Learning Gene Networks Underlying Clinical Phenotypes Under SNP Perturbations From Genome-Wide Data

Calvin McCarter
Machine Learning Department
Carnegie Mellon University
calvinm@cmu.edu

Committee members:
Seyoung Kim (Chair)
Pradeep Ravikumar
Kathryn Roeder
Dietrich Stephan (University of Pittsburgh)

January 14, 2019
Abstract

Recent technologies are generating an abundance of genome sequence data and molecular and clinical phenotype data, providing an opportunity to understand the genetic architecture and molecular mechanisms underlying diseases. Previous approaches have largely focused on the co-localization of single-nucleotide polymorphisms (SNPs) associated with clinical and expression traits, each identified from genome-wide association studies and expression quantitative trait locus (eQTL) mapping, and thus have provided only limited capabilities for uncovering the molecular mechanisms behind the SNPs influencing clinical phenotypes. Here we aim to extract rich information on the functional role of trait-perturbing SNPs that goes far beyond this simple co-localization. We introduce a computational framework called Perturb-Net for learning the gene network that modulates the influence of SNPs on phenotypes, using SNPs as naturally occurring perturbation of a biological system. Perturb-Net uses a probabilistic graphical model to directly model both the cascade of perturbation from SNPs to the gene network to the phenotype network and the network at each layer of molecular and clinical phenotypes. Perturb-Net learns the entire model by solving a single optimization problem with an extremely fast algorithm that can analyze human genome-wide data within a few hours. In our analysis of asthma data, for a locus that was previously implicated in asthma susceptibility but for which little is known about the molecular mechanism underlying the association, Perturb-Net revealed the gene network modules that mediate the influence of the SNP on asthma phenotypes. Many genes in this network module were well supported in the literature as asthma-related.
## Contents

1 **Introduction**  
1.1 Motivation .................................................. 1  
1.2 Thesis objectives ............................................ 2  
1.3 Background on sparse modeling  
1.3.1 The Lasso and sparse Gaussian graphical models .......... 3  
1.3.2 The sparse conditional Gaussian graphical model .......... 3  
1.4 Background on modeling genomic data  
1.4.1 GWAS and eQTL mapping .................................... 4  
1.4.2 Statistical learning approaches to gene network learning . . 5  
1.4.3 Three-way integrative genomics analyses .................. 5

2 **Optimization for Sparse CGGMs**  
2.1 Background on Newton Coordinate Descent Optimization ........ 7  
2.1.1 Optimization Method ...................................... 7  
2.1.2 Computational Complexity and Scalability Problems ........ 8  
2.2 Fast-sCGGM for Improving Computation Time .................. 9  
2.2.1 Coordinate Descent Optimization for $\Lambda$ ............ 10  
2.2.2 Coordinate Descent Optimization for $\Theta$ ............ 10  
2.3 Mega-sCGGM for Removing Memory Requirement ................. 11  
2.3.1 Blockwise Optimization for $\Lambda$ ..................... 11  
2.3.2 Blockwise Optimization for $\Theta$ ..................... 13  
2.3.3 Parallelization .......................................... 14  
2.3.4 Time Complexity Analysis ................................ 14  
2.4 Experiments .................................................. 15  
2.4.1 Synthetic Data Experiments ................................ 15  
2.4.2 Genomic Data Analysis .................................. 17  
2.5 Conclusions for Optimization for Sparse CGGMs ................ 18

3 **Sparse Gaussian Chain Graph Models**  
3.1 Sparse Gaussian Chain Graph Models  
3.1.1 Sparse Linear Regression as Chain Component Model .......... 20  
3.1.2 Sparse Conditional Gaussian Graphical Model as Chain Component Model .... 20  
3.2 Markov Properties and Chain Component Models ................. 21  
3.3 Sparse Two-Layer Gaussian Chain Graph Models for Structured Sparsity .... 22  
3.4 Sparse Multi-layer Gaussian Chain Graph Models .............. 23  
3.4.1 Models .................................................. 24  
3.4.2 Semi-supervised Learning ................................ 24  
3.5 Experiments .................................................. 24  
3.5.1 Simulation Study ......................................... 24  
3.5.2 Integrative Genomic Data Analysis ......................... 27  
3.6 Conclusions for Sparse Gaussian Chain Graph Models ........... 27

4 **Asthma Perturb-Net Analysis**  
4.1 Material and Methods ......................................... 29  
4.1.1 Efficient implementation of EM algorithm for semi-supervised learning .... 29  
4.1.2 Perturb-Net inference procedures .......................... 29
4.1 Prediction tasks
4.1.3 Prediction tasks ................................................................. 32
4.1.4 Lasso for comparison with our algorithms ............................. 32
4.1.5 Preparation of asthma dataset ........................................... 33
4.1.6 Comparison of the computation time of different algorithms ...... 33

4.2 Comparison of the scalability of Mega-sCGGM and other methods .................................................. 35

4.3 Analysis of asthma data ......................................................... 36
4.3.1 Overview of the Perturb-Net model for asthma ...................... 36
4.3.2 Gene modules that influence phenotypes are enriched for immune genes ........................................... 37
4.3.3 Local SNP perturbations and direct SNP perturbations have larger eQTL effect sizes ........................ 41
4.3.4 SNPs perturbing asthma phenotypes overlap with SNPs perturbing immune modules ........................ 41
4.3.5 The immune modules mediate SNP perturbation of phenotypes ..................................................... 44
4.3.6 Module 13 explains the molecular mechanism of the previously known association 
        between SNP rs63340 and asthma susceptibility .......................... 44

4.4 Comparison with other methods .............................................. 48

4.5 Discussion of Perturb-Net for Asthma .................................... 49
1 Introduction

1.1 Motivation

One of the key questions in biology is how genetic variation perturbs gene regulatory systems to influence disease susceptibility or other phenotypes in a population. Recent advances in technologies have allowed researchers to obtain genome sequence data along with phenotype data at different levels of biological systems, such as gene expression [35], proteome [4], metabolome [79], and various clinical phenotype data. Combining genome sequence data with various types of molecular and clinical phenotype data in a computational analysis has the potential to reveal the complex molecular mechanisms controlled by different genetic loci that underlie diseases and other phenotypes.

To study gene regulatory systems, many previous works have considered the naturally-occurring perturbation of gene expression by genetic variants such as single nucleotide polymorphisms (SNPs), captured in expression and genotype data collected from a population. Compared to experimental perturbation methods such as gene knockdown [18] and genome editing techniques [76], SNP perturbation for functional genomics studies has an advantage of being more cost effective, being easily applicable to humans, and being potentially more meaningful subtle perturbations because they exist in nature [44]. However, it comes with the computational challenge of having to isolate the perturbation effect of each individual genetic variant, when a large number of genetic variants are perturbing the gene network simultaneously. Several computational methods have been proposed to address this challenge. Sparse conditional Gaussian graphical models (sCGGMs) have been introduced for simultaneously identifying the gene network and expression quantitative trait loci (eQTLs) from population SNP and expression data [81, 98, 24]. Many other works have relied on statistically less powerful approaches of identifying eQTLs first and then incorporating the eQTLs in the network learning procedure [21, 72, 73].

However, there have been relatively few works on modeling how a gene network perturbed by SNPs mediates the SNP perturbation of phenotypes. Most of the existing methods did not directly address this problem and thus, provided only limited capabilities for uncovering the molecular mechanisms behind the SNP perturbation of clinical phenotypes. Many of the previous approaches were concerned with identifying simply the co-localization of eQTLs and trait-associated SNPs [29, 38, 26], each of which were identified in a separate eQTL mapping [32, 30, 21, 35] and a genome-wide association study [56, 87]. These methods did not provide a description of the regulatory roles of the trait-associated SNPs beyond their co-localization with eQTLs. The genome-transcriptome-phenome structured association method [17] focused only on identifying eQTLs and trait-associated SNPs, and was concerned with neither learning a gene network nor uncovering its role in modulating SNP effects on phenotypes. A predictive network model for diseases that involves Bayesian networks for gene regulatory networks have been proposed [65], but this approach relied on an elaborate pipeline of analysis to identify disease-related gene modules and genetic variants that could potentially lead to loss of statistical power.

Here, our goal is to extract rich information on the functional role of trait-perturbing SNPs that goes far beyond the simple co-localization with eQTLs, which was the focus of many of the previous studies [29, 38, 26]. Towards this goal, we introduce a computational framework called Perturb-Net for directly modeling and learning the gene network that modulates the influence of SNPs on phenotypes, using SNPs as naturally occurring perturbation of a biological system. Perturb-Net builds on the key idea in the previous work on sCGGMs [81, 98] for learning a gene network using SNP perturbations, and models the cascade of a gene network and a phenotype network under SNP perturbations as a cascade of sCGGMs, called a sparse Gaussian chain graph model. Our probabilistic graphical model framework naturally leads to a set of inference algorithms for inferring a detailed description of how different parts of the gene network mediate the influence of SNPs on phenotypes, given the model estimated from population genotype, expression, and phenotype data. The Perturb-Net model and inference procedures together provide a powerful tool for
studying the gene regulatory mechanisms whose perturbations by SNPs lead to diseases.

We present a statistically powerful and extremely efficient algorithm for learning the Perturb-Net model. The Perturb-Net learning algorithm is statistically powerful, since it estimates the entire model by solving a single optimization problem with minimal loss of statistical power and with a guarantee in finding the optimal solution due to the convexity of the optimization problem. The Perturb-Net learning algorithm is also computationally efficient and can analyze human genome-wide data with 500,000 SNPs, 11,000 gene expression levels, and several dozens of phenotype data within a few hours. The performance of the Perturb-Net learning algorithm directly depends on that of sCGGM optimization, since it uses the sCGGM learning algorithm as a key module. The previous state-of-the-art method [94] had limited scalability due to expensive computation time and large memory requirement, requiring more than 4 hours for only 10,000 SNPs and running out of memory for 40,000 SNPs. We present a new learning algorithm Fast-sCGGM and its extension Mega-sCGGM with orders-of-magnitude speed-up in computation time that runs on a single machine without running out of memory and that is parallelizable. Our new sCGGM learning algorithms allow Perturb-Net to be applied to human genome-wide data.

We demonstrate Perturb-Net on the data collected for participants in the Childhood Asthma Management Program Continuation Study (CAMP-CS) [33, 34, 58]. Perturb-Net revealed the asthma gene network and how different parts of this gene network mediate the SNP perturbations of phenotypes. Furthermore, for a locus that was previously implicated in asthma susceptibility but for which little has been known about the molecular mechanism underlying the association, Perturb-Net revealed the gene network modules that mediate the influence of the SNP on asthma phenotypes. Many genes in this network module were well supported in the literature as asthma-related, suggesting our framework can reveal the molecular mechanisms underlying the SNP perturbations of phenotypes.

1.2 Thesis objectives

We focus on four subgoals to provide statistical machine learning methods for genome-wide analysis of genotypic variation influencing clinical traits via gene regulatory networks.

1. **Develop scalable convex optimization methods for learning sCGGMs.** Our new learning algorithm has orders-of-magnitude speedup over previous methods and scales to genome-wide SNP data on machine with limited memory.

2. **Develop the sparse Gaussian chain graph model in order to model the cascaded influence from SNPs to gene expression levels to clinical phenotypes.** We address how to use the learned model to extract information about the role of both individual genes and also gene modules in mediating the effect of genetic variants on phenotypes. We also use our proposed model to enable utilization of samples where genotype and phenotype data are available but gene expressions are missing.

3. **We apply our method to asthma data to show how to extract from the estimated model rich information on the molecular interactions underpinning the association between gene variants and traits.** We show how to perform colocalization analysis on our learned model to uncover the gene modules with eQTLs that colocalize with the trait-associated SNPs. We also show how, using the learned model, one may focus in on the SNPs and gene modules which influence lung function traits and blood measurement traits.

The work described in the previous objectives has been completed. We plan to accomplish the following objectives.

4. **Compare the power and specificity of our model with other statistical models.** We will use both simulated and asthma data to examine whether our method provides better recovery of colocalized
SNPs at the same false-positive error rate as other methods. To evaluate the power and specificity for recovering gene networks, we will compare our approach with sparse GGMs. Furthermore, we will examine how the power of our approach compares to univariate tests in detecting SNP-gene and SNP-trait associations.

5. Develop computational methods for correcting for confounding factors in learning gene networks perturbed by genetic variants. We will combine a linear mixed effects model, previously developed for correcting for population structures in genome-wide association studies, with an sCGGM in a single statistical model. Our approach will employ the Kronecker sum of the two covariance matrices, one for gene networks and the other for genetic relationships of individuals. We will apply our approach to asthma data.

1.3 Background on sparse modeling

In this section, we first describe approaches to learning the structure of sparse probabilistic graphical models that build on Lasso [84] and Graphical Lasso [22]. Next, we describe the sCGGM [80, 94], as well as previously developed optimization methods for this model.

1.3.1 The Lasso and sparse Gaussian graphical models

Our work on sparse probabilistic graphical model learning is closely related to problems in sparse regression. The Lasso [84] for sparse linear regression is a key building block for many recent approaches to sparse regression and sparse graphical model learning, including our work. Rather than minimizing the least-squares error subject to a constraint on the number of non-zero parameters, the Lasso constrains the $\ell_1$-norm of the parameters, leading to a convex problem with global optimum and efficient solution via a coordinate descent algorithm.

GGMs [22] have been extremely popular as a tool for learning a network structure over a large number of continuous variables in many different application domains including neuroscience [59] and biology [22]. A sparse GGM can be estimated as a sparse inverse covariance matrix by minimizing the convex function of $\ell_1$-regularized negative log-likelihood. Non-zero elements in the estimated GGM parameter matrix indicate conditional independence relationships between the two corresponding variables given all the other variables.

Highly scalable learning algorithms such as Graphical Lasso [22], QUIC [40], and BigQUIC [41] have been proposed to learn GGMs. The QUIC algorithm finds in each iteration a generalized Newton descent direction by forming a second-order approximation of the smooth part of the objective and minimizing this along with the $\ell_1$ penalty. Finding the Newton descent direction is itself solved iteratively, by solving a series of Lasso subproblems. Given this Newton direction, the parameter estimates are updated with a step size found by backtracking line search. BigQUIC extends QUIC to high-dimensional problems where the large dense matrices computed in QUIC do not fit in memory. This algorithm divides the parameters into blocks and passes through these blocks while computing the Newton direction, requiring only smaller portions of these large dense matrices to avoid exceeding computer system memory.

1.3.2 The sparse conditional Gaussian graphical model

Sparse CGGMs have been introduced as a discriminative extension of sparse GGMs to model a sparse network over outputs conditional on input variables [80, 94]. CGGMs can be viewed as a Gaussian analogue of conditional random field [52], while GGMs are a Gaussian analogue of Markov random field. Sparse CGGMs have recently been applied to various settings including energy forecasting [93], finance [97], and biology [98], where the goal is to predict structured outputs influenced by inputs. A sparse CGGM can
be estimated by minimizing a convex function of $l_1$-regularized negative log-likelihood. This optimization problem is closely related to that for sparse GGMs because CGGMs also model the network over outputs. However, the presence of the additional parameters in CGGMs for the functional mapping from inputs to outputs makes the optimization significantly more complex than in sparse GGMs.

