University of California

Los Angeles

Learning Bayesian Networks via the Alternating Direction Method of Multipliers

A thesis submitted in partial satisfaction

of the requirements for the degree

Master of Science in Statistics

by

Jinchao Li

2015

Abstract of the Thesis

Learning Bayesian Networks via the Alternating Direction Method of Multipliers

by

Jinchao Li

Master of Science in Statistics

University of California, Los Angeles, 2015

Professor Qing Zhou, Chair

This thesis develops a fast splitting method for estimating Gaussian Bayesian networks from observational data. In Bayesian network, the directed acyclic graph (DAG) structure is used to represent the causal interactions of the variables. Without using prior knowledge about the network structure, we learn the structure by cyclically augmenting new edges to the graph while forcing the structure to be a DAG. For the given structure we learn the optimal edge coefficients using Alternating Direction Method of Multipliers. Our approach is compared with the most recent method of CDDr, and the experiment result shows that our method outperforms CDDr significantly.

The thesis of Jinchao Li is approved.

Arash Ali Amini

Yingnian Wu

Qing Zhou, Committee Chair

University of California, Los Angeles

2015

To my mother . . .

who—among so many other things—

saw to it that I learned to touch-type

while I was still in elementary school

Table of Contents

List of Figures

List of Tables

Vita

2005-2009	B.S. (Electrical Engineering), Zhejiang University, Hangzhou, Zhejiang, China.
2009-2010	M.S. (Electrical Engineering), UCLA, Los Angeles, California.

CHAPTER 1

Introduction

A graphical model [Lau96] is a graph representation of the relation among a set of random variables, where nodes are used to represent these variables and edges are used to represent the conditional independence relations among them. It has become an extremely popular tool for modeling as it provides a principled approach to dealing with uncertainty through the use of probability theory, and an effective approach to coping with complexity through the use of graph theory.

Based on whether the edges are directed or not, we can classify graphical models as either directed or undirected. For instance, the Markov random field is a special type of undirected graphical models and it has been used in many applications in statistics and machine learning. In contrast, the Bayesian network is a typical type of directed graphical models, where the joint distribution of the nodes is factorized via a directed acyclic graph (DAG) [Dar09][KF09]. Bayesian networks are very useful for data analysis due to a number of reasons. For example, they can be used to learn causal relationships from data and can readily handle incomplete data; they facilitate the combination of prior knowledge and data; they are efficient methods for avoiding data overfitting [Hec95] in Bayesian network learning.

Due to the wide applications of Bayesian networks, the development of fast structure learning algorithms becomes important. However, this is difficult due to the fact that the number of possible DAGs grows super-exponentially with the number of nodes [Rob77]. A substantial amount of work has been devoted to structure learning of Bayesian networks, and can roughly be classified into three categories:

Scored-based The scored-based approach defines a scoring function on the space of DAGs, and searches for the structure that maximizes the score function. The score functions are usually constructed based on posterior probability of the network structure or using minimum description length [LB94, HGC95]. When searching for the optimal network structure in the space of DAGs, various deterministic search strategies [HGC95, CB02] have been proposed. Alternatively, network structure can be sampled from a Bayesian posterior distribution using Monte Carlo methods [Dar09, KF09, EW08, Zho11]. Overall, the score-based approach works well for small size Bayesian network, but it is computationally impractical for large networks.

Constraint-based The constraint-based approach detects edges in a DAG with repeated conditional independence tests under certain model assumptions. The well-known examples in this category are the Peter-Clark (PC) algorithm [SG91, KB07] and the MMPC algorithm [TBA06]. Compared with the scored-based approach, the constraint-based approach is often faster since the independence tests are computationally efficient. However, the assumptions used in the approach are strict, which is a drawback of this approach.

Hybrid The hybrid approach is a mix of a score-based method and a constraint-based method. It applies conditional independence tests to remove as many edges as possible, and then use a score-based method to search for the optimal structure in the reduced DAG space [TBA06, GMP12].

In recent years, more and more researchers become interested in sparse undirected graphical models, and proposed all types of estimation methods, one of which is to estimate the graphical structure by applying an ℓ_1 regularization term on the log-likelihood function [BGdN06, YL07, BGd08, FHT07], a method usually referred to as the graphical Lasso. This method converts the difficulty of enumerating all possible topologies to a convex optimization problem with a tuning parameter on the ℓ_1 penalty term, which controls the trade-off between data fidelity and sparsity. However, an ℓ_1-regularized approach for Bayesian network structure learning was developed very recently by Fu and Zhou [FZ13], and the optimization is done by a clockwise coordinate descent algorithm. Since the loss function in [FZ13] is concave after minimizing over noise variances, and DAG constraint is also non-convex, the search space is highly non-convex, and therefore we can only guarantee local optimal. Aragam and Zhou [AZ15] obtain a convex loss function via re-parametrization, and introduce concave penalization to promote sparse DAGs. Their method is quite fast, and has tested on graphs with thousands of nodes.

In this article, based on the estimation framework by [FZ13] and [AZ15], we propose a fast structure learning algorithm by incorporating the alternating direction method of multipliers in the optimization of the ℓ_1-penalized log-likelihood. The new method can be orders of magnitude faster than coordinate descent for both sparse and dense structures and for both high-dimensional and low-dimension data. As one type of the score-based approach, our implementation also gives new insights into accelerating score-based algorithms.

The organization of the rest of this paper is as follows: In Section 2, we give a short introduction of Bayesian network and formulate the problem in a convex form by re-parametrization. In Section 3, we give a brief overview of the Alternating Direction Method of Multiplier (ADMM), and apply the ADMM to our problem. In Section 4, we generate synthetic data and compare our algorithm with the CCDr in [AZ15].

CHAPTER 2

Problem Formulation

2.1 Background on Gaussian Bayesian Network

In a Bayesian network, the joint probability distribution of a set of random variables $\{X_1, \dots, X_p\}$ can be factorized as

$$p(X_1, \dots, X_p) = \prod_{i=1}^{p} p(X_i | \Pi_j^{\mathcal{G}}), \tag{2.1}$$

where $\Pi_i^{\mathcal{G}}$ denotes the set of parents of X_i. By utilizing the factorization form in (2.1), the structure of a Bayesian network can be represented by a DAG $\mathcal{G}(V, E)$, where $V = \{1, \dots, p\}$ denotes the set of nodes in the graph, and $E = \{(i, j) | X_i \in \Pi_j^{\mathcal{G}}\}$ denotes the set of directed edges from the parent node to the child node.

In the Gaussian Bayesian Network (GBN) $\mathcal{G}(V, E)$, the relationships among the random variables are modeled by linear regression as

$$X_j = \sum_{i \in \Pi_j^{\mathcal{G}}} \beta_{ij} X_i + \epsilon_j, \qquad j = 1, \dots, p$$

where $\epsilon_j \sim N(0, \sigma_j^2)$, and β_{ij} is the coefficient representing the edge from X_i to X_j . For analysis convenience, we also define $B = (\beta_{ij})^{p \times p}$ to be the coefficient matrix, where $\beta_{ij} = 0$ if $i \notin \Pi_j^{\mathcal{G}}$, and the vector $\sigma^2 = (\sigma_j^2)_{1 \times p}$ as the variance vector.

