehigh, he spent 4 years in a beautiful coastal city Hangzhou in China, where he earned his B.S. in Control Science and Engineering from Chu KoChen Honors College in Zhejiang University.

Xiaocheng was born and raised in Guiyang, which is a lovely southern city in China blessed by its pleasant weather, unique landscape called Karst topography, and the most famous/best wine in the country.