A CGGM [80, 94] models the conditional probability density of $x \in \mathbb{R}^p$ given $y \in \mathbb{R}^q$ as follows:

$$p(y|x; \Lambda, \Theta) = \exp\{-y^T \Lambda y - 2x^T \Theta y\}/Z(x),$$

where $\Lambda$ is a $q \times q$ matrix for modeling the network over $y$ and $\Theta$ is a $p \times q$ matrix for modeling the mapping between the input variables $x$ and output variables $y$. The normalization constant is given as

$$Z(x) = (2\pi)^{q/2} |\Lambda|^{-1} \exp(x^T \Theta \Lambda^{-1} \Theta^T x).$$

Non-zeros in $\Lambda$ indicate conditional dependencies between the corresponding pairs of output variables, given all inputs and the other output variables. Similarly, non-zeros in $\Theta$ encode conditional dependencies between the corresponding output variable and input variable, given all the other input and output variables. Inference in a CGGM gives $p(y|x) = \mathcal{N}(B^T x, \Lambda^{-1})$, where $B = -\Theta \Lambda^{-1}$, showing the connection to multivariate linear regression.

A sparse estimate of CGGM parameters can be obtained by minimizing the negative log-likelihood combined with $l_1$-regularization. As observed in [80, 97, 94], this objective is convex. Several different approaches have been previously proposed to estimate sparse CGGMs, including OWL-QN [80], accelerated proximal gradient method [97], and Newton coordinate descent algorithm [94]. In particular, the Newton coordinate descent algorithm extends the QUIC algorithm [40] for sparse GGM estimation to the case of CGGMs, and has been shown to have superior computational speed and convergence. This approach finds in each iteration a descent direction by minimizing a quadratic approximation of the original negative log-likelihood function along with $l_1$ regularization. Then, the parameter estimate is updated with this descent direction and a step size found by line search.

The previous state-of-the-art Newton coordinate descent method [94] was not efficient enough to be applied to many real-world problems even with tens of thousands of variables. More importantly, it suffers from a large space requirement, because for very high-dimensional problems, several large dense matrices need to be precomputed and stored during optimization. For a CGGM with $p$ inputs and $q$ outputs, the algorithm requires storing several $p \times p$ and $q \times q$ dense matrices, which cannot fit in memory for large $p$ and $q$. In this dissertation, we introduce a fast algorithm that substantially improves the scalability both in terms of time and space.

### 1.4 Background on modeling genomic data

In this section we review existing methods for inferring the genetic basis of clinical traits and expression traits, for learning gene networks from gene expression data, and for combining such information with integrative methods. We also discuss the shortcomings and difficulties with applying such methods in piecemeal fashion, rather than addressing these problems together with a single model.

#### 1.4.1 GWAS and eQTL mapping

A genome-wide association study (GWAS) aims at finding the genome-wide set of genetic variants, typically SNPs, that are statistically associated with a trait in order to establish the genetic causes of phenotypic variation. The dominant paradigm in GWAS analysis is to perform separate univariate statistical tests for all SNP-trait pairs. While GWAS data have greatly facilitated discovery of the genetic basis of complex-trait diseases, such traits are very polygenic, with each causal SNP explaining on average a small proportion of population variation, so that GWAS analyses still often lack the statistical power to detect all the causal variants [86], especially in cases of linkage disequilibrium. To tackle this, combining all SNPs together
into a single sparse regression model for each phenotype has been proposed [92], using either Lasso or $\ell_1$-penalized logistic regression.

Expression quantitative trait loci (eQTL) analysis investigates the genetic basis of gene expression by treating gene transcript abundances as phenotypes, using computational methods similar to those for GWAS. The goal of such eQTL analyses is to learn the molecular basis of associations between genetic variation and traits. Because gene expression levels are influenced not only by genotype, but also by tissue and disease state, eQTL mapping studies must be specific to a particular disease state or tissue type, as in the GTEx Project [35]. As in GWAS, most eQTL studies rely on simple univariate testing for each SNP-transcript pair, in part due to the computational demands of analyzing tens of thousands of gene transcripts. Various computational methods have been developed to improve the scalability of univariate testing in eQTL mapping [75, 27].

1.4.2 Statistical learning approaches to gene network learning

The graphical Lasso [22] for learning sparse GGMs was motivated by the problem of gene network estimation from gene expression level data gathered through microarray experiments. Since then, it has gained wide use as a gene regulatory network learning approach [55, 49]. Various extensions of the sparse GGM have also been proposed to utilize prior knowledge about regulatory pathways [31, 78], to jointly learn multiple related networks for different tissues or disease types [19, 57], and to learn time-varying gene networks [90, 37]. However, these approaches do not jointly learn the eQTLs, and are therefore unable to factor out the portion of the observed correlations among gene expressions that arise from shared causal SNPs rather than gene-gene regulatory effects [64].

Sparse multivariate linear regression with sparse inverse covariance estimation (MRCE) [71] has been applied to learn gene networks after adjusting for the effect of genotype covariates [96, 12]. However, because this approach led to a non-convex optimization problem that scales poorly with the number of SNPs, these studies limited their analysis to only a few hundreds of SNPs.

The sparse CGGM has also been proposed as an approach to learning gene networks under SNP perturbations [98]. This method, unlike MRCE, distinguished between marginal SNP-gene relationships, captured by sparse regression coefficients, and conditional SNP-gene dependencies after conditioning on the learned gene network, in order to identify the genes directly perturbed by a SNP versus those indirectly affected through the gene regulatory network. The gene network learned from the sparse CGGM model has been used to infer the correlation matrix among genes after accounting for the gene correlations induced by SNPs [3]. In that study, the sparse CGGM was shown to scale to a few thousand SNPs and a few thousand gene expression levels, but was not yet capable of performing genome-wide analysis of human data. Furthermore, this approach did not allow incorporation of phenotypic data into the learning algorithm.

1.4.3 Three-way integrative genomics analyses

Colocalization methods are a leading general approach to integrating genotype, gene expression, and phenotype data in order to uncover genes which lead to disease traits. In colocalization analyses, GWAS and eQTL mapping are first performed separately to find SNP-gene and SNP-trait associations. Next, these approaches link genes and traits with common patterns of underlying SNP associations. Sherlock [38], COLOC [29], and eCAVIAR [39] are Bayesian methods that compare p-values from GWAS and eQTL mapping hypothesis tests to find SNPs that show up as associated with both gene expression and clinical traits. However, these analyses are limited by their reliance on univariate tests. Wen et al. [88] generalize COLOC and eCAVIAR with a model that incorporates the effect of multiple SNPs on each trait using a sparse regression model, but like the previous method it does not account for correlations among genes through the gene regulatory network or correlations between gene expression levels and clinical traits.
Another set of integrative techniques is built on imputation of gene expression levels using eQTL mapping study results, such as from the GTEx Project. These two-step procedures [26, 63] predict gene expression levels for individuals from their genotypes, then regress phenotypes on these imputed gene expression values. While these methods reduce the multiple-testing burden from examining genome-wide SNPs, they do not model the gene and trait networks, or the colocalization of SNPs regulating both genes and traits.

Another branch of three-way integrative analyses methods incorporate all variables into a Bayesian network [85, 16]. While Bayesian networks are able to model all the interactions among the genotype, gene expression, and phenotype variables, learning Bayesian networks poses serious statistical and computational difficulties. To address this, the authors of [85, 16] employed a complex analysis pipeline with rules to favor or exclude possible edges in the network model. The complexity of this pipeline makes it challenging to apply to data collected from other diseases. To overcome the complexities of the previous approaches, we will develop a method that models all interactions among genomic data types, automatically learns these interactions from data, and uses these learned interactions to discover the molecular basis of traits, including the role of the gene network and the colocalization between trait SNPs and eQTLs.
2 Optimization for Sparse CGGMs

In this chapter, we address the problem of scaling up the optimization of the sCGGM to very large problem sizes without requiring excessive time or memory. We propose sparse CGGM learning algorithms called Fast-sCGGM for reducing computation time and Mega-sCGGM for further improving Fast-sCGGM to remove the memory constraint. Fast-sCGGM improves the computation time of the previous Newton coordinate descent method [94] by alternately optimizing the network parameters and the input-output perturbation parameters. While Fast-sCGGM improves the computation time of the previous method, it is limited by the memory size required to store large $q \times q$ or $p \times q$ matrices during the iterative optimization. Hence, we combine Fast-sCGGM with block coordinate descent and introduce the Mega-sCGGM algorithm to scale up the optimization to very large problems on a machine with limited memory. During the iterative optimization, we update blocks of the large matrices so that within each block, the computation of the large matrices can be cached and re-used.

Our Fast-sCGGM algorithm is based on the key observation that the computation simplifies drastically if we alternately optimize the two sets of parameters for output network and for mapping inputs to outputs, instead of updating all parameters at once as in the previous approach. The previous approach updated all parameters simultaneously by forming a second-order approximation of the objective on all parameters, which requires an expensive computation of the large Hessian matrix of size $(p + q) \times (p + q)$ in each iteration. Our approach of alternate optimization forms a second-order approximation only on the network parameters, which requires the Hessian of size $q \times q$, as the other set of parameters can be updated easily using a simple coordinate descent.

In order to overcome the constraint on the space requirement, we then introduce Mega-sCGGM, an alternating Newton block coordinate descent method that can be applied to problems of unbounded size on a machine with limited memory. A naive approach to reduce the memory footprint would be to recompute portions of these matrices on demand for each coordinate update, which would be very expensive. Hence, we divide the parameters into blocks for block-wise updates such that the results of computation can be reused within each block. Block-wise parameter updates were previously used in BigQUIC [41] for learning a sparse GGM, where the block sparsity pattern of the network parameters was leveraged to overcome the space limitations. We propose an approach for block-wise update of the output network parameters in CGGMs that extends their idea. We then propose a new block-wise update strategy for the parameters for mapping inputs to outputs. These blocks are determined automatically in each iteration by exploiting the sparse structure within the network parameters and the input-output parameters. In our experiments, we show that we can solve problems with a million inputs and hundreds of thousands of outputs on a single machine.

2.1 Background on Newton Coordinate Descent Optimization

In this section we first describe the Newton coordinate descent algorithm [94], and then describe the difficulty with scaling this method to large problems due to its time and space complexity.

2.1.1 Optimization Method

Given a mean-centered dataset of $X \in \mathbb{R}^{n \times p}$ and $Y \in \mathbb{R}^{n \times q}$ for $n$ samples, and their covariance matrices $S_{xx} = \frac{1}{n}X^TX$, $S_{xy} = \frac{1}{n}X^TY$, $S_{yy} = \frac{1}{n}Y^TY$, a sparse estimate of CGGM parameters can be obtained by minimizing $l_1$-regularized negative log-likelihood. This can be written as the

$$\min_{\Lambda > 0, \Theta} f(\Lambda, \Theta) = g(\Lambda, \Theta) + h(\Lambda, \Theta),$$

(1)
where \( g(\Lambda, \Theta) = -\log |\Lambda| + \text{tr}(S_{yy} \Lambda + 2S_{xy}^T \Theta + \Lambda^{-1} \Theta^T S_{xx} \Theta) \) for the smooth negative log-likelihood and 
\( h(\Lambda, \Theta) = \lambda_\Lambda \| \Lambda \|_1 + \lambda_\Theta \| \Theta \|_1 \) for the non-smooth elementwise \( l_1 \) penalty. \( \lambda_\Lambda, \lambda_\Theta > 0 \) are regularization parameters. As observed in [80, 97, 94], this objective is convex.

The current state-of-the-art method for solving Eq. (1) for \( l_1 \)-regularized CGGM is the Newton coordinate descent algorithm [94] that extends QUIC [40] for \( l_1 \)-regularized GGM estimation. In each iteration, this algorithm found a generalized Newton descent direction by forming a second-order approximation of the smooth part of the objective and minimizing this along with the \( l_1 \) penalty. Given this Newton direction, the parameter estimates were updated with a step size found by line search using Armijo’s rule [8].

In each iteration, the Newton coordinate descent algorithm found the Newton direction as follows:

\[
D_{\Lambda}, D_{\Theta} = \arg\min_{\Delta_{\Lambda}, \Delta_{\Theta}} \bar{g}_{\Lambda, \Theta}(\Delta_{\Lambda}, \Delta_{\Theta}) + h(\Lambda + \Delta_{\Lambda}, \Theta + \Delta_{\Theta}),
\]

where \( \bar{g}_{\Lambda, \Theta} \) is the second-order approximation of \( g \) given by Taylor expansion:

\[
\bar{g}_{\Lambda, \Theta}(\Delta_{\Lambda}, \Delta_{\Theta}) = \text{vec}(\nabla g(\Lambda, \Theta)) \nabla_{\Theta} g(\Lambda, \Theta) + \frac{1}{2} \text{vec}(\Delta_{\Theta}) \nabla^2 g(\Lambda, \Theta) \text{vec}(\Delta_{\Theta}).
\]

The gradient and Hessian matrices above are given as:

\[
\nabla g(\Lambda, \Theta) = \begin{bmatrix} \nabla_{\Lambda} g(\Lambda, \Theta) & \nabla_{\Theta} g(\Lambda, \Theta) \end{bmatrix} \\
= [S_{yy} - \Sigma - \Psi \ 2S_{xy} + 2\Gamma]
\]

\[
\nabla^2 g(\Lambda, \Theta) = \begin{bmatrix} \nabla^2_{\Lambda} g(\Lambda, \Theta) & \nabla_{\Lambda} \nabla_{\Theta} g(\Lambda, \Theta) \\
\nabla_{\Lambda} \nabla_{\Theta} g(\Lambda, \Theta)^T & \nabla^2_{\Theta} g(\Lambda, \Theta) \end{bmatrix} \\
= \begin{bmatrix} \Sigma \otimes (\Sigma + 2\Psi) & -2\Sigma \otimes \Gamma^T \\
-2\Sigma \otimes \Gamma & 2\Sigma \otimes S_{xx} \end{bmatrix},
\]

where \( \Sigma = \Lambda^{-1}, \Psi = \Sigma \Theta^T S_{xx} \Theta \Sigma, \) and \( \Gamma = S_{xx} \Theta \Sigma. \) Given the Newton direction in Eq. (2), the parameters can be updated as \( \Lambda ← \Lambda + \alpha D_{\Lambda} \) and \( \Theta ← \Theta + \alpha D_{\Theta}, \) where step size \( 0 < \alpha \leq 1 \) ensures sufficient decrease in Eq. (1) and positive definiteness of \( \Lambda. \)

The Lasso problem [83] in Eq. (2) was solved using coordinate descent. Despite the efficiency of coordinate descent for Lasso, applying coordinate updates repeatedly to all \( q^2 + pq \) variables in \( \Lambda \) and \( \Theta \) is costly. So, the updates were restricted to an active set of variables given as:

\[
S_{\Lambda} = \{(\Delta_{\Lambda})_{ij} : |(\nabla_{\Lambda} g(\Lambda, \Theta))_{ij}| > \lambda_\Lambda \lor \Lambda_{ij} \neq 0\} \\
S_{\Theta} = \{(\Delta_{\Theta})_{ij} : |(\nabla_{\Theta} g(\Lambda, \Theta))_{ij}| > \lambda_\Theta \lor \Theta_{ij} \neq 0\}.
\]

Because the active set sizes \( m_{\Lambda} = |S_{\Lambda}|, m_{\Theta} = |S_{\Theta}| \) approach the number of non-zero entries in the sparse solutions for \( \Lambda^* \) and \( \Theta^* \) over iterations, this strategy yields a substantial speedup.

To further improve the efficiency of coordinate descent, intermediate results were stored for the large matrix products that need to be computed repeatedly. At the beginning of the optimization for Eq. (2), \( U := \Delta_{\Lambda} \Sigma \) and \( V := \Delta_{\Theta} \Sigma \) were computed and stored. Then, after a coordinate descent update to \( (\Delta_{\Lambda})_{ij}, \) row \( i \) and \( j \) of \( U \) were updated. Similarly, after an update to \( (\Delta_{\Theta})_{ij}, \) row \( i \) of \( V \) was updated.