Given n independent and identically distributed observations of the Bayesian network $X = (x_1, x_2, \dots, x_p), x_j \in \mathbf{R}^n$, the negative log-likelihood can be expressed as

$$L(B, \sigma|X) = \sum_{j=1}^{p} \left[\frac{n}{2} \log(\sigma_j^2) + \frac{1}{2\sigma_j^2} \|x_j - X\beta_j\|^2 \right], \tag{2.2}$$

with the constraint that the graph $\mathcal{G}$ is acyclic. Note that (2.2) is a non-convex function, computationally it is hard to obtain the global optimal value. By re-defining $\rho_j = 1/\sigma_j$ and $\phi_{ij} = \beta_{ij}/\sigma_j$, (2.2) turns into a convex form as

$$L(\Phi, \rho|X) = \sum_{j=1}^{p} \left[-n \log(\rho_j) + \frac{1}{2} \|\rho_j x_j - X\phi_j\|^2 \right].$$

In most applications, the underlying Bayesian network structure is sparse. By adding a regularization term on the objective function, sparsity can be promoted and the estimated structure will be more interpretable. Furthermore, for cases where the number of parameters is large, or the size of data is relatively small, introducing the regularization term helps void over-fitting. Therefore, we are interested in the following problem:

$$\begin{aligned}(\hat{\Phi}, \hat{\rho}) &= \underset{\Phi,\rho}{\operatorname{argmin}} \{ L(\Phi, \rho|X) + n \sum_{i,j} p_\lambda(|\beta_{ij}|) : B \text{ is a DAG} \} \\ &= \underset{B,\sigma}{\operatorname{argmin}} \{ \sum_{j=1}^{p} \left[-n \log(\rho_j) + \frac{1}{2} \|\rho_j x_j - X\phi_j\|^2 \right] + n \sum_{i,j} p_\lambda(|\beta_{ij}|) \} : B \text{ is a DAG} \},\end{aligned} \tag{2.3}$$

where $p_\lambda(x)$ is the penalty function.

2.2 Choice of Penalty Function

The most popular choice of penalty is ℓ_1 (Lasso) penalty, which has the virtue of sparsity promotion and fast computation. Therefore, the regularization term can be simply formulated as $p'_\lambda(x) = \lambda|x|$. For estimation problems, people are always interested in their asymptotical properties as they manifest the estimators' consistency. It has been shown [FP04] that a good procedure should satisfy $o$racle properties, which include

- Identifies the true submodel, $\mathcal{A} = \{i : \hat{\beta}_i \neq 0\}$.

- The estimatior satisfies, $\sqrt{n}(\hat{\beta}_{\mathcal{A}} - \beta^*_{\mathcal{A}}) \to N(0, \Sigma^*)$, where Σ^* is the true covariance matrix.

However, ℓ_1 penalized estimation problem enjoys oracle properties only when a nontrivial condition is satisfied, and therefore under certain scenarios the ℓ_1 penalized problem is inconsistent. This can be circumvented by employing adaptive Lasso [Zou06], in which adaptive weights are used for penalization. To be more specific, $p'_\lambda(x_i)$ is chosen as $p'_\lambda(x_i) = \lambda\omega_i|x_i|$, where ω_i is chosen separately for different i's. For instance, we can choose $\omega_{ij} = \min\left(|\hat{\beta}^{(\mathrm{OLS})}_{ij}|^{-\gamma}, M^\gamma\right)$, where M is a large positive number, and $\hat{\beta}^{(\mathrm{OLS})}_{ij}$'s are the OLS estimates. With this definition, the objective function is

$$Q(\Phi, \rho) = \sum_{j=1}^{p} \left[-n\log(\rho_j) + \frac{1}{2}\|\rho_j x_j - X\phi_j\|^2 \right] + n\lambda \sum_{i,j} \omega_i |\phi_{ij}| \tag{2.4}$$

By redefining $\tilde{\phi}_{ij} = \omega_{ij}\phi_{ij}$ and $\tilde{x}_{hi} = x_{hi}/\omega_{ij}$ for $i \neq j$, the objective function turns into

$$Q(\tilde{\Phi}, \rho) = \sum_{j=1}^{p} \left[-n\log(\rho_j) + \frac{1}{2}\|\rho_j x_j - \tilde{X}\tilde{\phi}_j\|^2 \right] + n\lambda \sum_{i,j} |\tilde{\phi}_{ij}| \tag{2.5}$$

Therefore (2.4) with $w_{ij} = 1$ and (2.5) can be solved by the same algorithm. Since the main goal of this paper is developing fast algorithm, without loss of generality, we set $w_{ij} = 1$ for all i, j in the following part of the paper, and effectively use the ℓ_1 penalty $p_\lambda(\phi_{ij}) = \lambda|\phi_{ij}|$.

2.3 Objective Function Reformulation

Objective Function with Ordering Note that $Q(\Phi, \rho)$ can be decomposed as $Q(\Phi, \rho) = \sum_{j=1}^{p} Q_j(\phi_j, \rho_j)$, where

$$Q_j(\phi_j, \rho_j) = -n\log(\rho_j) + \frac{1}{2}\|\rho_j x_j - X\phi_j\|^2 + n\lambda\|\phi_j\|_1.$$

Then the problem yields the following form:

$$(\hat{\Phi}, \hat{\rho}) = \underset{\Phi,\rho}{\operatorname{argmin}}\{\sum_{j=1}^{p} Q_j(\phi_j, \rho_j) : \Phi \text{ is a DAG}\}, \tag{2.6}$$

where $\Phi = (\phi_{ij})_{p\times p}$, $\rho = (\rho_j)_{1\times p}$. Our original estimator can be obtained from

$$(\hat{B}, \hat{\sigma}) = \begin{cases} \hat{\beta}_{ij} = \hat{\phi}_{ij}/\hat{\rho}_j, & i \neq j \\ \hat{\beta}_{ij} = 0, & \\ \hat{\sigma}_j^2 = 1/\hat{\rho}_j^2, & j = 1, \ldots, p \end{cases}$$

Furthermore, given a sparsity pattern of ϕ_j (the parents of node j), the objective function reduces to

$$Q_j(\phi_j, \rho_j) = -n\log(\rho_j) + \frac{1}{2}\|\rho_j x_j - X_{\pi_j}(\phi_j)_{\pi_j}\|^2 + n\lambda\|(\phi_j)_{\pi_j}\|_1,$$

where π_j means the set of parents of node j, $(\phi_j)_{\pi_j}$ denotes the non-zero parts of ϕ_j, X_{π_j} denotes the stack of column vectors x_i's where $i \in \pi_j$. Therefore, given all the parents sets $\pi_j, j = 1, \ldots, p$, minimization of Q can be decomposed into p independent convex problems, and these problems can be solved independently. Therefore, parallel computation can also be implemented for this algorithm.

2.4 Choice of Penalty Parameters and Regularization Paths

The estimation result is highly determined by the value of λ. To be specific, as $\lambda \to 0$, the estimation goes back to the unpenalized maximum likelihood. Conversely, if $\lambda \to +\infty$, then $\hat{\Phi}(\lambda) \to 0$. Practically speaking, we have no clue about the optimal value of the penalty parameter λ. Thus, one way to address this issue is to compute over a sequence of estimates $\{\hat{\Phi}(\lambda_i), \hat{\rho}(\lambda_i)\}$ for $\lambda_i > \lambda_{i+1} > 0, i = 0, 1, \ldots, L$ and then apply model selection methods to choose the best λ. Practically, we can choose λ_0 so that $\hat{\Phi}(\lambda_0) = 0$, and decrease the value of λ_i in a log-scale. For those $L + 1$ pre-specified λ-values, the number of estimated edges increases when the value of λ decreases, and therefore more time is spent on computing the model with small λ. In implementation, we use the optimal values for the previous λ as the initial value for the current λ as a warm start.