### 2.1.2 Computational Complexity and Scalability Problems

Although the Newton coordinate descent method is computationally more efficient than other previous approaches, it does not scale to problems even with tens of thousands of variables. The main computational
\textbf{input} : Inputs $X \in \mathbb{R}^{n \times p}$ and $Y \in \mathbb{R}^{n \times q}$; regularization parameters $\lambda_\Lambda, \lambda_\Theta$

\textbf{output}: Parameters $\Lambda, \Theta$

Initialize $\Theta \leftarrow 0, \Lambda \leftarrow I_q$

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

Determine active sets $S_\Lambda, S_\Theta$

Solve via coordinate descent:

\[
D_\Lambda = \arg\min_{\Delta \Lambda} \bar{g}_\Lambda,\Theta(\Lambda + \Delta \Lambda, \Theta) + h(\Lambda + \Delta \Lambda, \Theta)
\]

Update $\Lambda^+ = \Lambda + \alpha D_\Lambda$, where step size $\alpha$ is found with line search

Solve via coordinate descent:

\[
\Theta^+ = \arg\min_\Theta g_\Lambda(\Theta) + \lambda_\Theta \|\Theta\|_1
\]

end

\textbf{Algorithm 1: Fast-sCGGM}

The cost of the algorithm comes from computing the large $(p + q) \times (p + q)$ Hessian matrix in Eq. (4) in each application of Eq. (2) to find the Newton direction. At the beginning of the optimization in Eq. (2), large dense matrices $\Sigma, \Psi$, and $\Gamma$, for computing the gradient and Hessian in Eqs. (3) and (4), are precomputed and reused throughout the coordinate descent iterations. Initializing $\Sigma = \Lambda^{-1}$ via Cholesky decomposition costs up to $O(q^3)$ time, although in practice, sparse Cholesky decomposition exploits sparsity to invert $\Lambda$ in much less than $O(q^3)$. Computing $\Psi = \frac{1}{n} R^T R$, where $R = X \Theta \Sigma$, requires $O(nmp + nq^2)$ time, and computing $\Gamma$ costs $O(npq + nq^2)$. After the initializations, the cost of coordinate descent update per each active variable $(\Delta_\Lambda)_{ij}$ and $(\Delta_\Theta)_{ij}$ is $O(p + q)$. During the coordinate descent for solving Eq. (2), the entire $(p + q) \times (p + q)$ Hessian matrix in Eq. (4) needs to be evaluated, whereas for the gradient in Eq. (3) only those entries corresponding to the parameters in active sets are evaluated.

A more serious obstacle to scaling up to problems with large $p$ and $q$ is the space required to store dense matrices $\Sigma$ (size $q \times q$), $\Psi$ (size $q \times q$), and $\Gamma$ (size $p \times q$). In our experiments on a machine with 104 Gb RAM, the Newton coordinate descent method exhausted memory when $p + q$ exceeded 80,000.

In the next section, we propose a modification of the Newton coordinate descent algorithm that significantly improves the computation time. Then, we introduce block-wise update strategies to our algorithm to remove the memory constraint.

### 2.2 Fast-sCGGM for Improving Computation Time

In this section, we introduce our Fast-sCGGM algorithm for learning an $l_1$-regularized CGGM that significantly reduces computation time compared to the previous method. Instead of performing Newton descent for all parameters $\Lambda$ and $\Theta$ simultaneously, our approach alternately updates $\Lambda$ and $\Theta$, optimizing Eq. (1) over $\Lambda$ given $\Theta$ and vice versa until convergence.

Our approach is based on the key observation that with $\Lambda$ fixed, the problem of solving Eq. (1) over $\Theta$ becomes simply minimizing a quadratic function with $l_1$ regularization. Thus, it can be solved efficiently using a coordinate descent method, without the need to form a second-order approximation or to perform line search. On the other hand, optimizing Eq. (1) for $\Lambda$ given $\Theta$ still requires forming a quadratic approximation to find a generalized Newton direction and performing line search to find the step size. However, this computation involves only $q \times q$ Hessian matrix and is significantly simpler than performing the same type of computation on both $\Lambda$ and $\Theta$ jointly as in the previous approach.
2.2.1 Coordinate Descent Optimization for $\Lambda$

Given fixed $\Theta$, the problem of minimizing the objective in Eq. (1) with respect to $\Lambda$ becomes

$$\arg\min_{\Lambda \succ 0} g_\Theta(\Lambda) + \lambda_\Lambda \|\Lambda\|_1,$$

where $g_\Theta(\Lambda) = -\log |\Lambda| + \text{tr}(S_{\text{yy}}\Lambda + \Lambda^{-1}\Theta^T S_{\text{xx}}\Theta)$. In order to solve this, we first find a generalized Newton direction that minimizes the $l_1$-regularized quadratic approximation of $g_\Theta(\Lambda)$:

$$D_\Lambda = \arg\min_{\Delta_\Lambda} \tilde{g}_{\Lambda, \Theta}(\Delta_\Lambda) + \lambda_\Lambda \|\Lambda + \Delta_\Lambda\|_1,$$

where $\tilde{g}_{\Lambda, \Theta}(\Delta_\Lambda)$ is obtained from a second-order Taylor expansion and is given as

$$\tilde{g}_{\Lambda, \Theta}(\Delta_\Lambda) = \text{vec}(\nabla_\Lambda g(\Lambda, \Theta))^T \text{vec}(\Delta_\Lambda) + \frac{1}{2} \text{vec}(\Delta_\Lambda)^T \nabla^2_\Lambda g(\Lambda, \Theta) \text{vec}(\Delta_\Lambda).$$

The $\nabla_\Lambda g(\Lambda, \Theta)$ and $\nabla^2_\Lambda g(\Lambda, \Theta)$ above are components of the gradient and Hessian matrices corresponding to $\Lambda$ in Eqs. (3) and (4). We solve the Lasso problem in Eq. (5) via coordinate descent. Similar to the Newton coordinate descent method, we maintain $U := \Delta_\Lambda \Sigma$ to reuse intermediate results of the large matrix-matrix product. Given the Newton direction for $\Lambda$, we update $\Lambda \leftarrow \Lambda + \alpha \Delta_\Lambda$, where $\alpha$ is obtained by line search.

During coordinate descent to solve the Lasso problem, each element of $\Delta_\Lambda$ is updated as follows:

$$(\Delta_\Lambda)_{ij} \leftarrow (\Delta_\Lambda)_{ij} - c_\Lambda + S_{\lambda_\Lambda/a_\Lambda}(c_\Lambda - \frac{b_\Lambda}{a_\Lambda}),$$

where $S_r(w) = \text{sign}(w) \max(|w| - r, 0)$ is the soft-thresholding operator and

$$a_\Lambda = \Sigma_{ii}^2 + \Sigma_{ii} \Sigma_{jj} + \Sigma_{ii} \Psi_{jj} + 2 \Sigma_{ij} \Psi_{ij} + \Sigma_{jj} \Psi_{ii},$$

$$b_\Lambda = (S_{\text{yy}})_{ij} - \Sigma_{ij} - \Psi_{ij} + (\Sigma \Delta_\Lambda \Sigma)_{ij} + (\Psi \Delta_\Lambda \Sigma)_{ij} + (\Psi \Delta_\Lambda \Sigma)_{ji},$$

$$c_\Lambda = \Lambda_{ij} + (\Delta_\Lambda)_{ij}.$$

Restricting the generalized Newton descent to $\Lambda$ simplifies the computation significantly for coordinate descent updates, compared to the previous approach [94] that applies it to both $\Lambda$ and $\Theta$ jointly. Our updates only involve $\nabla_\Lambda g(\Lambda, \Theta)$ and $\nabla^2_\Lambda g(\Lambda, \Theta)$, and no longer involve $\nabla_\Theta g(\Lambda, \Theta)$ and $\nabla_\Lambda \nabla_\Theta g(\Lambda, \Theta)$, eliminating the need to compute the large $p \times q$ dense matrix $\Gamma$ in $O(npq + nq^2)$ time. Our approach also reduces the computational cost for the coordinate descent update of each element of $\Delta_\Lambda$ from $O(p + q)$ to $O(q)$.

2.2.2 Coordinate Descent Optimization for $\Theta$

With $\Lambda$ fixed, the optimization problem in Eq. (1) with respect to $\Theta$ becomes

$$\arg\min_{\Theta} g_\Lambda(\Theta) + \lambda_\Theta \|\Theta\|_1,$$

where $g_\Lambda(\Theta) = \text{tr}(2S_{\text{xy}}^T \Theta + \Lambda^{-1}\Theta^T S_{\text{xx}}\Theta)$. Since $g_\Lambda(\Theta)$ is a quadratic function itself, there is no need to form its second-order Taylor expansion or to determine a step size via line search. Instead, we solve Eq. (6) directly with coordinate descent method, storing and maintaining $V := \Theta \Sigma$. Our approach reduces the computation time for updating $\Theta$ compared to the corresponding computation in the previous
algorithm [94]. We avoid computing the large \( p \times q \) matrix \( \Gamma \), which had dominated overall computation time with \( O(npq) \). Our approach also eliminates the need for line search for updating \( \Theta \). Finally, it reduces the cost for each coordinate descent update in \( \Theta \) to \( O(p) \), compared to \( O(p + q) \) for the corresponding computation for \( \Delta_\Theta \) in the previous method.

The coordinate descent updates applied directly to \( \Theta \) take the following form:

\[
\Theta_{ij} \leftarrow \Theta_{ij} - c_\Theta + S_{\lambda \Theta / a_\Theta} (c_\Theta - b_\Theta / a_\Theta),
\]

where

\[
a_\Theta = 2 \sum_{ij} (S_{xx})_{ii}
\]

\[
b_\Theta = 2 (S_{xy})_{ij} + 2 (S_{xx} \Theta \Sigma)_{ij}
\]

\[
c_\Theta = \Theta_{ij}.
\]

Our approach is summarized in Algorithm 1. In practice, we approximately solve Eqs. (5) and (6) by using a warm-start for \( \Lambda \) and \( \Theta \) with the results of the previous iteration and making a single pass over the active set. This ensures decrease in the objective in Eq. (1) and reduces the overall computation time in practice.

2.3 Mega-sCGGM for Removing Memory Requirement

The Fast-sCGGM algorithm in the previous section improves the computation time of the previous state-of-the-art method, but is still limited by the space required to store large matrices during coordinate descent computation. Solving Eq. (5) for updating \( \Lambda \) requires precomputing and storing \( q \times q \) matrices, \( \Sigma \) and \( \Psi \), whereas solving Eq. (6) for updating \( \Theta \) requires \( \Sigma \) and a \( p \times p \) matrix \( S_{xx} \). A naive approach to reduce the memory footprint would be to recompute portions of these matrices on demand for each coordinate update, which would be very expensive.

Here, we describe Mega-sCGGM that combines the alternating Newton coordinate descent algorithm in Fast-sCGGM with block coordinate descent to scale up the optimization to very large problems on a machine with limited memory. During coordinate descent optimization, we update blocks of \( \Lambda \) and \( \Theta \) so that within each block, the computation of the large matrices can be cached and re-used, where these blocks are determined automatically by exploiting the sparse structure. For \( \Lambda \), we extend the block coordinate descent approach in BigQUIC [41] developed for GGMs to take into account the conditioning variables in CGGMs. For \( \Theta \), we describe a new approach for block coordinate descent update. Our algorithm can, in principle, be applied to problems of any size on a machine with limited memory.

2.3.1 Blockwise Optimization for \( \Lambda \)

**Block Coordinate Descent Method** A coordinate-descent update of \( (\Delta_\Lambda)_{ij} \) requires the \( i \)th and \( j \)th columns of \( \Sigma \) and \( \Psi \). If these columns are in memory, they can be reused. Otherwise, it is a cache miss and we should compute them on demand. \( \Sigma_i \) for the \( i \)th column of \( \Sigma \) can be obtained by solving linear system \( \Lambda \Sigma_i = e_i \) with conjugate gradient method in \( O(m_\Lambda K) \) time, where \( K \) is the number of conjugate gradient iterations. Then, \( \Psi_i \) can be obtained from \( \frac{1}{n} R_i^T R_i \) in \( O(nq) \) time, where \( R = X \Theta \Sigma \).

In order to reduce cache misses, we perform block coordinate descent, where within each block, the columns of \( \Sigma \) are cached and re-used. Suppose we partition \( \mathcal{N} = \{1, \ldots, q\} \) into \( k_\Lambda \) blocks, \( C_1, \ldots, C_{k_\Lambda} \). We apply this partitioning to the rows and columns of \( \Delta_\Lambda \) to obtain \( k_\Lambda \times k_\Lambda \) blocks. We perform coordinate-descent updates in each block, updating all elements in the active set within that block. Let \( A_{C_r} \) denote a \( q \) by \( |C_r| \) matrix containing columns of \( A \) that corresponds to the subset \( C_r \). In order to perform coordinate-descent updates on \( (C_2, C_r) \) block of \( \Delta_\Lambda \), we need \( \Sigma_{C_2}, \Sigma_{C_r}, \Psi_{C_2}, \) and \( \Psi_{C_r} \). Thus, we pick the smallest
**input**: Inputs $X \in \mathbb{R}^{n \times p}$ and outputs $Y \in \mathbb{R}^{n \times q}$; regularization parameters $\lambda_{\Lambda}, \lambda_{\Theta}$

**output**: Parameters $\Lambda, \Theta$

Initialize $\Theta \leftarrow 0$, $\Lambda \leftarrow I_q$

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

Determine active sets $S_{\Lambda}, S_{\Theta}$

Partition columns of $\Lambda$ into $k_{\Lambda}$ blocks

Initialize $\Delta_{\Lambda} \leftarrow 0$

for $z = 1$ to $k_{\Lambda}$ do

Compute $\Sigma_{C_z}, U_{C_z},$ and $\Psi_{C_z}$

for $r = 1$ to $k_{\Lambda}$ do

if $z \neq r$ then

Identify columns $B_{zr} \subset C_r$ with active elements in $\Lambda$

Compute $\Sigma_{B_{zr}}, U_{B_{zr}},$ and $\Psi_{B_{zr}}$

end

Update all active $(\Delta_{\Lambda})_{ij}$ in $(C_z, C_r)$

end

end

Update $\Lambda^+ \leftarrow \Lambda + \alpha \Delta_{\Lambda}$, where step size $\alpha$ is found with line search

Partition columns of $\Theta$ into $k_{\Theta}$ blocks

for $r = 1$ to $k_{\Theta}$ do

Compute $\Sigma_{C_r}$, and initialize $V \leftarrow \Theta \Sigma_{C_r}$

for row $i \in \{1, \ldots, p\}$ if $I_\phi(S(i, C_r))$ do

Compute $(S_{xx})_{ij}$ for non-empty rows $j$ in $V_{C_r}$

Update all active $\Theta_{ij}$ in $(i, C_r)$

end

end

end

**Algorithm 2: Mega-sCGGM**

Possible $k_{\Lambda}$ such that we can store $2q/k_{\Lambda}$ columns of $\Sigma$ and $2q/k_{\Lambda}$ columns of $\Psi$ in memory. When updating the variables within block $(C_z, C_r)$ of $\Delta_{\Lambda}$, there are no cache misses once $\Sigma_{C_z}, \Sigma_{C_r}, \Psi_{C_z},$ and $\Psi_{C_r}$ are computed and stored. After updating each $(\Delta_{\Lambda})_{ij}$ to $(\Delta_{\Lambda})_{ij} + \mu$, we maintain $U_{C_z}$ and $U_{C_r}$ by $U_{it} \leftarrow U_{it} + \mu \Sigma_{jt}$, $U_{jt} \leftarrow U_{jt} + \mu \Sigma_{it}, \forall t \in \{C_z \cup C_r\}$.