Model Selection Criteria The information-theoretic model selection criteria include the Akaike Information Criteria AIC, Second-Order Akaike $\mathrm{AIC_c}$, and Bayes Information Criteria BIC, defined as follows [BA10]:

$$\mathrm{AIC} = -2L + 2k, \qquad \mathrm{AIC_c} = -2L + \frac{2nk}{n-k-1}, \qquad \mathrm{BIC} = -2L + k\log n, \tag{2.7}$$

where L is the log-likelihood of the ML estimate, n is the sample size, and k is the effective number of parameters. In our application, L is given by

$$L = \sum_{j=1}^{p} \left[-n\log(\rho_j) + \frac{1}{2}\|\rho_j x_j - X\phi_j\|^2 \right]$$

where $\{\rho, \Phi\}$ is the optimal solution of (2.6), and k is the number of structurally non-zero variables in the model. By evaluating the models using information-theoretic model selection criteria, we choose the one with the lowest score. Among all these criteria, AIC is known to perform poorly if n is small compared with parameter k, $\mathrm{AIC_c}$ is a correction to AIC for small n by placing a penalty on models with high complexity, and BIC intends to choose simpler models than AIC or $\mathrm{AIC_c}$ for large n. In this paper, we use BIC as the model selection criterion.

CHAPTER 3

Algorithm

Although the objective function can be re-parameterized into a convex form, the DAG constraint makes the problem non-convex again, and therefore traditional convex optimization techniques are not directly applicable. In this paper, we extend the algorithms in [FZ13, AZ15], and apply a cyclic block coordinate-descent algorithm to enforce the DAG constraint, while minimizing (2.3) given a DAG structure via the ADMM.

3.1 Algorithm overview

Instead of minimizing all the variables simultaneously, a block coordinate-descent algorithm performs optimization over a subset (a block) of the full variables while holding other variables constant, and cycles through the remaining variables. This procedure continues until convergence. This algorithm has been proved successful in different settings [FHHT07, WL08], and it works very well for cases where the update for a block of variables can be performed efficiently. At a high level, the proposed algorithm in our paper is a modified coordinate descent method. For each iteration, the algorithm consists of three steps:

1. Fix the value of Φ, and optimize over ρ.
2. Fix the value of ρ, and cycle through blocks $\{\phi_{ij}, \phi_{ji}\}$ for different $i, j = 1, \ldots, p$ with $i \neq j$ sequentially.

3. Given the DAG structure identified by Step 1-2, apply the ADMM to obtain the optimal value of Φ and ρ.

The algorithm iterates though Step 1-3, and terminates until some stopping criterion is met.

Step 1 Given ϕ_i, for each ρ_i, we need to solve the convex minimization problem (3.1)

$$\underset{\rho_i}{\text{minimize}} \quad -n\log(\rho_i) + \frac{1}{2}||\rho_i x_i - X\phi_i||^2 \tag{3.1}$$

to update $\rho_i, i = 1, \dots, p$. By taking derivative, we only need to solve

$$(x_i^T x_i)\rho_i^2 - (x_i^T X\phi_i)\rho_i - n = 0,$$

for which, the only non-negative solution is $\rho_i = \frac{-b+\sqrt{b^2-4ac}}{2a}$, where $a = \|x_i\|^2, b = -(x_i^T X\phi_i)$ and $c = -n$.

Step 2 Due to the DAG constraint, we know that ϕ_{ij} and ϕ_{ji} can not be nonzero in the same time. This means when we perform blockwise coordinate descent, ϕ_{ij} and ϕ_{ji} should be considered as a block. We check whether it introduces a cycle in the graph by adding the edge $X_i \to X_j$. If so, ϕ_{ji} is set to 0, and the minimization is taken over ϕ_{ij}. Otherwise, we proceed to check whether a cycle will be introduced by adding the edge $X_i \to X_j$. If it introduces a cycle, we set $\phi_{ij} = 0$, and minimize over ϕ_{ji}. If neither introduces a cycle, we choose the one with a smaller objective value. In general, we update ϕ_{ij} by

$$\phi_{ij}^* = \underset{\phi_{ij}}{\text{argmin}} \quad \frac{1}{2}\|\phi_{ij}x_i - (\rho_j x_j - \sum_{k\neq i} x_k\phi_{kj})\|^2 + n\lambda|\phi_{ij}|.$$

This is equivalent to solving

$$\begin{aligned}\phi_{ij}^* &= \underset{\phi_{ij}}{\text{argmin}} \quad \frac{1}{2}\left(\phi_{ij} - \frac{x_i^T(\rho_j x_j - \sum_{k\neq i} x_k\phi_{kj})}{\|x_i\|^2}\right)^2 + \frac{n\lambda}{\|x_i\|^2}|\phi_{ij}| \\ &= \underset{\phi_{ij}}{\text{argmin}} \quad \frac{1}{2}\left(\phi_{ij} - \frac{\rho_j x_i^T x_j + \phi_{ij}\|x_i\|^2 - x_i^T X\phi_j}{\|x_i\|^2}\right)^2 + \frac{n\lambda}{\|x_i\|^2}|\phi_{ij}|\end{aligned} \tag{3.2}$$

(3.2) is a standard soft-thresholding problem. Let us consider the problem

$$\phi_{ij}^* = \underset{\phi_{ij}}{\text{argmin}} \quad \frac{1}{2}(\phi_{ij} - u)^2 + t\,|\phi_{ij}|_1$$

Here, ϕ_{ij}^* can be obtained by a simple soft thresholding procedure:

$$\phi_{ij}^* = \begin{cases} u - t, & u \geq t \\ 0, & -t \leq u \leq t \cdot \\ u + t, & u \leq -t \end{cases}$$

Therefore, the solution to (3.2) can be obtained by substituting $t = \frac{n\lambda}{\|x_i\|^2}$ and $u = \frac{\rho_j x_i^T x_j + \phi_{ij}\|x_i\|^2 - x_i^T X \phi_j}{\|x_i\|^2}$.

Step 3 In Step 3, we obtain the optimal solution of (Φ, ρ) given the DAG structure from step 2. For this ℓ_1 penalized minimization problem, it can be solved by a series of monotone operator splitting methods, including the Douglas-Rachford Splitting Method[EB92], the Alternating Direction Method of Multipliers (ADMM) [BPC$^+$11], the Forward-Backward Method [Tse00] and etc. In this paper, we use the ADMM algorithm due to its popularity in ℓ_1 penalized problems, with details provided in the next subsection.

3.2 Alternating Direction Method of Multipliers

The method of Alternating Direction Method of Multipliers (ADMM) has a long history (as in [Gab83, Tse90, Tse91]), and has become increasingly popular for large-scale applications in image processing [Ess10], distributed optimization, statistical learning [BPC$^+$11], and some other areas. This algorithm is intended to blend the decomposability of dual ascent with the robustness of the method of multipliers.