To go through all blocks, we update blocks $(C_z, C_1), \ldots, (C_z, C_{k_A})$ for each $z \in \{1, \ldots, k_{\Lambda}\}$. Since all of these blocks share $\Sigma_{C_z}$ and $\Psi_{C_z}$, we precompute and store them in memory. When updating an off-diagonal block $(C_z, C_r), z \neq r$, we compute $\Sigma_{C_r}$ and $\Psi_{C_r}$. In the worst case, overall $\Sigma$ and $\Psi$ will be computed $k_{\Lambda}$ times.

**Reducing Computational Cost Using Graph Clustering** In typical real-world problems, the graph structure of $\Lambda$ will exhibit clustering, with an approximately block diagonal structure. We exploit this structure by choosing a partition $\{C_1, \ldots, C_{k_{\Lambda}}\}$ that reduces cache misses. Within diagonal blocks $(C_z, C_z)$’s, once $\Sigma_{C_z}$ and $\Psi_{C_z}$ are computed, there are no cache misses. For off-diagonal blocks $(C_z, C_r)$’s, $r \neq z$, we have
2.3.2 Blockwise Optimization for $\Theta$

**Block Coordinate Descent Method**  The coordinate descent update of $\Theta_{ij}$ requires $(S_{xx})_i$ and $\Sigma_j$ to compute $(S_{xx})_i^T V_j$, where $V_j = \Theta \Sigma_j$. If $(S_{xx})_i$ and $\Sigma_j$ are not already in the memory, it is a cache miss. Computing $(S_{xx})_i$ takes $O(np)$, which is expensive if we have many cache misses.

We propose a block coordinate descent approach for solving Eq. (6) that groups these computations to reduce cache misses. Given a partition of $\{1, \ldots, q\}$ into $k_\Theta$ subsets, $C_1, \ldots, C_{k_\Theta}$, we divide $\Theta$ into $p \times k_\Theta$ blocks, where each block comprises a portion of a row of $\Theta$. We denote each block $(i, C_r)$, where $i \in \{1, \ldots, p\}$. Since updating block $(i, C_r)$ requires $(S_{xx})_i$ and $\Sigma_{C_r}$, we pick the smallest possible $k_\Theta$ such that we can store $q/k_\Theta$ columns of $\Sigma$ in memory. While performing coordinate descent updates within block $(i, C_r)$ of $\Theta$, there are no cache misses, once $(S_{xx})_i$ and $\Sigma_{C_r}$ are in memory. After updating each
\( \Theta_{ij} \) to \( \Theta_{ij} + \mu \), we update \( V_{C_r} \) by \( V_{it} \leftarrow V_{it} + \mu \Sigma_{jt}, \forall t \in C_r \).

In order to sweep through all blocks, each time we select a \( q \in \{1, \ldots, k_\Theta\} \) and update blocks \((1, C_r), \ldots, (p, C_r)\). Since all of these \( p \) blocks with the same \( C_r \) share the computation of \( \Sigma_{C_r} \), we compute and store \( \Sigma_{C_r} \) in memory. Within each block, the computation of \((S_{xx})_i\) is shared, so we pre-compute and store it in memory, before updating this block. The full matrix of \( \Sigma \) will be computed once while sweeping through the full \( \Theta \), whereas in the worst case \( S_{xx} \) will be computed \( k_\Theta \) times.

**Reducing Computational Cost Using Row-wise Sparsity** We further reduce cache misses for \((S_{xx})_i\) by strategically selecting partition \( C_1, \ldots, C_{k_\Theta} \), based on the observation that if the active set is empty in block \((i, C_r)\), we can skip this block and forgo computing \((S_{xx})_i\). We therefore choose a partition where the active set variables are clustered into as few blocks as possible. Formally, we want to minimize \( \sum_{i,C} \) where \( C \) strategically selecting partition \( V \) corresponds to an empty row in \( \emptyset \). We therefore perform graph clustering over the graph \( G = (V, E) \) defined from the active set in \( \Theta \), where \( V = \{1, \ldots, q\} \) with one node for each column of \( \Theta \), and \( E = \{(j,k) \mid i : \Theta_{ij} \in S_\Theta, \Theta_{ik} \in S_\Theta\} \), connecting two nodes \( j \) and \( k \) with an edge if both \( \Theta_{ij} \) and \( \Theta_{ik} \) are in the active set. This edge set corresponds to the non-zero elements of \( \Theta^T \Theta \), so the graph can be computed quickly in \( O(m_{\Theta q}) \).

We also exploit row-wise sparsity in \( \Theta \) to reduce the cost of each cache miss. Every empty row in \( \Theta \) corresponds to an empty row in \( V = \Theta \Sigma \). Because we only need elements in \((S_{xx})_i \) for the dot product \((S_{xx})_i^T V_j \), we skip computing the \( k \)th element of \((S_{xx})_i \) if the \( k \)th row of \( \Theta \) is all zeros. Our strategy for updating \( \Theta \) is illustrated in Figure 2.

Our method is summarized in Algorithm 2. See Appendix for analysis of the computational cost.

### 2.3.3 Parallelization

The most expensive computations in our algorithm are embarrassingly parallelizable, allowing for further speedups on machines with multiple cores. Throughout the algorithm, we parallelize matrix and vector multiplications. In addition, for block-wise \( \Lambda \) updates, we compute multiple columns of \( \Sigma_{C_r} \) and \( \Psi_{C_r} \) as well as multiple columns of \( \Sigma_{C_s} \) and \( \Psi_{C_s} \) for multiple cache misses in parallel, running multiple conjugate gradient methods in parallel. For block-wise \( \Theta \) updates, we compute multiple columns of \( \Sigma \) in parallel before sweeping through blocks and perform a parallel computation within each cache miss, computing elements within each \((S_{xx})_i \) in parallel.

### 2.3.4 Time Complexity Analysis

In this section we describe the time complexity of the alternating Newton block coordinate descent method. The active set sizes for \( \Lambda \) and \( \Theta \) are \( m_\Lambda \) and \( m_\Theta \), respectively. Also, \( K \) is the number of conjugate gradient iterations.

The time complexity of each \( \Lambda \) update is dominated by the cost of precomputing columns of \( \Sigma \) and \( \Psi \). The cost of these precomputations is \( O \left( \left[ 1 + \frac{B_\Lambda}{q} \right] [m_\Lambda K q + n q^2] \right) \) where \( B_\Lambda = \sum_{z \neq r} \left| \{ j \mid i \in C_z, j \in C_r, (i,j) \in S_\Lambda \} \right| \) is the number of cache misses. Although the worst-case of \( B_\Lambda = k_\Lambda q \) requires computing \( \Sigma \) and \( \Psi \) a total of \( k_\Lambda \) times, in practice, graph clustering dramatically reduces this additional cost of block-wise optimization. In the best case, when graph clustering identifies perfect block-diagonal structure in the active set, the number of cache misses \( B_\Lambda = 0 \) and we incur no runtime penalty from limited memory.

For \( \Theta \), the overall runtime is dominated by the cost of precomputing columns of \( S_{xx} \) and \( \Sigma \). The complexity of these operations is \( O(m_\Lambda K q + m_{\Theta q} + n \tilde{p} B_{\Theta}) \), where \( \tilde{p} \) is the number of non-empty rows in \( \Theta \) and \( B_{\Theta} = \sum_{i,r} \left| \{ i \mid j \in C_r, (i,j) \in S_\Theta \} \right| \) is the number of cache misses. Without any row-wise sparsity we have \( \tilde{p} = p \) and \( B_{\Theta} = k_{\Theta} p \), so the worst-case is that \( S_{xx} = \tilde{p} \) is computed a total of \( k_{\Theta} \) times. This additional cost is substantially reduced in real datasets, where most input variables influence few or none of the outputs.
2.4 Experiments

We compare the performance of our methods with the existing state-of-the-art Newton coordinate descent algorithm, using synthetic and real-world genomic datasets. All methods were implemented in C++ with parameters represented in sparse matrix format. All experiments were run on 2.6GHz Intel Xeon E5 machines with 8 cores and 104 Gb RAM, running Linux OS. We run the Newton coordinate descent and alternating Newton coordinate descent algorithms as a single thread job on a single core. For our alternating Newton block coordinate descent method, we run it on a single core and with parallelization on 8 cores.

2.4.1 Synthetic Data Experiments

We compare the different methods on two sets of synthetic datasets, one for chain graphs and another for random graphs with clustering for $\Lambda$, generated as follows. For chain graphs, the true sparse parameters $\Lambda$ is set with $\Lambda_{i,i-1} = 1$ and $\Lambda_{i,i} = 2.25$ and the true $\Theta$ is set with $\Theta_{i,i} = 1$. We perform one set of chain graph experiments with $p = q$, and another set with an additional $q$ irrelevant features unconnected to any outputs, so that $p = 2q$. For random graphs with clustering, following the procedure in [41] for generating

Figure 3: Comparison of scalability on chain graphs. We vary $p$ and $q$, where (A) $p = q$ and (B) $p = 2q$. The Newton coordinate descent and alternating Newton coordinate descent methods could not be run beyond the problem sizes shown due to memory constraint. (C) Convergence when $q = 20,000$ and $p = 40,000$.

Figure 4: Comparison of scalability on random graphs with clustering. (A) Varying $p$ with $q$ fixed at 10,000. (B) Varying $q$ with $p$ fixed at 40,000. (C) Active set size versus time with $p = 20,000$ and $q = 10,000$.

In the best case, if the active set of $\Theta$ has a block structure, where each group of inputs are influencing only one group of outputs, overall $S_{xx}$ will be computed only once. We can further save the computation of $S_{xx}$, if the entire row of $\Theta$ is not in the active set, by entirely skipping the computation for the corresponding column of $S_{xx}$.
a GGM, we set the true $\Lambda$ to a graph with clusters of nodes of size 250 and with 90% of edges connecting randomly-selected nodes within clusters. We set the number of edges so that the average degree of each node is 10, with edge weights set to 1. We then set the diagonal values so that $\Lambda$ is positive definite. To set the sparse patterns for $\Theta$, we randomly select $100\sqrt{p}$ input variables as having edges to at least one output and distribute total $10q$ edges among those selected inputs to influence randomly selected outputs. We set the edge weights of $\Theta$ to 1.

Then, we draw samples from the CGGM defined by these true $\Lambda$ and $\Theta$. We generate datasets with $n = 100$ samples for the chain graphs and $n = 200$ samples for random graphs with clustering. We choose $\lambda_\Lambda$ and $\lambda_\Theta$ so that the number of edges in the estimated $\Lambda$ and $\Theta$ is close to ground truth. Following the strategy used in GGM estimation [41], we use the minimum-norm subgradient of the objective as our stopping criterion: $\|\text{grad}^S(\Lambda, \Theta_1)\|_1 < 0.01(\|\Lambda\|_1 + \|\Theta\|_1)$.

We compare the scalability of the different methods on chain graphs of different sizes. We show the computation times for datasets with $p = q$ in Figure 3A and for datasets with $p = 2q$ with $q$ additional irrelevant features in Figure 3B. For large problems, computation times are not shown for Newton coordinate descent and alternating Newton coordinate descent methods because they could not complete the optimization with limited memory. In addition, for large problems, alternating block coordinate descent was terminated after 60 hours of computation. We provide results on varying the sample size $n$ in Appendix. In Figure 3C, using the dataset with $p = 40,000$ and $q = 20,000$, we plot the suboptimality in the objective $f - f^*$ over time, where $f^*$ is obtained by running alternating Newton coordinate descent algorithm to numerical precision. Our new methods converge substantially faster than the previous approach, regardless of desired accuracy level. We notice that as expected from the convexity of the optimization problem, all algorithms converge to the global optimum and find nearly identical parameter estimates.

In Figure 4, we compare scalability of different methods for random graphs with clustering. In Figure 4A, we vary $p$, while setting $q$ to 10,000. In Figure 4B, we vary $q$, fixing $p$ to 40,000. Similar to the results from chain graph, for larger problems, Newton coordinate descent and alternating Newton coordinate descent methods ran out of memory and alternating block coordinate descent was terminated after 60 hours. For all problem sizes, our alternating Newton coordinate descent algorithm significantly reduces the computation time of the previous method, the Newton coordinate descent algorithm. This gap in the computation time increases for larger problems. In Figure 4C, we compare the convergence in sparsity pattern for the different methods as measured by the active set size, for $p = 20,000$ and $q = 10,000$. All our methods recover the optimal sparsity pattern much more rapidly than the previous approach.

Figures 3 and 4 show that our alternating Newton block coordinate descent can run on much larger problems than any other methods, while those methods without block coordinate descent run out of memory. For example, in Figure 4A alternating Newton block coordinate descent could handle problems with one million inputs, while without block-wise optimization it ran out of memory when $p > 100,000$. We also notice that on a single core, the alternating Newton block coordinate descent is slightly slower than the same method without block-wise optimization because of the need to recompute $\Sigma$ and $S_{xx}$. However, it is still substantially faster than the previous method.

Finally, we evaluate the parallelization scheme for our alternating Newton block coordinate descent method on multi-core machines. Given a dataset generated from cluster graph with $p = 40,000$ and $q = 20,000$, in Figure 5, we show the folds of speedup for different numbers of cores with respect to a single core.
We obtained about 7 times speedup on a 8-core machine with 104Gb RAM, and about 12 times speedup on a 16-core machine with 28Gb RAM. In general, we observe greater speedup on larger problems and for random graphs, because such problems tend to have more cache misses that can be computed in parallel.

We additionally analyzed the performance of our approach in scaling with the number of samples. We compare the performance of the different algorithms on synthetic datasets with different sample sizes \( n \), using a chain graph structure with \( p = q = 10,000 \). Figure 6A shows that our methods run significantly faster than the previous method across all sample sizes. In Figure 6B we measure the accuracy in recovering the true chain graph structure in terms of \( F_1 \)-score for different sample sizes \( n \). At convergence, \( F_1 \)-score was the same for all methods to three significant digits. As expected, the accuracy improves as the sample size increases.

### 2.4.2 Genomic Data Analysis

We compare the different methods on a genomic dataset. The dataset consists of genotypes for 442,440 single nucleotide polymorphisms (SNPs) and 10,256 gene expression levels for 171 individuals with asthma, after removing genes with variance < 0.01. We fit a sparse CGGM using SNPs as inputs and expressions as outputs to model a gene network influenced by SNPs. We also compared the methods on a smaller dataset of 34,249 SNPs from chromosome 1 and expression levels for 3,268 genes with variance > 0.1. As typically sparse model structures are of interests in this type of analysis, we chose regularization parameters so that the number of non-zero entries in each of \( \Theta \) and \( \Lambda \) at convergence was approximately 10 times the number
of genes. The alternating Newton block coordinate descent was run on an 8-core machine with parallelization. The computation time of different methods are provided in Table 1. On the largest problem, the previous approach could not run due to memory constraint, whereas our block coordinate descent converged in around 11 hours. We also compare the convergence of the different methods on the dataset with 34,249 SNPs and 3,268 gene expressions in Figure 7, and find that our methods provide vastly superior convergence than the previous method.

2.5 Conclusions for Optimization for Sparse CGGMs

In this chapter, we addressed the problem of large-scale optimization for sparse CGGMs. We proposed a new optimization procedure, called alternating Newton coordinate descent, that reduces computation time by alternately optimizing for the two sets of parameters Λ and Θ. Further, we extended this with block-wise optimization so that it can run on any machine with limited memory.
3 Sparse Gaussian Chain Graph Models

Probabilistic graphical models have been extensively studied as a powerful tool for modeling a set of conditional independencies in a probability distribution [47]. In this chapter, we are concerned with a class of graphical models, called chain graph models, that has been proposed as a generalization of undirected graphical models and directed acyclic graphical models [53, 25, 11]. Chain graph models are defined over chain graphs that contain a mixed set of directed and undirected edges but no partially directed cycles.

In particular, we study the problem of learning the structure of Gaussian chain graph models in a high-dimensional setting. While the problem of learning sparse structures from high-dimensional data has been studied extensively for other related models such as Gaussian graphical models (GGMs) [23] and more recently conditional Gaussian graphical models (CGGMs) [80, 94], to our knowledge, there is little previous work that addresses this problem for Gaussian chain graph models. Even with a known chain graph structure, current methods for parameter estimation are hindered by the presence of multiple locally optimal solutions [1, 20, 95].

Since the seminal work on conditional random fields (CRFs) [51], a general recipe for constructing chain graph models [47] has been given as using CRFs as building blocks for the model. We employ this construction for Gaussian chain graph models and propose to use the recently-introduced sparse CGGMs [80, 94] as a Gaussian equivalent of general CRFs. When the goal is to learn the model structure, we show that this construction is superior to the popular alternative approach of using linear regression as component models. Some of the key advantages of our approach are due to the fact that the sparse Gaussian chain graph models inherit the desirable properties of sparse CGGM such as convexity of the optimization problem and structured output prediction. In fact, our work is the first to introduce a joint estimation procedure for both the graph structure and parameters as a convex optimization problem, given the groups of variables for chain components. Another advantage of our approach is the ability to model a functional mapping from multiple related variables to other multiple related variables in a more natural way via moralization in chain graphs than other approaches that rely on complex penalty functions for inducing structured sparsity [62, 43].

Our work on sparse Gaussian chain graphs is motivated by problems in integrative genomic data analyses [85, 16]. While sparse GGMs have been extremely popular for learning networks from datasets of single modality such as gene-expression levels [23], we propose that sparse Gaussian chain graph models with CGGM components can be used to learn a cascade of networks by integrating multiple types of genomic data in a single statistical analysis. We show that our approach can reveal the module structures as well as the functional mapping between modules in different types of genomic data effectively. Furthermore, as the cost of collecting each data type differs, we show that semi-supervised learning can be used to make effective use of both fully-observed and partially-observed data.

3.1 Sparse Gaussian Chain Graph Models

We consider a chain graph model for a probability distribution over \( J \) random variables \( \mathbf{x} = \{x_1, \ldots, x_J\} \). The chain graph model assumes that the random variables are partitioned into \( C \) chain components \( \{x_1, \ldots, x_C\} \), the \( \tau \)th component having size \( |\tau| \). In addition, it assumes a partially directed graph structure, where edges between variables within each chain component are undirected and edges across two chain components are directed. Given this chain graph structure, the joint probability distribution factorizes as follows:

\[
p(\mathbf{x}) = \prod_{\tau=1}^{C} p(\mathbf{x}_\tau | \mathbf{x}_{pa(\tau)}),
\]

where \( \mathbf{x}_{pa(\tau)} \) is the set of variables that are parents of one or more variables in \( \mathbf{x}_\tau \). Each factor \( p(\mathbf{x}_\tau | \mathbf{x}_{pa(\tau)}) \) models the conditional distribution of the chain component variables \( \mathbf{x}_\tau \) given \( \mathbf{x}_{pa(\tau)} \). This model can also be viewed as being constructed with CRFs for \( p(\mathbf{x}_\tau | \mathbf{x}_{pa(\tau)}) \)'s [51].
As an alternative model for the regularization parameters that control the amount of sparsity in the parameters. As in MRCE [70], the structure and the parameters, we adopt sparse multivariate regression with covariance estimation (MRCE) when the chain component model has the form of Eq. (7), in order to jointly estimate the sparse graph and introduce our approach for learning chain graph model structures.

In this section, we focus on Gaussian chain graph models, where both $p(x)$ and $p(x_r | x_{pa(r)})$’s are Gaussian distributed. Below, we review linear regression models and CGGMs as chain component models, and introduce our approach for learning chain graph model structures.

### 3.1.1 Sparse Linear Regression as Chain Component Model

As the specific functional form of $p(x_r | x_{pa(r)})$ in Gaussian chain graphs models, a linear regression model with multivariate responses has been widely considered [5, 6, 20]:

$$p(x_r | x_{pa(r)}) = N(B_r x_{pa(r)}, \Omega_r^{-1}), \tag{7}$$

where $B_r \in \mathbb{R}^{\lvert \tau \rvert \times \lvert \text{pa}(\tau) \rvert}$ is the matrix of regression coefficients and $\Omega_r$ is the $\lvert \tau \rvert \times \lvert \tau \rvert$ inverse covariance matrix that models correlated noise. Then, the non-zero elements in $B_r$ indicate the presence of directed edges from $x_{pa(r)}$ to $x_r$, and the non-zero elements in $\Omega_r$ correspond to the undirected edges among the variables in $x_r$. When the graph structure is known, an iterative procedure has been proposed to estimate the model parameters, but it converges only to one of many locally-optimal solutions [20].

When the chain component model has the form of Eq. (7), in order to jointly estimate the sparse graph structure and the parameters, we adopt sparse multivariate regression with covariance estimation (MRCE) [70] for each chain component and solve the following optimization problem:

$$\min_{\tau = 1} \sum_{C} \text{tr}(B_r X_{\tau} - X_{pa(\tau)} B_r B_r^T \Omega_r X_r - X_{pa(r)} B_r^T)^T) - \lambda \sum_{\tau = 1} \sum_{C} || B_r ||_1 + \gamma \sum_{\tau = 1} \sum_{C} || \Omega_r ||_1,$$

where $X_\alpha \in \mathbb{R}^{N \times \alpha}$ is a dataset for $N$ samples, $|| \cdot ||_1$ is the sparsity-inducing $L_1$ penalty, and $\lambda$ and $\gamma$ are the regularization parameters that control the amount of sparsity in the parameters. As in MRCE [70], the problem above is not convex, but only bi-convex.

### 3.1.2 Sparse Conditional Gaussian Graphical Model as Chain Component Model

As an alternative model for $p(x_r | x_{pa(r)})$ in Gaussian chain graph models, a re-parameterization of the linear regression model in Eq. (7) with natural parameters has been considered [53]. This model also has been...
called a CGGM [80] or Gaussian CRF [94] due to its equivalence to a CRF. A CGGM for \( p(x_\tau | x_{\text{pa}(\tau)}) \) takes the standard form of undirected graphical models as a log-linear model:

\[
p(x_\tau | x_{\text{pa}(\tau)}) = \exp \left( - \frac{1}{2} x_\tau^T \Lambda_\tau x_\tau - x_\tau^T \Theta_{\tau, \text{pa}(\tau)} x_{\text{pa}(\tau)} \right) / A(x_{\text{pa}(\tau)}),
\]

where \( \Lambda_\tau \in \mathbb{R}^{|\tau| \times |\tau|} \) and \( \Theta_{\tau, \text{pa}(\tau)} \in \mathbb{R}^{|\tau| \times |\text{pa}(\tau)|} \) are the parameters for the feature weights between pairs of variables within \( x_\tau \) and between pairs of variables across \( x_\tau \) and \( x_{\text{pa}(\tau)} \), respectively, and \( A(x_{\text{pa}(\tau)}) \) is the normalization constant. The non-zero elements of \( \Lambda_\tau \) and \( \Theta_{\tau, \text{pa}(\tau)} \) indicate edges among the variables in \( x_\tau \) and between \( x_\tau \) and \( x_{\text{pa}(\tau)} \), respectively.

The linear regression model in Eq. (7) can be viewed as the result of performing inference in the probabilistic graphical model given by the CGGM in Eq. (8). This relationship between the two models can be seen by re-writing Eq. (8) in the form of a Gaussian distribution:

\[
p(x_\tau | x_{\text{pa}(\tau)}) = \mathcal{N}(-\Lambda_\tau^{-1} \Theta_{\tau, \text{pa}(\tau)} x_{\text{pa}(\tau)}, \Lambda_\tau^{-1}),
\]

where marginalization in a CGGM involves computing \( B_\tau x_{\text{pa}(\tau)} = -\Lambda_\tau^{-1} \Theta_{\tau, \text{pa}(\tau)} x_{\text{pa}(\tau)} \) to obtain a linear regression model parameterized by \( B_\tau \).

While maximum likelihood estimation leads to the equivalent parameter estimates for CGGMs and linear regression models via the transformation \( B_\tau = -\Lambda_\tau^{-1} \Theta_{\tau, \text{pa}(\tau)} \), imposing a sparsity constraint on each model leads to different estimates for the sparsity pattern of the parameters and the model structure [80]. The graph structure of a sparse CGGM directly encodes the probabilistic dependencies among the variables, whereas the sparsity pattern of \( B_\tau = -\Lambda_\tau^{-1} \Theta_{\tau, \text{pa}(\tau)} \) obtained after marginalization can be interpreted as indirect influence of covariates \( x_{\text{pa}(\tau)} \) on responses \( x_\tau \). As illustrated in Figures 8C and 8F, the CGGM parameters \( \Theta_{\tau, \text{pa}(\tau)} \) (directed edges with solid line) can be interpreted as direct dependencies between pairs of variables across \( x_\tau \) and \( x_{\text{pa}(\tau)} \), whereas \( B_\tau = -\Lambda_\tau^{-1} \Theta_{\tau, \text{pa}(\tau)} \) obtained from inference can be viewed as indirect and inferred dependencies (directed edges with dotted line).

We argue that when the goal is to learn the model structure, performing the estimation with CGGMs for chain component models can lead to a more meaningful representation of the underlying structure in data than imposing a sparsity constraint on linear regression models. Then the corresponding linear regression model can be inferred via marginalization. This approach also inherits many of the advantages of sparse CGGMs such as convexity of optimization problem.

### 3.2 Markov Properties and Chain Component Models

When a CGGM is used as the component model, the overall chain graph model is known to have Lauritzen-Wermuth-Frydenberg (LWF) Markov properties [25]. The LWF Markov properties also correspond to the standard probabilistic independencies in more general chain graphs constructed by using CRFs as building blocks [47].
Many previous works have noted that LWF Markov properties do not hold for the chain graph models with linear regression models [5, 6]. The alternative Markov properties (AMP) were therefore introduced as the set of probabilistic independencies associated with chain graph models with linear regression component models [5, 6]. It has been shown that the LWF and AMP Markov properties are equivalent only for chain graph structures that do not contain the graph in Figure 8A as a subgraph [5, 6]. For example, according to the LWF Markov property, in the chain graph model in Figure 8A, \( x_1 \perp x_3 | x_2 \) as \( x_1 \) and \( x_3 \) are separated by \( x_2 \) in the moralized graph in Figure 8B. However, the corresponding AMP Markov property implies a different probabilistic independence relationship, \( x_1 \perp x_3 | x_4 \).

We observe that when using sparse CGGMs as chain component models, we estimate a model with the LWF Markov properties and perform marginalization in this model to obtain a model with linear-regression chain components that can be interpreted with the AMP Markov properties.

### 3.3 Sparse Two-Layer Gaussian Chain Graph Models for Structured Sparsity

Another advantage of using CGGMs as chain component models instead of linear regression is that the moralized graph, which is used to define the LWF Markov properties, can be leveraged to discover the underlying structure in a correlated functional mapping from multiple inputs to multiple outputs. In this section, we show that a sparse two-layer Gaussian chain graph model with CGGM components can be used to learn structured sparsity. The key idea behind our approach is that while inference in CGGMs within the chain graph model can reveal the shared sparsity patterns for multiple related outputs, a moralization of the chain graph can reveal those for multiple inputs.

Statistical methods for learning models with structured sparsity were extensively studied in the literature of multi-task learning, where the goal is to find input features that influence multiple related outputs simultaneously [62, 43, 15]. Most of the previous works assumed the output structure to be known a priori. Then, they constructed complex penalty functions that leverage this known output structure, in order to induce structured sparsity pattern in the estimated parameters in linear regression models. In contrast, a sparse CGGM was proposed as an approach for performing a joint estimation of the output structure and structured sparsity for multi-task learning. As was discussed in Section 3.1.2, once the CGGM structure is estimated, the inputs relevant for multiple related outputs could be revealed via probabilistic inference in the graphical model.

While sparse CGGMs focused on leveraging the output structure for improved predictions, another aspect of learning structured sparsity is to consider the input structure to discover multiple related inputs jointly influencing an output. As CGGM is a discriminative model that does not model the input distribution, it is unable to capture input relatedness directly, although discriminative models in general are known to improve prediction accuracy. We address this limitation of CGGMs by embedding CGGMs within a chain graph and examining the moralized graph.

We set up a two-layer Gaussian chain graph model for inputs \( x \) and outputs \( y \) as follows:

\[
p(y, x) = p(y|x)p(x) = \left( \exp\left( -\frac{1}{2} y^T A_{yy} y - x^T \Theta_{xy} y / A_1(x) \right) \right) \left( \exp\left( -\frac{1}{2} x^T A_{xx} x / A_2 \right) \right),
\]

where a CGGM is used for \( p(y|x) \) and a GGM for \( p(x) \), and \( A_1(x) \) and \( A_2 \) are normalization constants. As the full model factorizes into two factors \( p(y|x) \) and \( p(x) \) with distinct sets of parameters, a sparse graph structure and parameters can be learned by using the optimization methods for sparse CGGM [94] and sparse GGM [42, 23].

The estimated Gaussian chain graph model leads to a GGM over both the inputs and outputs, which
Figure 9: Illustration of sparse two-layer Gaussian chain graphs with CGGMs. (A) A two-layer Gaussian chain graph. (B) The results of performing inference and moralization in (A). The dotted edges correspond to indirect dependencies inferred by inference. The edges among $x_j$'s represent the dependencies introduced by moralization.

reveals the structure of the moralized graph:

$$p(y, x) = N\left(0, \begin{pmatrix} \Lambda_{yy} & \Theta_{xy}^T \\ \Theta_{xy} & \Lambda_{xx} + \Theta_{xy} \Lambda_{yy}^{-1} \Theta_{xy}^T \end{pmatrix}^{-1}\right).$$

In the above GGM, we notice that the graph structure over inputs $x$ consists of two components, one for $\Lambda_{xx}$ describing the conditional dependencies within the input variables and another for $\Theta_{xy} \Lambda_{yy}^{-1} \Theta_{xy}^T$ that reflects the results of moralization in the chain graph. If the graph $\Lambda_{yy}$ contains connected components, the operation $\Theta_{xy} \Lambda_{yy}^{-1} \Theta_{xy}^T$ for moralization induces edges among those inputs influencing the outputs in each connected component.

Our approach is illustrated in Figure 9. Given the model in Figure 9A, Figure 9B illustrates the inferred structured sparsity for a functional mapping from multiple inputs to multiple outputs. In Figure 9B, the dotted edges correspond to inferred indirect dependencies introduced via marginalization in the CGGM $p(y|x)$, which reveals how each input is influencing multiple related outputs. On the other hand, the additional edges among $x_j$'s have been introduced by moralization $\Theta_{xy} \Lambda_{yy}^{-1} \Theta_{xy}^T$ for multiple inputs jointly influencing each output. Combining the results of marginalization and moralization, the two connected components in Figure 9B represent the functional mapping from $\{x_1, x_2\}$ to $\{y_1, y_2\}$ and from $\{x_3, x_4, x_5\}$ to $\{y_3, y_4, y_5\}$, respectively.