3.2.1 Overview of Alternating Direction Method of Multipliers

Suppose the function $f(x)$ is convex and twice-differentiable, and the function $h(x)$ is convex and possibly non-differentiable. Let us consider the general problem

$$\begin{array}{ll} \underset{x,y}{\text{minimize}} & f(x) + h(y) \\ \text{subject to} & Ax + By = b, \end{array} \tag{3.3}$$

where $x \in \mathbf{R}^n$, $y \in \mathbf{R}^m$, $A \in \mathbf{R}^{p \times n}$, $B \in \mathbf{R}^{p \times m}$ and $b \in \mathbf{R}^p$. The augmented Lagrangian is defined as

$$\mathcal{L}_t(x, y, \nu) = f(x) + h(y) + \nu^T(Ax + By - b) + \frac{1}{2t}\|Ax + By - b\|_2^2,$$

where $t > 0$ is called the penalty parameter. The ADMM method intends to optimize over x and y sequentially, and then update ν. It can be written in the following form:

$$\begin{aligned} x^{k+1} &= \operatorname{argmin}_x \mathcal{L}_t(x, y^k, \nu^k) \\ y^{k+1} &= \operatorname{argmin}_y \mathcal{L}_t(x^{k+1}, y, \nu^k) \\ \nu^{k+1} &= \nu^k + \tfrac{1}{t}(Ax^{k+1} + By^{k+1} - b). \end{aligned} \tag{3.4}$$

Letting $z = t\nu$, instead of working on (3.4), we define

$$\tilde{\mathcal{L}}_t(x, y, z) = f(x) + h(y) + \frac{1}{2t}\|Ax + By - b + z\|_2^2,$$

and then the updates can be formulated as

$$\begin{aligned} x^{k+1} &= \operatorname{argmin}_x \tilde{\mathcal{L}}_t(x, y^k, z^k) \\ y^{k+1} &= \operatorname{argmin}_y \tilde{\mathcal{L}}_t(x^{k+1}, y, z^k) \\ z^{k+1} &= z^k + (Ax^{k+1} + By^{k+1} - b). \end{aligned} \tag{3.5}$$

In practice, this algorithm may converge slowly to high accuracy. However, it can converge to modest accuracy in a few iterations, and this is sufficient for many applications and especially for large-scale problems.

Choice of Penalty Parameter The penalty parameter ρ can be fixed (e.g., fixed as 1), or can vary for each iteration. The goal of using a varying penalty parameter is to make the performance less dependent of the initial choice of the penalty parameter, and improve the convergence. A simple method is to keep the primal and dual residual norms within a factor of $\mu > 1$ so as to make sure that both converge to zero, which can be described as follows [HYW00]:

$$t^{k+1} = \begin{cases} \tau^{incr} t^k & \text{if } \|r^k\|_2 < \mu \|s^k\|_2, \\ t^k / \tau^{\text{decr}} & \text{if } \|s^k\|_2 < \mu \|r^k\|_2, \\ t^k & \text{otherwise,} \end{cases} \tag{3.6}$$

where τ^{incr} and τ^{decr} are the parameters used to control the change of the penalty parameter. Typically we choose $\mu = 10$ and $\tau^{\text{incr}} = \tau^{\text{decr}} = 2$.

3.2.2 Convergence of ADMM

There are many existing convergence literatures for ADMM under different assumptions. The basic and general assumption is that the functions f and g are closed, proper and convex. This assumption guarantees that the updates of x^k and y^k in (3.4) are solvable. Another important assumption is that strong duality holds for the problem, which implies that when (x^*, y^*) attains the optimal solution, the dual variable ν^* also attains its optimal. The convergence of the algorithm can be described as follows:

- *R*esidual convergence: $r^k \to 0$ and $s^k \to 0$ as $k \to \infty$.
- *O*bjective convergene: $f(x^k) + g(y^k) \to p^*$ as $k \to \infty$, where p^* is the optimal value
- *D*ual variable convergence: $\nu \to \nu^*$ as $k \to \infty$, where ν^* is the optimal dual variable.

The converge proof can be found in references such as [Gab83, EB92]. Practically, since ADMM is a first order method, it converges slow to high accuracy. However, it converges very fast (only a few tens of iterations) to modest accuracy, which makes it practically very useful, and specially for applications that do not require very high accuracy.

3.2.3 Optimality Condition and Stopping Criterion

The optimality conditions for problem (3.3) can be written as

$$\text{primal feasibility}: \quad Ax^* + By^* - b = 0, \tag{3.7}$$

$$\text{dual feasibility}: \quad 0 \in \partial f(x^*) + A^T \nu^*, \tag{3.8}$$

$$0 \in \partial g(y^*) + B^T \nu^*, \tag{3.9}$$

where ∂ denotes the sub-differential operator, and if the function is differentiable, ∂ means the derivative. Define the *primal residual* at iteration k as

$$r^k = Ax^k + By^k - b. \tag{3.10}$$

Since x^{k+1} minimizes $\mathcal{L}_t(x, y^k, \nu^k)$, we have that

$$\begin{aligned} 0 &\in \partial f(x^{k+1}) + A^T \nu^k + \frac{1}{t} A^T (Ax^{k+1} + By^k - b) \\ &= \partial f(x^{k+1}) + A^T \left(\nu^k + \rho r^{k+1} + \frac{1}{t} B(y^k - y^{k+1}) \right) \\ &= \partial f(x^{k+1}) + A^T \nu^{k+1} + \frac{1}{t} A^T B \left(y^k - y^{k+1} \right). \end{aligned}$$

This means that $\frac{1}{t} A^T B (y^{k+1} - y^k) \in \partial f(x^{k+1}) + A^T \nu^{k+1}$. By utilizing the optimality condition of (3.8), we can define the *dual residual* at iteration $k+1$ as

$$s^{k+1} = \frac{1}{t} A^T B (y^{k+1} - y^k). \tag{3.11}$$

Similarly, since y^{k+1} minimizes $\mathcal{L}_t(x^k, y, \nu^k)$, we have that

$$\begin{aligned} 0 &\in \partial g(y^{k+1}) + B^T \nu^k + \frac{1}{t} B^T (Ax^{k+1} + By^{k+1} - b) \\ &= \partial g(y^{k+1}) + B^T \nu^k + \frac{1}{t} B^T r^{k+1} \\ &= \partial g(y^{k+1}) + B^T \nu^{k+1}. \end{aligned}$$

This means that $\{y^{k+1}, \nu^{k+1}\}$ always satisfy the optimality condition (3.9). Therefore, the primal and dual residuals can be defined as in (3.10) and (3.11) respectively.

Stopping Criteria With above definitions of residuals, a reasonable stopping criterion can be defined as

$$\|r^k\| \leq \epsilon^{\text{pri}} \quad \text{and} \quad \|s^k\| \leq \epsilon^{\text{dual}}, \tag{3.12}$$

where $\epsilon^{\text{pri}} > 0$ and ϵ^{dual} are feasibility tolerance for primal optimality and dual optimality condition, respectively. In [BPC+11], the tolerances are chosen with the combination of an absolute and relative criterion as

$$\begin{aligned}
\epsilon^{\text{pri}} &= \sqrt{p}\epsilon^{\text{abs}} + \epsilon^{\text{rel}} \max\{\|Ax^k\|_2, \|By^k\|_2, \|b\|_2\}, \\
\epsilon^{\text{dual}} &= \sqrt{n}\epsilon^{\text{abs}} + \epsilon^{\text{rel}}\|A^T\nu^k\|_2,
\end{aligned}$$

where $\epsilon^{\text{abs}} > 0$ is an absolute tolerance and $\epsilon^{\text{rel}} > 0$ is a relative tolerance. Typically, ϵ^{rel} is chosen as 10^{-3} or 10^{-4}.

3.3 ADMM Applied on the Inner Loop

Let $\mathcal{V}_{active}$ be the set of edges in a DAG and $\mathcal{A}(\Phi)$ be the edge set of the coefficient matrix Φ. In order to apply ADMM method to (2.4) restricted to the edge set $\mathcal{V}_{active}$, we reformulate the problem in a convex form as follows:

$$\begin{array}{ll}
\underset{\Phi,\Psi,\rho}{\text{minimize}} & \sum_{j=1}^{p} \left[-n\log(\rho_j) + \frac{1}{2}\|\rho_j x_j - X\phi_j\|^2\right] + n\sum_{i,j} p_\lambda(|\psi_{ij}|) + \delta_{\mathcal{V}_{active}}(\psi) \\
\text{subject to} & \phi = \psi
\end{array}, \tag{3.13}$$

where the indicator function $\delta_{\mathcal{V}_{active}}(\psi)$ is defined as

$$\delta_{\mathcal{V}_{active}}(\psi) = \begin{cases} 0 & \text{if } \mathcal{A}(\psi) \subset \mathcal{V}_{active}, \\ +\infty & \text{otherwise.} \end{cases} \tag{3.14}$$

Let us denote

$$f(\phi, \rho) = \sum_{j=1}^{p} \left(-n \log(\rho_j) + \frac{1}{2} \|\rho_j x_j - X\phi_j\|^2 \right) + \delta_{\mathcal{V}_{active}}(\phi),$$

and

$$g(\psi) = n \sum_{i,j} p_\lambda(|\psi_{ij}|) + \delta_{\mathcal{V}_{active}}(\psi).$$

Then $f(\phi, \rho)$ is convex and differentiable, and $g(\psi)$ is convex and non-differentiable. Apply ADMM updates as in (3.4), the iterations can be formulated as

$$\{\phi^{k+1}, \rho^{k+1}\} = \underset{\phi,\rho}{\operatorname{argmin}}\, f(\phi, \rho) + \frac{1}{2t} \|\phi - \psi^k + \eta^k\|_F^2, \tag{3.15}$$

$$\psi^{k+1} = \underset{\psi}{\operatorname{argmin}}\, g(\psi) + \frac{1}{2t} \|\phi^{k+1} - \psi + \eta^k\|_F^2, \tag{3.16}$$

$$\eta^{k+1} = \eta^k + (\phi^{k+1} - \psi^{k+1}). \tag{3.17}$$

The update of (3.17) is straightforward. For (3.15) and (3.16), since we already know the active set, we only apply the updates on the active set. The detailed minimization of (3.15) and (3.16) are discussed in details in what follows.

Proximal Operators The proximal operator prox_{tf} of f is defined by

$$\operatorname{prox}_{tf}(v) = \underset{x}{\operatorname{argmin}} \left(f(x) + \frac{1}{2t} \|x - v\|_2^2 \right).$$

This operator aims to find a point close to the minimizer of f while keeping it close to v. It is often referred to as an implicit gradient descent step with stepsize t (compared with the gradient descent method). In order to solve for (3.15) and (3.16), we have to evaluate two prox-operators: $\operatorname{prox}_{tf}(\phi, \rho)$ and $\operatorname{prox}_{tg}(\phi)$. If these two operators can be evaluated efficiently, then each iteration of ADMM can be computed easily.

Evaluation of $\operatorname{prox}_{tf}(\phi, \rho)$ Since $\{\phi_i, \rho_i\}$ and $\{\phi_j, \rho_j\}$ are not coupled for $i \neq j$, the problem (3.15) can be decomposed into p independent problems,

$$\underset{\phi_j,\rho_j}{\text{minimize}} \quad -n \log \rho_j + \frac{1}{2} \|\rho_j x_j - X\phi_j\|_2^2 + \frac{1}{2t} \|\phi_j - \psi_j^k + \eta_j^k\|_2^2 + \delta_{\pi_j}(\phi_j) \tag{3.18}$$

for $j = 1, \dots, p$. Denote by π_j the parents of node j, and let X_{π_j} be the stack of columns π_j of X. Solving (3.18) is equivalent to

$$\underset{(\phi_j)_{\pi_j}, \rho_j}{\text{minimize}} \quad -n \log \rho_j + \frac{1}{2} \|\rho_j x_j - X_{\pi_j} (\phi_j)_{\pi_j}\|_2^2 + \frac{1}{2t} \|(\phi_j)_{\pi_j} - (\psi_j)^k_{\pi_j} + (\eta_j)^k_{\pi_j}\|_2^2. \quad (3.19)$$

By taking partial derivatives over $\phi_{\pi_j j}$ and ρ_j, the results can be obtained by solving

$$-X^T_{\pi_j} \left(\rho_j x_j - X_{\pi_j} (\phi_j)_{\pi_j}\right) + \frac{1}{t} \left((\phi_j)_{\pi_j} - (\psi_j)^k_{\pi_j} + (\eta_j)^k_{\pi_j}\right) = 0, \quad (3.20)$$

$$\|x_j\|^2 \rho_j^2 - x_j^T X_{\pi_j} (\phi_j)_{\pi_j} \rho_j - n = 0. \quad (3.21)$$

Define $D(\pi_j; t) \triangleq (X^T_{\pi_j} X_{\pi_j} + \frac{1}{t} I_{\pi_j})^{-1}$. Then from (3.20), we have

$$\phi_j = D(\pi_j; t)(X^T_{\pi_j} x_j)\rho_j + \frac{1}{t} D(\pi_j; t) \left((\psi_j)^k_{\pi_j} - (\eta_j)^k_{\pi_j}\right) \quad (3.22)$$

By substituting (3.22) to (3.21), we can obtain

$$\left(\|x_j\|^2 - x_j^T X_{\pi_j} D(\pi_j; t) X^T_{\pi_j} x_j\right) \rho_j^2 - \frac{1}{t} x_j^T X_{\pi_j} D(\pi_j; t) \left((\psi_j)^k_{\pi_j} - (\eta_j)^k_{\pi_j}\right) \rho_j - n = 0 \quad (3.23)$$

Solve (3.23) for ρ_j, and substitute the result back to (3.22), we can obtain the solution for $(\phi_j)_{\pi_j}$. Note that we can precompute $X^T X$, and choose the corresponding elements to construct $\|x_j\|^2$, $X^T_{\pi_j} x_j$ and $X^T_{\pi_j} X_{\pi_j}$, thus avoiding repeated computation. For the inverse matrix $D(\pi_j; t) = (X^T_{\pi_j} X_{\pi_j} + \frac{1}{t} I_{\pi_j})^{-1}$, since the subset π_j usually is very small, the computation does not take much time.