### 3.4 Sparse Multi-layer Gaussian Chain Graph Models

In this section, we extend the two-layer Gaussian chain graph model from the previous section into a multi-layer model to model data that are naturally organized into multiple layers. Our approach is motivated by problems in integrative genomic data analysis. In order to study the genetic architecture of complex diseases, data are often collected for multiple data types, such as genotypes, gene expressions, and phenotypes for a population of individuals [85, 16]. The primary goal of such studies is to identify the genotype features that influence gene expressions, which in turn influence phenotypes. In such problems, data can be naturally organized into multiple layers, where the influence of features in each layer propagates to the next layer in sequence. In addition, it is well-known that the expressions of genes within the same functional module are correlated and influenced by the common genotype features and that the coordinated expressions of gene modules affect multiple related phenotypes jointly. These underlying structures in the genomic data can be potentially revealed by inference and moralization in sparse Gaussian chain graph models with CGGM components.

In addition, we explore the use of semi-supervised learning, where the top and bottom layer data are fully observed but the middle-layer data are collected only for a subset of samples. In our application, genotype data and phenotype data are relatively easy to collect from patients’ blood samples and from observations. However, gene-expression data collection is more challenging, as invasive procedure such as surgery or biopsy is required to obtain tissue samples.
3.4.1 Models

Given variables, \( x = \{x_1, \ldots, x_J\}, y = \{y_1, \ldots, y_K\}, \) and \( z = \{z_1, \ldots, z_L\}, \) at each of the three layers, we set up a three-layer Gaussian chain graph model as follows:

\[
p(z, y|x) = p(z|y)p(y|x)
= \left(\exp\left(-\frac{1}{2}z^T \Lambda_{zz} z - y^T \Theta_{yz} z\right)/C_2(y)\right) \left(\exp\left(-\frac{1}{2}y^T \Lambda_{yy} y - x^T \Theta_{xy} y\right)/C_1(x)\right),
\]

where \( C_1(x) \) and \( C_2(y) \) are the normalization constants. In our application, \( x, y, \) and \( z \) correspond to genotypes, gene-expression levels, and phenotypes, respectively. As the focus of such studies lies on discovering how the genotypic variability influences gene expressions and phenotypes rather than the structure in genotype features, we do not model \( p(x) \) directly.

Given the estimated sparse model for Eq. (10), structured sparsity pattern can be recovered via inference and moralization. Computing \( B_{xy} = -\Lambda_{yx}^{-1} \Theta_{xy}^T \) and \( B_{yz} = -\Lambda_{zz}^{-1} \Theta_{yz}^T \) corresponds to performing inference to reveal how multiple related \( y_k \)’s in \( \Lambda_{yx} \) (or \( z_i \)’s in \( \Lambda_{zz} \)) are jointly influenced by a common set of relevant \( x_j \)’s (or \( y_k \)’s). On the other hand, the effects of moralization can be seen from the joint distribution \( p(z, y|x) \) derived from Eq. (10):

\[
p(z, y|x) = N\left(-\Lambda_{zz,yy}^{-1} \Theta_{yz,xy}^T x, \Lambda_{zz,yy}^{-1}\right),
\]

where \( \Theta_{yz,xy} = (0_{J \times L}, \Theta_{xy}) \) and \( \Lambda_{zz,yy} = \left(\begin{array}{cc} \Lambda_{zz} & \Theta_{yz}^T \\ \Theta_{yz} & \Lambda_{yy} + \Theta_{yz}^T \Lambda_{zz}^{-1} \Theta_{yz} \end{array}\right). \)

3.4.2 Semi-supervised Learning

Given a dataset \( \mathcal{D} = \{\mathcal{D}_o, \mathcal{D}_h\}, \) where \( \mathcal{D}_o = \{X_o, Y_o, Z_o\} \) for the fully-observed data and \( \mathcal{D}_h = \{X_h, Z_h\} \) for the samples with missing gene-expression levels, for semi-supervised learning, we adopt an EM algorithm that iteratively maximizes the expected log-likelihood of complete data:

\[
\mathcal{L}(\mathcal{D}_o; \Theta) + \mathbb{E}\left[\mathcal{L}(\mathcal{D}_h, Y_h; \Theta)\right],
\]

combined with \( L_1 \)-regularization, where \( \mathcal{L}(\mathcal{D}_o; \Theta) \) is the data log-likelihood with respect to the model in Eq. (10) and the expectation is taken with respect to:

\[
p(y|z, x) = N(\mu_{y|x,z}, \Sigma_{y|x,z}),
\]

\( \mu_{y|x,z} = -\Sigma_{y|x,z}(\Theta_{yz} z + \Theta_{xy}^T x) \) and \( \Sigma_{y|x,z} = (\Lambda_{yy} + \Theta_{yz} \Lambda_{zz}^{-1} \Theta_{yz}^T)^{-1}. \)

3.5 Experiments

In this section, we empirically demonstrate that CGGMs are more effective components for sparse Gaussian chain graph models than linear regression for various tasks, using synthetic and real-world genomic datasets. We used the sparse three-layer structure for \( p(z, y|x) \) in all our experiments.

3.5.1 Simulation Study

In simulation study, we considered two scenarios for true models, CGGM-based and linear-regression-based Gaussian chain graph models. We evaluated the performance in terms of graph structure recovery and prediction accuracy in both supervised and semi-supervised settings.
Figure 10: Illustration of the structured sparsity recovered by the model with CGGM components, simulated dataset. (A) $\Lambda_{zz}$. (B) $B_{yz} = -\Lambda_{zz}^{-1}\Theta_{yz}^T$ shows the effects of marginalization (white vertical bars). The effects of moralization are shown in (C) $\Lambda_{yy} + \Theta_{yz}\Lambda_{zz}^{-1}\Theta_{yz}^T$, and its decomposition into (D) $\Lambda_{yy}$ and (E) $\Theta_{yz}\Lambda_{zz}^{-1}\Theta_{yz}^T$.

In order to simulate data, we assumed the problem size of $J=500$, $K=100$, and $L=50$ for $x$, $y$, and $z$, respectively, and generated samples from known true models. Since we do not model $p(x)$, we used an arbitrary choice of multinomial distribution to generate samples for $x$. The true parameters for CGGM-based simulation were set as follows. We set the graph structure in $\Lambda_{yy}$ to a randomly-generated scale-free network with a community structure [91] with six communities. The edge weights were drawn randomly from a uniform distribution $[0.8, 1.2]$. We then set $\Lambda_{yy}$ to the graph Laplacian of this network plus small positive values along the diagonal so that $\Lambda_{yy}$ is positive definite. We generated $\Lambda_{zz}$ using a similar strategy, assuming four communities. $\Theta_{xy}$ was set to a sparse random matrix, where 0.4% of the elements have non-zero values drawn from a uniform distribution $[-1.2, -0.8]$. $\Theta_{yz}$ was generated using a similar strategy, with a sparsity level of 0.5%. We set the sparsity pattern of $\Theta_{yz}$ so that it roughly respects the functional mapping from communities in $y$ to communities in $z$. Specifically, after reordering the variables in $y$ and $z$ by performing hierarchical clustering on each of the two networks $\Lambda_{yy}$ and $\Lambda_{zz}$, the non-zero elements were selected randomly around the diagonal of $\Theta_{yz}$.

We set the true parameters for the linear-regression-based models using the same strategy as the CGGM-based simulation above for $\Lambda_{yy}$ and $\Lambda_{zz}$. We set $B_{xy}$ so that 50% of the variables in $x$ have non-zero influence on five randomly chosen variables in $y$ in one randomly chosen community in $\Lambda_{yy}$. We set $B_{yz}$ in a similar manner, assuming 80% of the variables in $y$ are relevant to eight randomly-chosen variables in $z$ from a randomly-chosen community in $\Lambda_{zz}$.

Each dataset consisted of 600 samples, of which 400 and 200 samples were used as training and test sets. To select the regularization parameters, we estimated a model using 300 samples, evaluated prediction errors on the other 100 samples in the training set, and selected the values with the lowest prediction errors. We used the optimization methods in [94] for CGGM-based models and the MRCE procedure [70] for linear-regression-based models.

Figure 10 illustrates how the model with CGGM chain components can be used to discover the structured sparsity via inference and moralization. In each panel, black and bright pixels correspond to zero and non-zero values, respectively. While Figure 10A shows how variables in $z$ are related in $\Lambda_{zz}$, Figure 10B shows $B_{yz} = -\Lambda_{zz}^{-1}\Theta_{yz}^T$ obtained via marginalization within the CGGM $p(z|y)$, where functional mappings from variables in $y$ to multiple related variables in $z$ can be seen as white vertical bars. In Figure 10C, the effects of moralization $\Lambda_{yy} + \Theta_{yz}\Lambda_{zz}^{-1}\Theta_{yz}^T$ are shown, which further decomposes into $\Lambda_{yy}$ (Figure 10D) and
Figure 11: Precision/recall curves for graph structure recovery in CGGM-based simulation study. (A) $\Lambda_{yy}$, (B) $\Lambda_{zz}$, (C) $B_{xy}$, (D) $B_{yz}$, and (E) $\Theta_{xy}$. (CG: CGGM-based models with supervised learning, CG-semi: CG with semi-supervised learning, LR: linear-regression-based models with supervised learning, LR-semi: LR with semi-supervised learning.)

Figure 12: Prediction errors in CGGM-based simulation study. The same estimated models in Figure 11 were used to predict (A) $y$ given $x, z$, (B) $z$ given $x$, (C) $y$ given $x$, and (D) $z$ given $y$.

Figure 13: Performance for graph structure recovery in linear-regression-based simulation study. Precision/recall curves are shown for (A) $\Lambda_{yy}$, (B) $\Lambda_{zz}$, (C) $B_{xy}$, and (D) $B_{yz}$.

Figure 14: Prediction errors in linear-regression-based simulation study. The same estimated models in Figure 13 were used to predict (A) $y$ given $x, z$, (B) $z$ given $x$, (C) $y$ given $x$, and (D) $z$ given $y$.

$\Theta_{yz} \Lambda_{zz}^{-1} \Theta_{yz}^T$ (Figure 10E). The additional edges among variables in $y$ in Figure 10E correspond to the edges introduced via moralization and show the groupings of the variables $y$ as the block structure along the diagonal. By examining Figures 10B and 10E, we can infer a functional mapping from modules in $y$ to modules in $z$.

In order to systematically compare the performance of the two types of models, we examined the average performance over 30 randomly-generated datasets. We considered both supervised and semi-supervised settings. Assuming that 200 samples out of the total 400 training samples were missing data for $y$, for supervised learning, we used only those samples with complete data; for semi-supervised learning, we used
all samples, including partially-observed cases.

The precision/recall curves for recovering the true graph structures are shown in Figure 11, using datasets simulated from the true models with CGGM components. Each curve was obtained as an average over 30 different datasets. We observe that in both supervised and semi-supervised settings, the models with CGGM components outperform the ones with linear regression components. In addition, the performance of the CGGM-based models improves significantly, when using the partially-observed data in addition to the fully-observed samples (the curve for CG-semi in Figure 11), compared to using only the fully-observed samples (the curve for CG in Figure 11). This improvement from using partially-observed data is substantially smaller for the linear-regression-based models. The average prediction errors from the same set of estimated models in Figure 11 are shown in Figure 12. The CGGM-based models outperform in all prediction tasks, because they can leverage the underlying structure in the data and estimate models more effectively.

For the simulation scenario using the linear-regression-based true models, we show the results for precision/recall curves and prediction errors in Figures 13 and 14, respectively. We find that even though the data were generated from chain graph models with linear regression components, the CGGM-based methods perform as well as or better than the other models.

We also examined the performance of the CGGM-based models, as we varied the number of samples with missing observations for $y$. Figure 15 shows precision/recall curves averaged over 30 datasets, using datasets simulated from CGGM-based true models. We find that the accuracy for recovering the true graph structures improves, as more data are available for $y$, although the recovery of $\Lambda_{zz}$ remains unaffected by the number of samples with missing data.

### Table 2: Prediction errors, mouse diabetes data

<table>
<thead>
<tr>
<th>Task</th>
<th>CG-semi</th>
<th>CG</th>
<th>LR-semi</th>
<th>LR</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y \mid x, z$</td>
<td>0.9070</td>
<td>0.9996</td>
<td>1.0958</td>
<td>0.9671</td>
</tr>
<tr>
<td>$z \mid x$</td>
<td>1.0661</td>
<td>1.0585</td>
<td>1.0505</td>
<td>1.0614</td>
</tr>
<tr>
<td>$y \mid x$</td>
<td>0.8989</td>
<td>0.9382</td>
<td>0.9332</td>
<td>0.9103</td>
</tr>
<tr>
<td>$z \mid y$</td>
<td>1.0712</td>
<td>1.0861</td>
<td>1.1095</td>
<td>1.0765</td>
</tr>
</tbody>
</table>

#### 3.5.2 Integrative Genomic Data Analysis

We applied the two types of three-layer chain graph models to single-nucleotide-polymorphism (SNP), gene-expression, and phenotype data from the pancreatic islets study for diabetic mice [85]. We selected 200 islet gene-expression traits after performing hierarchical clustering to find several gene modules. Our dataset also included 1000 SNPs and 100 pancreatic islet cell phenotypes. Of the total 506 samples, we used 406 as training set, of which 100 were held out as a validation set to select regularization parameters, and
used the remaining 100 samples as test set to evaluate prediction accuracies. We considered both supervised and semi-supervised settings, assuming gene expressions are missing for 150 mice. In supervised learning, only those samples without missing gene expressions were used.

As can be seen from the prediction errors in Table 2, the models with CGGM chain components are more accurate in various prediction tasks. In addition, the CGGM-based models can more effectively leverage the samples with partially-observed data than linear-regression-based models.

3.6 Conclusions for Sparse Gaussian Chain Graph Models

In this chapter, we addressed the problem of learning the structure of Gaussian chain graph models in a high-dimensional space. We argued that when the goal is to recover the model structure, using sparse CGGMs as chain component models has many advantages such as recovery of structured sparsity, computational efficiency, globally-optimal solutions for parameter estimates, and superior performance in semi-supervised learning.
4 Asthma Perturb-Net Analysis

4.1 Material and Methods

We describe the model and learning/inference algorithms for Perturb-Net for learning the gene network under SNP perturbations that underlies clinical phenotypes.

4.1.1 Efficient implementation of EM algorithm for semi-supervised learning

The standard EM algorithm iterates between an M-step for finding the parameter estimate maximizing the expected log-likelihood (or minimizing the negative log-likelihood) and an E-step for finding the expected sufficient statistics based on the posterior probability distribution in Eq. (3.4.2). The M-step is carried out by using our Mega-sCGGM algorithm. In E-step, a naive inversion of \( \Lambda_{yy} + \Theta_{yz} \Lambda_{zz}^{-1} \Theta_{yz}^T \) to obtain \( \Sigma_{y|x,z} \) is expensive and storage of this dense matrix may exceed computer memory for large gene expression datasets. We reduce the time cost and avoid memory limit in E-step, assuming that the number of phenotypes \( r \) is relatively small compared to the number of genes (i.e., \( r << q \)), which is typical for most studies. Instead of explicitly performing the E-step, we embed the E-step within the M-step, such that the E-step results are represented implicitly to fit in memory and computed explicitly on-demand as needed in the M-step. Specifically, instead of performing the full E-step, we implicitly represent \( \Lambda_{yy} + \Theta_{yz} \Lambda_{zz}^{-1} \Theta_{yz}^T \) as \( \Lambda_{yy} + KK^T \), using low-rank component \( K = \Theta_{yz}L_z^T \) and the sparse Cholesky factorization of trait network \( L_zL_z^T = \Lambda_{zz} \). Then, during M-step, we invert \( \Lambda_{yy} + KK^T \), one column at a time as needed, using the conjugate gradient method. This modified EM algorithm is equivalent to the original EM algorithm that iterates between an M-step and an E-step, producing the same estimate.

4.1.2 Perturb-Net inference procedures

While the sparse Gaussian chain graph model explicitly represents pair-wise dependencies among variables as edges in the graph, there are other dependencies that are only implicitly represented in the model but can be revealed by performing an inference on the estimated probabilistic graphical model. Here, we provide an overview of the inferred dependencies, all of which involve simple matrix operations.

The following two inference methods directly follow from the inference method for a sparse CGGM (Figure 16A) [81, 98], which infers the indirect perturbation effects that arise from the direct perturbation effects propagating to other parts of the network.

- **Indirect SNP perturbation effects on gene expression levels:** \( B_{xy} = -\Theta_{xy} \Lambda_{yy}^{-1} \), where \( [B_{xy}]_{i,j} \) represents the indirect perturbation effect of SNP \( i \) on the expression level of gene \( j \) (blue dashed arrow in Figure 16A). This can be seen by deriving the marginal distribution from the sparse CGGM component model \( p(y|x) \) as follows:
  
  \[
  p(y|x) = N(B_{xy}^T x, \Lambda_{yy}^{-1}).
  \]  
  
  From Eq. (11), the marginal distribution for the expression level \( [y]_i \) of gene \( i \) can be obtained as \( p([y]_i|x) = N([B_{xy}]_{i,i}^T x, [\Lambda_{yy}^{-1}]_{i,i}) \). While \( [\Theta_{xy}]_{i,j} \) represents the direct perturbation effect of SNP \( i \) on the expression of gene \( j \), \( [B_{xy}]_{i,j} \) represents the overall perturbation effect that aggregates all indirect influence of this SNP on gene \( j \) through other genes. When SNP \( i \) does not influence the expression of gene \( j \) directly but exerts influence on gene \( j \) through other genes connected to gene \( j \) in the network \( \Lambda_{yy} \), we have \( [\Theta_{xy}]_{i,j} = 0 \) but \( [B_{xy}]_{i,j} \neq 0 \).

- **Indirect effects of gene expression levels on clinical phenotypes:** \( B_{yz} = -\Theta_{yz} \Lambda_{zz}^{-1} \), where \( [B_{yz}]_{i,j} \) represents the indirect influence of the expression level of gene \( i \) on phenotype \( j \) (red dashed arrow in Figure 16A). This can be seen by deriving the marginal distribution from the sparse CGGM component model \( p(y|x) \) as follows:
  
  \[
  p(y|x) = N(B_{yz}^T x, \Lambda_{zz}^{-1}).
  \]  

29
Figure 16: Illustration of the Perturb-Net approach. (A) Perturb-Net uses a sparse Gaussian chain graph model with a cascade of two sparse CGGMs, one for a gene network influenced by SNPs (blue solid edges and nodes) and the other for a clinical trait network influenced by gene expression levels (red solid edges and nodes). The sparse CGGM inference procedures can be used to infer hidden interactions in each of the two component sparse CGGMs, such as the indirect effect of SNP $x_1$ on expression level $y_2$ through expression level $y_3$ (blue dashed arrow) and the indirect effect of expression level $y_3$ on phenotype $z_2$ through phenotype $z_3$ (red dashed arrow). (B) The inference procedures of sparse Gaussian chain graph models are used to infer the information on how the gene network mediates SNP effects on phenotypes. Examples of such inferred interactions are shown for the perturbation effect of SNP $x_1$ on phenotype $z_3$ (purple dashed arrow), which can be decomposed into two components mediated by each of the two gene modules (yellow and orange nodes), and the inferred dependencies between expression level $y_3$ and expression level $y_{14}$ (blue dashed line) induced by phenotype $z_3$ in the posterior gene network, after seeing the clinical phenotypes.
arrow in Figure 16A). Similarly as above, this can be seen by deriving the marginal distribution from the sparse CGGM component model \( p(z|y) \), as follows:

\[
p(z|y) = N(B_{yz}^T, \Lambda_{zz}^{-1}).
\]

Then, the marginal distribution for \([z]_i\) of phenotype \(i\) can be obtained as \(p([z]_i|y) = N(\left([B_{yz}]_{i,:} \right)^T y, [\Lambda_{zz}^{-1}]_{i,i})\). While \([\Theta_{yz}]_{i,j}\) represents the direct influence of gene \(i\) on phenotype \(j\), \([B_{yz}]_{i,j}\) represents the overall influence that aggregates all indirect influence of this expression level on phenotype \(j\) through other phenotypes.

The sparse Gaussian chain graph model provides the following additional inference procedures for extracting the information on whether SNP perturbation effects on the gene network reach the phenotypes and how different genes or subnetworks of the gene network mediate SNP effects on phenotypes (Figure 16B).

- **SNP effects on clinical phenotypes**: \(B_{xz} = B_{xy}B_{yz}\), where \([B_{xz}]_{i,j}\) represents the overall influence of SNP \(i\) on phenotype \(j\) mediated by gene expression levels in gene network \(\Lambda_{yy}\) (purple dashed arrow in Figure 16B). The effects of SNPs on phenotypes are not directly modeled in our model but can be inferred by deriving the marginal distribution \(p(z|x)\) as follows:

\[
p(z|x) = N(B_{xz}^T x, \Lambda_{zz}^{-1} + \Lambda_{zz}^{-1} \Theta_{yz}^T \Lambda_{yy}^{-1} \Theta_{yz} \Lambda_{zz}^{-1}).
\]

The marginal distribution for the phenotype \([z]_i\) of phenotype \(i\) given \(x\) can be obtained as \(p([z]_i|x) = N(\left([B_{xz}]_{i,:} \right)^T x, [\Lambda_{zz}^{-1} + \Lambda_{zz}^{-1} \Theta_{yz}^T \Lambda_{yy}^{-1} \Theta_{yz} \Lambda_{zz}^{-1}]_{i,i})\), where each element \([B_{xz}]_{i,j}\) represents the overall influence of SNP \(i\) on phenotype \(j\) mediated by the gene network in \(\Lambda_{yy}\) and other phenotypes connected to phenotype \(j\) in \(\Lambda_{zz}\).

- **SNP effects on clinical phenotypes mediated by a gene module**: The overall SNP effects on phenotypes in \(B_{xz}\) above can be decomposed into the SNP effects on phenotypes mediated by each gene module. Let \(M\) be a gene module that consists of a subset of the \(q\) genes whose expression levels were modeled in \(\Lambda_{yy}\) (yellow and orange gene modules in Figure 16B). Then, the effects of SNPs on phenotypes mediated by the genes in module \(M\) can be obtained as follows:

\[
B_{xz}^M = \sum_{k \in M} [B_{xy}]_{:,k}[B_{yz}]_{k,:},
\]

where \([B_{xy}]_{a,:}\) represents the \(a\)th row of \(B_{xy}\) and \([B_{yz}]_{b,:)\) represents the \(b\)th column of \(B_{yz}\). In the above equation, \([B_{xz}^M]_{i,j}\) quantifies the effect of SNP \(i\) on phenotype \(j\) through the expression levels of genes in module \(M\). If \(M_1, \ldots, M_s\) are disjoint subsets of \(q\) genes, where \(\bigcup_{m=1}^{s} M_m\) is the full set of \(q\) genes, we have the following decomposition:

\[
B_{xz} = \sum_{m=1}^{s} B_{xz}^{M_m}.
\]

- **Inferred dependencies among genes after seeing phenotype data**: \(\Lambda_{yy|x,z} = \Lambda_{yy} + \Theta_{yz} \Lambda_{zz}^{-1} \Theta_{yz}^T\) represents gene network \(\Lambda_{yy}\) augmented with the component \(\Theta_{yz} \Lambda_{zz}^{-1} \Theta_{yz}^T\) introduced through dependencies in phenotype network \(\Lambda_{zz}\) (blue dashed edge in Figure 16B). In this augmented network, additional edges are introduced between two genes if their expression levels influence the same trait or if they both affect traits that are connected in the phenotype network \(\Lambda_{zz}\). The posterior gene network
\( \Lambda_{y|x,z} \), which contains the dependencies among expression levels after taking into account phenotype data, can be obtained by inferring the posterior distribution given phenotypes from the estimated Gaussian chain graph model as follows:

\[
p(y|x, z) = N\left(-\left(z^T \Omega_{yz}^T + x^T \Theta_{xy}\right) \Lambda_{y|x,z}^{-1}, \Lambda_{y|x,z}^{-1}\right),
\]

where

\[
\Lambda_{y|x,z} = \Lambda_{yy} + \Theta_{yz} \Lambda_{zz}^{-1} \Theta_{yz}^T.
\]

The inferred network \( \Lambda_{y|x,z} \) can also be seen by inferring from the estimated model the joint distribution

\[
p(z, y|x) = N\left(-\Lambda_{zz}^{-1}(zz, yy) \Theta_{yx}^T \left(z^T \Lambda_{zz}^{-1} (yz, xy) x, \Lambda_{yy}^{-1}(zz, yy)\right), \right),
\]

where \( \Theta_{yx} = (0_{p \times r}, \Theta_{xy}) \) with \( p \times r \) matrix of 0’s and \( \Lambda_{(zz, yy)} = \left(\Lambda_{zz} \Theta_{yz}^T \Lambda_{yz} \Lambda_{y|x,z}\right) \). This joint distribution is an alternative representation of the same Gaussian chain graph model in Eq. (10) and corresponds to another sparse CGGM over \( y \) and \( z \) conditional on \( x \). This process of introducing the additional dependencies via \( \Theta_{yz} \Lambda_{zz}^{-1} \Theta_{yz}^T \) in this new sparse CGGM, which is equivalent to the original chain graph model, is also known as moralization in the probabilistic graphical model literature [48].

### 4.1.3 Prediction tasks

We use the estimated Perturb-Net model and the results of probabilistic inference on this model to make predictions on previously unseen patients. From each of the two component sparse CGGMs in our model, we make the following predictions:

- \( \hat{y}_{\text{new}}|x_{\text{new}} = B_{xy}^T x_{\text{new}} \) for predicting the expression levels \( \hat{y}_{\text{new}} \) given the genotypes \( x_{\text{new}} \) of a new patient
- \( \hat{z}_{\text{new}}|y_{\text{new}} = B_{yz}^T y_{\text{new}} \) for predicting the phenotypes \( \hat{z}_{\text{new}} \) given the expression levels \( y_{\text{new}} \) of a new patient

From the full sparse Gaussian chain graph model, we make the following predictions:

- \( \hat{z}_{\text{new}}|x_{\text{new}} = B_{xz}^T x_{\text{new}} \) for predicting the phenotypes \( \hat{z}_{\text{new}} \) given the genotypes \( x_{\text{new}} \) of a new patient
- \( \hat{y}_{\text{new}}|x_{\text{new}}, z_{\text{new}} = -\left(z_{\text{new}}^T \Theta_{yz}^T + x_{\text{new}}^T \Theta_{xy}\right) \Lambda_{y|x,z}^{-1} \) for predicting the gene expression levels \( \hat{y}_{\text{new}} \) given the genotypes \( x_{\text{new}} \) and the phenotypes \( z_{\text{new}} \) of a new patient

### 4.1.4 Lasso for comparison with our algorithms

We compare the performance of our method with that of Lasso [84, 92], a popular statistical method based on linear regression models for studying the associations among SNPs, expression measurements, and phenotypes. We begin by setting up a two-layer multivariate regression model for genotypes \( x \in \{0, 1, 2\}^p \), expression measurements \( y \in \mathbb{R}^q \), and phenotypes \( z \in \mathbb{R}^r \) as follows:

\[
y = \Lambda_{xy}^T x + \epsilon_y, \quad \epsilon_y \sim \mathcal{N}(0_q, \Omega_y),
\]

\[
z = \Lambda_{yz}^T y + \epsilon_z, \quad \epsilon_z \sim \mathcal{N}(0_r, \Omega_z),
\]
where $A_{xy} \in \mathbb{R}^{p \times q}$ and $A_{yz} \in \mathbb{R}^{q \times r}$ are regression coefficients, $\epsilon_y \in \mathbb{R}^q$ and $\epsilon_z \in \mathbb{R}^r$ are noise distributed with zero means and diagonal covariances $\Omega_y = \text{diag}(\sigma^2_{y_1}, \ldots, \sigma^2_{y_q})$ and $\Omega_z = \text{diag}(\sigma^2_{z_1}, \ldots, \sigma^2_{z_q})$.

Given genotype data $X \in \{0, 1, 2\}^{n \times p}$ for $n$ samples and $p$ SNPs, expression data $Y \in \mathbb{R}^{n \times q}$ for $q$ genes, and phenotype data $Z \in \mathbb{R}^{n \times r}$ for $r$ phenotypes, we obtain a Lasso estimate of the regression coefficients by minimizing $L_1$-regularized negative log-likelihood as follows:

\[
\min_{A_{xy}} \frac{1}{n} \text{tr} \left( (Y - X^T A_{xy}) (Y - X^T A_{xy})^T \right) + \gamma_1 ||A_{xy}||_1,
\]

\[
\min_{A_{yz}} \frac{1}{n} \text{tr} \left( (Z - Y^T A_{yz}) (Z - Y^T A_{yz})^T \right) + \gamma_2 ||A_{yz}||_1.
\]

Using the Lasso estimate of the regression coefficients $A_{xy}$ and $A_{yz}$, we compute predictions for this model analogously to our sparse Gaussian chain graph model.

- $\hat{y}_{\text{new}} | x_{\text{new}} = A_{xy}^T x_{\text{new}}$
- $\hat{z}_{\text{new}} | y_{\text{new}} = A_{yz}^T y_{\text{new}}$
- $\hat{z}_{\text{new}} | x_{\text{new}} = A_{xz}^T x_{\text{new}}$, where $A_{xz} = A_{xy} A_{yz}$.
- $\hat{y}_{\text{new}} | x_{\text{new}}, z_{\text{new}} = (z_{\text{new}}^T A_{yz} + x_{\text{new}}^T A_{xy} \Omega_z) \Omega_{y|x,z}^{-1}$, where $\Omega_{y|x,z} = ([\Omega_y]^{-1} + A_{yz} \Omega_z^{-1} A_{yz}^T)^{-1}$.

For this prediction task, we estimate the variances as follows [68]:

\[
\sigma^2_{y_i} = \frac{1}{n - s_{y_i}} \left( [Y]_{:,i} - X^T [A_{xy}]_{:,i} \right)^T \left( [Y]_{:,i} - X^T [A_{xy}]_{:,i} \right), \quad \text{for } i = 1, \ldots, q,
\]

\[
\sigma^2_{z_i} = \frac{1}{n - s_{z_i}} \left( [Z]_{:,i} - Y^T [A_{yz}]_{:,i} \right)^T \left( [Z]_{:,i} - Y^T [A_{yz}]_{:,i} \right), \quad \text{for } i = 1, \ldots, r,
\]

where $s_{y_i}$ and $s_{z_i}$ are the numbers of non-zero entries in $[A_{xy}]_{:,i}$ and $[A_{yz}]_{:,i}$ respectively.