Evaluation of $\mathrm{prox}_{tg}(\psi)$ If $p_\lambda(|\phi_{ij}|) = \lambda |\phi_{ij}|$, then prox-operator is a simple soft-thresholding. From the indicator function (3.14), we know that $\phi_{ij} = 0$ for $(i, j) \notin \mathcal{V}_{active}$. Otherwise, the result can be obtained by applying soft-thresholding for all $(i, j) \in \mathcal{V}_{active}$,

$$\psi_{ij} = \begin{cases} (\phi_{ij}^{k+1} + \eta_{ij}^k) - \lambda t n, & (\phi_{ij}^{k+1} + \eta_{ij}^k) > \lambda t n, \\ 0, & -\lambda t n \le (\phi_{ij}^{k+1} + \eta_{ij}^k) \le \lambda t n, \\ (\phi_{ij}^{k+1} + \eta_{ij}^k) + \lambda t n, & (\phi_{ij}^{k+1} + \eta_{ij}^k) < -\lambda t n. \end{cases}$$

3.4 Full Algorithm

The complete algorithm is given in Algorithm 1. Compared with CCDr in [AZ15], both algorithms run the cyclic coordinate descent method to obtain the active set. The difference is that we apply the ADMM to obtain the optimal solution given the active set, while [AZ15] still applies coordinate descent.

Algorithm 1 Coordinate Descent with ADMM Algorithm (CDAM)

```
Initialization:
    • Initial estimate $(\Phi^0, \rho^0)$, a sequence of penalty parameters $\lambda_0 > \lambda_1 > \cdots > \lambda_L$, error tolerance $\epsilon > 0$.
    • Normalize the data so that $\|x_j\|^2 = 1$ for all $j$'s, compute $X^T X$ and save the result for later usage.
For each $\lambda_i$:
    1. For $i > 0$, set $(\Phi^0(\lambda_i), \rho^0(\lambda_i)) = (\Phi(\lambda_{i-1}), \rho(\lambda_{i-1}))$
    2. Fix $\Phi$, and minimize (2.6) with respect to $\rho$ as in (3.1).
    3. Cycle through the $p(p-1)/2$ blocks $\{\phi_{ij}, \phi_{ji}\}$ for $i, j = 1, \dots, i \neq j$, minimizing with respect to each block as in (3.2), and identify the active edge set $\mathcal{V}_{active}$:
        (a) If $\phi_{ij} = 0$, then minimize (2.6) with respect to $\phi_{ji}$, and set $(\phi_{ij}, \phi_{ji}) = (0, \phi^*_{ji})$.
        (b) If $\phi_{ji} = 0$, then minimize (2.6) with respect to $\phi_{ij}$, and set $(\phi_{ij}, \phi_{ji}) = (\phi^*_{ij}, 0)$.
        (c) If neither of them apply, then choose the updates which leads to the smaller value.
    4. For the structure as identified in the active set, apply the ADMM on the following convex optimization problem over $(\Phi, \rho)$:
           minimizes  $\sum_{j=1}^{p} \left[-n \log(\rho_j) + \frac{1}{2}\|\rho_j x_j - X\phi_j\|^2\right] + n \sum_{i,j} p_\lambda(|\phi_{ij}|)$
           subject to  $\mathcal{A}(\Phi) \subset \mathcal{V}_{active}$        (3.24)
    5. Repeat (2-4), until the active set does not change. Save the current optimal values as $\left(\hat{\Phi}(\lambda_i), \hat{\rho}(\lambda_i)\right)$
Transform the final estimates $\left(\hat{\Phi}(\lambda_i), \hat{\rho}(\lambda_i)\right)$ back to the original parameter space $\left(\hat{B}(\lambda_i), \hat{\sigma}(\lambda_i)\right)$.
```

CHAPTER 4

Simulation

In [AZ15], the algorithm of CCDr has been compared with classic algorithms including the PC algorithm [SG91], the GES algorithm [Chi03], the HC algorithm [HGC95] and the MMHC algorithm [TBA06]. In this paper, the proposed algorithm CDAM is a modified version of the CCDr in [AZ15].The main difference between them is that step 4 in Algorithm 1. for the CDAM is replaced by coordinate descent in the CCDr. In this section, the experimental comparison is between the CDAM and the original CCDr algorithm. The data generation part is implemented using R and C, and the algorithm part is implemented using C++. For matrix operations, we use the C++ library *eigen*, which provides fast and versatile data structures/operations implementation for (sparse) matrices. All of the tests were performed on a mid 2014 Apple MacBookPro with a 2.5GHz Intel Core i7 processor and 16GB 1600MHz DDR3, running Mac OS X 10.10.1.

4.1 Pre-calculation

As described in Algorithm 1, we need to calculate $x_i^T x_j$ for $i, j, = 1, \ldots, p$, and use the values for multiple times. Pre-calculation of those values in the beginning of the algorithm and storing the values for later use can reduce time complexity significantly. Moreover, for the ADMM algorithm, in contrast to coordinate descent, we need to calculate $(X_{\pi_j}^T X_{\pi_j} + \frac{1}{t} I_{\pi_j})^{-1}$ for each active set. The set π_j usually is very small due to the sparsity in the estimated Bayesian network, so the inverse does not take much time. Furthermore, for our algorithm, we only need to identify the active set several times until the structure converges. Therefore, the inverse operations are usually evaluated only a few times, and thus increase the time complexity slightly.

4.2 Comparison on Random Graphs

In this section we provide detailed comparison between the performance of our algorithm and the CCDr algorithm in [AZ15]. In order to give a fair comparison, we adopt some settings in [AZ15]. The difference between our algorithm and the CCDr algorithm is the method used in the inner loop (step 4) given a fixed active set: we use the ADMM, while [AZ15] uses coordinate descent. Therefore, we compare both the inner loops and the full algorithm. For the comparison, note that in the ADMM algorithm, the parameter t was fixed to 2.0, and the λ was chosen as the same optimal value as that in the CCDr for the corresponding setting. The setting for the comparison includes the number of nodes (variables) p, the number of edges s_0, and number of random samples n. We generate random graphs using *pcalg* package (randomDAG, rmvDAG) with the following settings:

$$p \in \{50, 200\}, s_0/p \in \{0.2, 0.5, 1.0, 2.0\}, n/p \in \{0.2, 1, 5\}.$$

These parameters cover both low ($n > p$) and high ($p > n$) dimensional cases and a range of sparsity levels (s_0/p). In order to compare the ADMM and coordinate descent for our application, we record for each setting the running time for the inner loops (step 4 in Algorithm 1) and the total time of the CCDr and the CDAM algorithms. Table 4.1 and Table 4.2 show the results for these comparisons with $p = 50$ and $p = 200$ respectively. In particular, Fig.4.1 shows the convergence of one outer iteration for both algorithms. We see that the ADMM runs much faster than coordinate descent. When the problem size becomes larger, the advantage of the ADMM is even more significant. In terms of the full algorithm (including the common cyclic DAG check part), the CDAM can be twice faster. That's because

p	s_0/p	n/p	λ	Total Inner Iteration (s)			Total Time (s)		
				CCDr	CDAM	$\frac{\text{CCDr}}{\text{CDAM}}$	CCDr	CDAM	$\frac{\text{CCDr}}{\text{CDAM}}$
50	0.2	0.2	0.25	0.00388	0.00195	1.99	0.01383	0.01190	1.16
		1	0.055	0.00764	0.00191	3.99	0.01611	0.01038	1.55
		5	0.012	0.01992	0.00361	5.51	0.03766	0.02136	1.76
	0.5	0.2	0.22	0.00790	0.00285	2.78	0.02285	0.01779	1.28
		1	0.047	0.02080	0.00705	2.95	0.05307	0.03931	1.35
		5	0.01	0.03582	0.00891	4.02	0.08578	0.05887	1.46
	1.0	0.2	0.2	0.02263	0.01515	1.49	0.11941	0.11193	1.07
		1	0.045	0.03383	0.00746	4.53	0.07896	0.05260	1.50
		5	0.01	0.11521	0.02957	3.90	0.22858	0.14295	1.60
	2.0	0.2	0.2	0.01235	0.00413	2.99	0.03637	0.02815	1.29
		1	0.04	0.09827	0.03018	3.26	0.29442	0.22632	1.30
		5	0.01	0.08944	0.02568	3.48	0.23174	0.16799	1.38
Average				0.039	0.0115	3.38	0.1016	0.0741	1.37

Table 4.1: Small Graph: $p = 50$

the time spent on cyclic DAG check takes up to 20% - 60% of the full algorithm.

In the previous comparison, the value of λ is fixed (chosen as the optimal value) for each setting. Since in real applications, we have to run through different λ's, and choose the optimal one by a model selection method, the effect of different λ's on the efficiency of the algorithms is of great interest. Next, we aim to test the case of large sparse graphs with high-dimensional data: the number of nodes (variables) p is large, while the number of edges (connectivity of the variables) s_o is small. Besides, the number of samples n is small compared with the number of

p	s_0/p	n/p	λ	Total Inner Iteration (s)			Total Time (s)		
				CCDr	CDAM	$\frac{\text{CCDr}}{\text{CDAM}}$	CCDr	CDAM	$\frac{\text{CCDr}}{\text{CDAM}}$
200	0.2	0.2	0.075	1.0582	0.01979	53.46	2.1538	1.1153	1.93
		1	0.016	1.2932	0.01702	76.00	2.0387	0.7625	2.67
		5	0.0033	2.6886	0.04830	55.66	4.4193	1.7790	2.48
	0.5	0.2	0.07	1.0830	0.03203	33.81	2.5953	1.5444	1.68
		1	0.015	3.2306	0.05646	57.22	5.3544	2.1803	2.46
		5	0.003	2.6356	0.04915	53.63	4.1407	1.5543	2.66
	1.0	0.2	0.065	1.8535	0.06204	29.88	4.9813	3.1899	1.56
		1	0.014	3.6196	0.08745	41.39	7.1491	3.6170	1.98
		5	0.003	5.7699	0.12874	44.98	9.8960	4.2544	2.33
	2.0	0.2	0.058	3.9474	0.17941	22.00	12.997	9.2289	1.41
		1	0.013	6.7901	0.20160	33.68	14.827	8.2388	1.80
		5	0.0028	12.599	0.32794	38.41	23.917	11.648	2.05
	Average			3.8807	0.1008	38.50	7.8725	4.0927	1.92

Table 4.2: Small Graph: $p = 200$

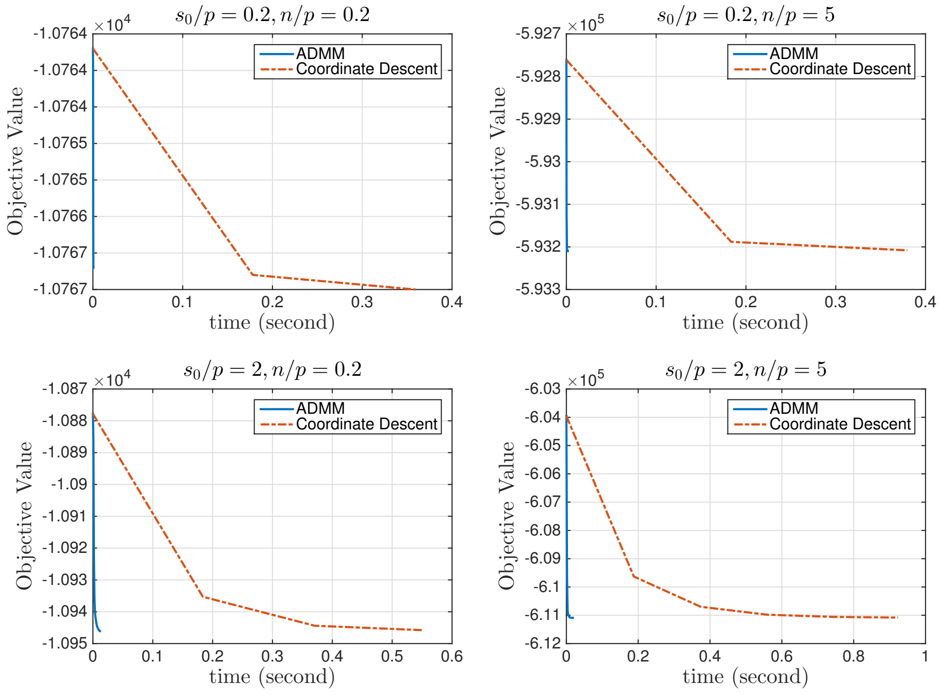

Figure 4.1: Convergene Rate Comparison

nodes. The settings are as follows:

$$p \in \{500, 1000\}, s_0/p = 0.2, n/p = 0.2.$$

From Table 4.3, we can see that for the same setting, a large λ puts large penalty on the structure, and therefore few edges are introduced in the graph. This makes structure identification procedure faster. Besides, with the increase of problem size, the advantage of the ADMM becomes more significant with two to three orders of magnitude faster than coordinate descent. The time spent on the ADMM iterations is so small that it is totally ignorable compared to the time of cyclic DAG check. This essentially makes the time complexity of our algorithm only depends on the cyclic DAG check part.

Remark For a given active set, the ADMM outperforms coordinate descent significantly. This is more obvious when the number of edges is large (it can

p	λ	Total Inner Iteration (s)			Total Time (s)			Est. Edges
		CCDr	CDAM	$\frac{\text{CCDr}}{\text{CDAM}}$	CCDr	CDAM	$\frac{\text{CCDr}}{\text{CDAM}}$	
500	0.04	13.631	0.04085	333.66	22.440	8.8499	2.54	41
	0.033	18.927	0.05257	360.03	29.427	10.552	2.79	122
	0.025	36.858	0.52228	70.57	182.44	146.10	1.25	1382
1000	0.05	20.696	0.05991	345.43	49.619	28.983	1.71	3
	0.018	204.71	0.24044	851.37	360.95	156.49	2.30	236
	0.015	259.26	0.62561	414.42	798.42	539.78	1.48	1417

Table 4.3: Large Graph

be at least 10x faster). The bottleneck for the full algorithm is the active set identification part, which requires check for the cycles when an edge is added or reversed. This will be rather slow for large graphs. The focus of further work should be put on how to optimize this step.

4.3 Comparison on Real Networks

While in this previous section, we tested our algorithm based on random structures, those random graphs may not be representative of real networks. Therefore, in this section, we use the network structures from the Bayesian Network Repository [1], which is used as a benchmark for structure learning methods. To be specific, we load the structure using the *bnlearn* package, and use rmvDAG in *pcalg* package to generate the data. We focus on high-dimensional estimation and fix the number of samples as $n = 50$. The networks *hailfinder, win95pts, pathfinder, andes* are tested. Table 4.4 shows the results for the comparisons with

[1] http://www.cs.huji.ac.il/site/labs/compbio/Repository/

Name	p	s_0	λ	Total Inner Iteration (s)			Total Time (s)		
				CCDr	CDAM	$\frac{\text{CCDr}}{\text{CDAM}}$	CCDr	CDAM	$\frac{\text{CCDr}}{\text{CDAM}}$
hailfinder	56	66	0.12	0.2244	0.0203	11.03	0.2767	0.0726	3.81
win95pts	76	112	0.05	2.4765	0.1737	14.26	3.5301	1.2274	2.87
pathfinder	109	195	0.11	9.4376	0.4894	19.28	12.5235	3.5753	3.50
andes	223	338	0.1414	5.3723	0.5120	10.49	17.2736	12.4134	1.39
hepar2	70	123							
munin1	186	273							
pigs	441	592							
diabetes	413	602							

Table 4.4: Real Network

those real networks. We see that CDAM outperforms CCDr in the same scale as in the random graphs.

CHAPTER 5

Discussion

In this paper, we have shown that by applying the ADMM algorithm on the inner loop of CCDr, the speed of the new algorithm CDAM can be better than doubled in various cases. When the scale of the network increases, the improvement can be more manifest.

For the future work, in addition to applying the ADMM method, there are some other commonly used first order splitting methods including Douglas Rachford on the primal, primal-dual Douglas Rachford, and etc.. Although those methods have the same order of convergence rates in general, the real convergence rates vary case by case, and therefore they are still worth trying as an alternative method for the ADMM algorithm.

Bibliography

[AZ15] B. Aragam and Q. Zhou. Concave penalized estimation of sparse Gaussian Bayesian networks. *Journal of Machine Learning Research*, 2015.

[BA10] K. P. Burnham and D. R. Anderson. *Model Selection and Multi-Model Inference: A Practical Information-Theoretic Approach.* Springer, 2010.

[BGd08] O. Banerjee, L. El Ghaoui, and A. d'Aspremont. Model selection through sparse maximum likelihood estimation for multivariate Gaussian or binary data. *Journal of Machine Learning Research*, 9(485-516), 2008.

[BGdN06] O. Banerjee, L. El Ghaoui, A. d'Aspremont, and G. Natsoulis. Convex optimization techniques for fitting sparse Gaussian graphical models. *Proceedings of the 23rd International Conference on Machine Learning*, 2006.

[BPC$^+$11] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, pages 1–122, 2011.

[CB02] D. M. Chickering and C. Boutilier. Optimal structure identification with greedy search. *Journal of Machine Learning Research*, 3:507–554, 2002.

[Chi03] D. M. Chickering. Optimal structure identification with greedy search. *The Journal of Machine Learning Research*, 3:507–554, 2003.

[Dar09] A. Darwiche. *Modeling and Reasoning with Bayesian Networks.* Cambridge University Press, 2009.

[EB92] J. Eckstein and D.P. Bertsekas. On the Douglas-Rachford splitting method and the proximal algorithm for maximal monotone operators. *Mathematical Programming*, 55(1-3):293–318, 1992.

[Ess10] J. E. Esser. *Primal Dual Algorithms for Convex Models and Applications to Image Restoration, Registration and Nonlocal Inpainting*. PhD thesis, Department of Mathematics, University of California, Los Angeles, April 2010.

[EW08] B. Ellis and W. H. Wong. Learning causal bayesian network structures from experimental data. *Journal of the American Statistical Association*, page 103(482), 2008.

[FHHT07] J. Friedman, T. Hastie, H. Höfling, and R. Tibshirani. Pathwise coordinate optimization. *Annals of Applied Statistics*, 2007.

[FHT07] J. Friedman, T. Hastie, and R. Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 2007.

[FP04] J. Fan and H. Peng. On non-concave penalized likelihood with diverging number of parameters. *The annals of statistics*, 32, 2004.

[FZ13] F. Fu and Q. Zhou. Learning sparse causal gaussian networks with experimental intervention: Regularization and coordinate descent. *Journal of the American Statistical Association*, 108(501):288–300, 2013.

[Gab83] D. Gabay. Applications of the method of multipliers to variational inequalities. In M. Fortin and R. Glowinski, editors, *Augmented Lagrangian methods: Applications to the numerical solution of boundary-value problems*. Elsevier Science Ltd, 1983.

[GMP12] J. A. Gámez, J. L. Mateo, and J. M. Puerta. One iteration chc algorithm for learning bayesian networks: an effective and efficient algorithm for high dimensional problems. *Progress in Artificial Intelligence*, 1(4):329–346, 2012.

[Hec95] D. Heckerman. A tutorial on learning with bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, March 1995.

[HGC95] D. Heckerman, D. Geiger, and D. M. Chickering. Learning bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243, 1995.

[HYW00] B.S. He, H. Yang, and S.L. Wang. Alternating direction method with self-adaptive penalty parameters for monotone variational inequalities. *Journal of Optimization Theory and applications*, 106(2):337–356, 2000.

[KB07] M. Kalisch and P. Bühlmann. Estimating high-dimensional directed acyclic graphs with pc-algorithm. *Journal of Machine Learning Research*, 8:613–636, 2007.

[KF09] D. Koller and N. Friedman. *Probabilistic Graphical Models. Principles and Techniques.* MIT Press, 2009.

[Lau96] S. L. Lauritzen. *Graphical Models.* Oxford University Press, Oxford, 1996.

[LB94] W. Lam and F. Bacchus. Learning bayesian belief networks: An approach based on the mdl principle. *Computational Intelligence*, 10:269–293, 1994.

[Rob77] R.W. Robinson. Counting unlabeled acyclic digraphs. In *Combinatorial Mathematics V*, volume 622 of *Lecture Notes in Mathematics*, pages 28–43. Springer Berlin Heidelberg, 1977.

[SG91] P. Sprites and C. Glymour. An algorithm for fast recovery of sparse causal graphs. *Social Science Computer Review*, 9(1):62–72, 1991.

[TBA06] I. Tsamardinos, L. E. Brown, and C. F. Aliferis. The max-min hill-climbing bayesian network structure learning algorithm. *Machine Learning*, 65(1):31–78, 2006.

[Tse90] P. Tseng. Further applications of a splitting algorithm to decomposition in variational inequalities and convex programming. *Mathematical Programming*, 48(249-263), 1990.

[Tse91] P. Tseng. Applications of a splitting algorithm to decomposition in convex programming and variational inequalities. *SIAM Journal on Control and Optimization*, 29(1)(119-138), 1991.

[Tse00] P. Tseng. A modified forward-backward splitting method for maximal monotone mappings. *SIAM Journal on Control and Optimization*, 38(2):431–446, 2000.

[WL08] T. Wu and K. Lange. Coordinate descent procedures for lasso penalized regression. *The Annals of Applied Statistics*, 2, 2008.

[YL07] M. Yuan and Y. Lin. Model selection and estimation in the Gaussian graphical model. *Biometrika*, 94(1)(19-35), 2007.

[Zho11] Q. Zhou. Multi-domain sampling with applications to structural inference of bayesian networks. *Journal of the American Statistical Association*, 106(496):1317–1330, 2011.

[Zou06] H. Zou. The adaptive lasso and its oracle properties. *Journal of the American Statistical Association*, 101, 2006.