### 4.1.5 Preparation of asthma dataset

We applied our method to a dataset comprising genotype, gene expression, and clinical phenotype data, collected from asthma patients participating in CAMP study [33, 34, 58]. We used 174 non-Hispanic Caucasian subjects for whom both genotype and clinical phenotype data were available. For a subset of 140 individuals, gene expression data from primary peripheral blood CD4+ lymphocytes were also available. After removing SNPs with minor allele frequency less than 0.1 and those with missing reference SNP ids, we obtained 495,597 SNPs for autosomal chromosomes. We then imputed missing genotypes using fastPHASE [74]. Given expression levels for 22,184 mRNA transcripts profiled with Illumina HumanRef8 v2 BeadChip arrays [58], we removed transcript levels with expression variance less than 0.01, which resulted in a set of 11,598 expression levels to be used in our analysis. Then, we converted the expression values to their $z$-scores. The clinical phenotype data comprised 35 phenotypes (Table 3), including 25 features related to lung function and 10 features collected via blood testing. The clinical phenotypes were converted to their $z$-scores within each phenotype so that all phenotypes have equal variance. We then imputed missing values using low-rank matrix completion [14].

### 4.1.6 Comparison of the computation time of different algorithms

In order to compare the computation of different algorithms, we used the following software and hardware setup. For Lasso, we used the implementation in GLMNET [67] with a backend written in Fortran. For Newton coordinate descent, which is the previous state-of-the-art approach for optimizing sparse CGGMs,
Table 3: Description of asthma clinical phenotypes in CAMP data.

<table>
<thead>
<tr>
<th>Phenotype Name</th>
<th>Phenotype Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREFVCP</td>
<td>lung</td>
<td>Baseline FVC % predicted</td>
</tr>
<tr>
<td>HPRFVCP</td>
<td>lung</td>
<td>Hankinson pre BD FVC % predicted</td>
</tr>
<tr>
<td>HPRFVCPC</td>
<td>lung</td>
<td>Hankinson pre BD FVC percentile</td>
</tr>
<tr>
<td>HPRFVCZS</td>
<td>lung</td>
<td>Hankinson pre BD FVC z-score</td>
</tr>
<tr>
<td>PREFEVPP</td>
<td>lung</td>
<td>Pre BD FEV % predicted</td>
</tr>
<tr>
<td>HPRFEVPP</td>
<td>lung</td>
<td>Hankinson pre BD FEV % predicted</td>
</tr>
<tr>
<td>HPRFEVPC</td>
<td>lung</td>
<td>Hankinson pre BD FEV percentile</td>
</tr>
<tr>
<td>HPRFEVZS</td>
<td>lung</td>
<td>Hankinson pre BD FEV z-score</td>
</tr>
<tr>
<td>PO1FEVPP</td>
<td>lung</td>
<td>1st Post BD FEV % predicted</td>
</tr>
<tr>
<td>HPO1FEPP</td>
<td>lung</td>
<td>1st Hankinson post BD FEV % predicted</td>
</tr>
<tr>
<td>HPO1FEPC</td>
<td>lung</td>
<td>1st Hankinson post BD FEV percentile</td>
</tr>
<tr>
<td>HPO1FEZS</td>
<td>lung</td>
<td>1st Hankinson post BD FEV z-score</td>
</tr>
<tr>
<td>PREFF</td>
<td>lung</td>
<td>Pre BD FEV/FVC ratio (%)</td>
</tr>
<tr>
<td>HPRFFPC</td>
<td>lung</td>
<td>Hankinson pre BD FEV/FVC percentile</td>
</tr>
<tr>
<td>HPRFFZS</td>
<td>lung</td>
<td>Hankinson pre BD FEV/FVC z-score</td>
</tr>
<tr>
<td>MC928</td>
<td>lung</td>
<td>Baseline (pre-diluent) FEV1/FVC ratio</td>
</tr>
<tr>
<td>PREFVC</td>
<td>lung</td>
<td>Pre BD FVC</td>
</tr>
<tr>
<td>PREFEV</td>
<td>lung</td>
<td>Pre BD FEV</td>
</tr>
<tr>
<td>PO1FEV</td>
<td>lung</td>
<td>1st Post BD FEV</td>
</tr>
<tr>
<td>MC935b</td>
<td>lung</td>
<td>FEV1 15 minutes after 2puffs albuterol</td>
</tr>
<tr>
<td>PREPF</td>
<td>lung</td>
<td>Pre BD peak flow</td>
</tr>
<tr>
<td>POSPF</td>
<td>lung</td>
<td>Post BD peak flow</td>
</tr>
<tr>
<td>lnpc20</td>
<td>lung</td>
<td>Airway responsiveness to methacholine</td>
</tr>
<tr>
<td>rescueBD7day</td>
<td>lung</td>
<td>Rescue BD use last 7 days</td>
</tr>
<tr>
<td>preventBD7day</td>
<td>lung</td>
<td>Preventative BD use last 7 days</td>
</tr>
<tr>
<td>HEMOG</td>
<td>blood</td>
<td>Hemoglobin</td>
</tr>
<tr>
<td>WBC</td>
<td>blood</td>
<td>white blood cell count</td>
</tr>
<tr>
<td>MONOPCT</td>
<td>blood</td>
<td>Monocytes %</td>
</tr>
<tr>
<td>NEUTPCT</td>
<td>blood</td>
<td>Neutrophils (segs, polys) %</td>
</tr>
<tr>
<td>LYMPHPCCT</td>
<td>blood</td>
<td>Lymphocytes %</td>
</tr>
<tr>
<td>BASOPCT</td>
<td>blood</td>
<td>Basophils %</td>
</tr>
<tr>
<td>TOTEOSP</td>
<td>blood</td>
<td>Total eosinophils count by % of WBC</td>
</tr>
<tr>
<td>EOSPCT</td>
<td>blood</td>
<td>Eosinophils %</td>
</tr>
<tr>
<td>log10ige</td>
<td>blood</td>
<td>Logarithm transformed IgE level</td>
</tr>
<tr>
<td>log10eos</td>
<td>blood</td>
<td>Logarithm transformed eosinophil level</td>
</tr>
</tbody>
</table>

we took the implementation written in C++ provided by the authors [94] and sped up this implementation with the Eigen matrix library, by employing low-rank matrix representations and using sparse matrix multiplications. For all methods, the code was compiled and run with OpenMP multi-threading enabled on the same machines with 20Gb of memory and 16 cores. We used the same regularization parameters for our method and the previous method for sparse CGGM optimization, so the resulting solutions were identical with the same sparsity levels. For Lasso, we chose the regularization parameters so that the $L_1$-norm of the regression matrix roughly matched that of our inferred indirect SNP effects.
4.2 Comparison of the scalability of Mega-sCGGM and other methods

![Comparison of computation time of different methods. The computation time of our Mega-sCGGM is compared with that of previous learning algorithm for sparse CGGMs and Lasso. We applied all methods to all expression data and genotype data from chromosome 1, chromosomes 1-6, chromosomes 1-16, and chromosomes 1-22. The previous algorithm for sparse CGGMs ran out of memory at chromosome 1, so we obtained its computation time with much smaller datasets with 1,000 and 10,000 SNPs.](image)

We assess the scalability of Mega-sCGGM and other previous algorithms on the expression measurements of 11,598 genes and the genotypes of 495,597 SNPs for 140 subjects from the CAMP data. We estimated sparse CGGMs, using both our new method and the previous state-of-the-art method based on the Newton coordinate descent method [94]. Since the sparse CGGM optimization problem is convex with a single globally optimal solution, both our and previous methods obtain the same parameter estimates, although the computation time differs between the two methods. We also obtained the computation time of Lasso implemented in GLMNET [67, 84], the well-known computationally efficient algorithm for learning a simple but less powerful regression model. Although the sparse multivariate regression with covariance estimation [71] has also provided a methodology that could be used for learning a gene network influenced by SNPs, this approach has been found to take days to learn a model from a small dataset of only 1,000 SNPs and 500 gene expression levels [98], so we did not include it in our experiment. All of the optimization methods were run on the same hardware setup with comparable software implementations.

In our comparison of different methods, our algorithm significantly outperformed the previous state-of-the-art method for learning a sparse CGGM in terms of both computation time and memory requirement and scaled similarly to Lasso (Figure 17). In comparison of our method with Lasso on datasets with 40,056 SNPs from chromosome 1, 21,757 SNPs for chromosomes 1 through 6, and 495,597 SNPs from all autosomal chromosomes and all expression measurements, our method was not substantially slower than Lasso, even
though our method learns a more expressive model than Lasso. The previous CGGM optimization algorithm ran out of memory even on the smallest dataset above with SNPs only from chromosome 1, so we compared the two algorithms on a much smaller dataset with 1,000 and 10,000 SNPs. On 10,000 SNPs, the previous algorithm required more than four hours, whereas in less than four hours, our algorithm was able to run on all 495,597 SNPs.

4.3 Analysis of asthma data

We now fit a sparse Gaussian chain graph model to the genotype, expression, clinical phenotype data gathered from participants in the Childhood Asthma Management Program (CAMP) [33, 34, 58]. After preprocessing the data, we applied our method to the data from 140 subjects for whom all data were available for 495,597 SNPs on 22 autosomal chromosomes, 11,598 gene expression levels, and 35 phenotypes (Table 3) and 34 additional subjects for whom data were available only for genotypes and phenotypes but not for expression levels. Below we perform a detailed analysis of the estimated model.

4.3.1 Overview of the Perturb-Net model for asthma

We first examined the overall estimated model for the module structures in the phenotype and gene networks (Figure 18). To see the structure in the phenotype network $\Lambda_{zz}$, we reordered the nodes of the network by applying hierarchical clustering to each set of the lung function and blood test phenotypes. This revealed the dense connectivities within the two known groups of phenotypes and the two sub-clusters within the group of lung function phenotypes (Figure 18A).

The gene network $\Lambda_{yy}$ also showed a clear module structure (Figure 18B). To find the module structure in the network, we identified the genes that are connected to at least one other gene in the network $\Lambda_{yy}$ and partitioned the network over those genes into 20 subnetworks with roughly equal number of nodes, using the network clustering algorithm METIS [45]. Out of 11,598 genes, 6,102 genes were connected to at least one other gene in the network. For the rest of our analysis, we focus on the network and modules over the 6,102 genes, since these genes are likely to form modules for pathways with a functional impact on asthma phenotypes. Modules 1-15 were densely connected clusters of co-expressed genes, suggesting those modules are likely to consist of a functionally coherent set of genes, whereas modules 16-20 had relatively fewer edge connections within each cluster.

Next, we considered the effects of the gene modules on the lung and blood phenotypes in $\Theta_{yz}$ and the SNP perturbations of the gene modules in $\Theta_{xy}$. Modules 1-12 had relatively small effects on the phenotypes despite their dense connectivities, whereas modules 13-20 appeared to have stronger effects on both groups of phenotypes (Figure 18C). The SNP effects on the modules in $\Theta_{xy}$ for the top 1000 eQTL hotspots, determined by overall SNP effects on all genes ($\sum_j |[\Theta_{xy}]_{i,j}|$ for each SNP $i$), showed that many of these hotspots perturb the expression of genes in the same module in the gene network (Figure 18D). Given these observations from the visual inspection of $\Theta_{yz}$ and $\Theta_{xy}$, we summarized $\Theta_{yz}$ and $\Theta_{xy}$ at module level and compared the module-level summaries across modules. To quantify the module-level influence of expression levels on each group of phenotypes, from the direct influence $\Theta_{yz}$ and indirect influence $B_{yz}$ we computed the overall effect sizes of all genes in the given gene module on all phenotypes in each of the lung and blood phenotype groups ($\sum_{i \in M, j \in K} |[\Theta_{yz}]_{i,j}|$ and $\sum_{i \in M, j \in K} |[B_{yz}]_{i,j}|$ for each gene module $M$ and phenotype group $K$). Similarly, from $\Theta_{xy}$ and $B_{xy}$ we computed the overall SNP effect sizes on all genes in the given module ($\sum_{i \in M} |[\Theta_{xy}]_{i,j}|$ and $\sum_{i \in M} |[B_{xy}]_{i,j}|$ for each SNP $i$ and module $M$).

Among the 20 gene modules, modules 13-20 overall had stronger influence on both lung and blood phenotypes than the other gene modules (Figures 19A and 19B), although SNP perturbations were found across all gene modules without any preference to those modules with stronger influence on phenotypes (Figure 19C). For modules 13-20, the overall effect sizes on the lung phenotypes ranged between 0.8 and
7.5 for direct and indirect influence with an exception of module 14, whereas for modules 1-12, the overall effect sizes were less than 0.8 (Figure 19A). Modules 13-20 also had strong effects on the blood phenotypes (Figure 19B), although module 13 had substantially stronger effect on the blood phenotypes than on the lung phenotypes. On the other hand, the overall SNP effects were similar across all gene modules for both the direct and indirect SNP effects (Figure 19B). The overall indirect SNP effects were larger for some modules (e.g., module 14), but this was largely because of the substantially stronger edge connectivities in that module, which led to stronger propagation of the direct SNP perturbation effects.

### 4.3.2 Gene modules that influence phenotypes are enriched for immune genes

To determine the functional role of the gene modules, we performed gene ontology (GO) gene set enrichment analysis [9, 28]. For each module, we performed a Fisher’s exact test to find the significantly enriched GO categories in biological processes (p-value < 0.05 after Bonferroni correction for multiple testing), using the GO database with annotations for 21,002 genes.

<table>
<thead>
<tr>
<th>Module</th>
<th>Biological Process Description</th>
<th>Size</th>
<th>Pathway</th>
<th>P-value</th>
<th>Overlap*</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cellular macromolecule metabolic process</td>
<td>314</td>
<td>2.94 × 10^{-12}</td>
<td>159 / 7006</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Cellular nitrogen compound metabolic process</td>
<td>297</td>
<td>1.64 × 10^{-4}</td>
<td>112 / 5164</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Nucleobase-containing compound metabolic process</td>
<td>314</td>
<td>5.62 × 10^{-13}</td>
<td>123 / 4538</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Organelle organization</td>
<td>314</td>
<td>2.02 × 10^{-4}</td>
<td>79 / 3167</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Nucleobase-containing compound metabolic process</td>
<td>314</td>
<td>4.58 × 10^{-4}</td>
<td>106 / 4538</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Unclassified</td>
<td>314</td>
<td>NA</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Cellular localization</td>
<td>314</td>
<td>1.80 × 10^{-4}</td>
<td>60 / 2287</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Cellular metabolic process</td>
<td>314</td>
<td>1.73 × 10^{-4}</td>
<td>178 / 9003</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Cellular metabolic process</td>
<td>314</td>
<td>8.39 × 10^{-6}</td>
<td>185 / 9003</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Macromolecule metabolic process</td>
<td>314</td>
<td>8.73 × 10^{-5}</td>
<td>159 / 7749</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Heterocycle metabolic process</td>
<td>314</td>
<td>7.63 × 10^{-3}</td>
<td>103 / 4715</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Translation</td>
<td>298</td>
<td>1.65 × 10^{-9}</td>
<td>27 / 383</td>
<td></td>
</tr>
<tr>
<td>13*</td>
<td>Immune system process</td>
<td>297</td>
<td>3.66 × 10^{-12}</td>
<td>83 / 2552</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Response to stimulus</td>
<td>297</td>
<td>1.70 × 10^{-9}</td>
<td>162 / 8009</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Response to stress</td>
<td>297</td>
<td>8.43 × 10^{-9}</td>
<td>90 / 3333</td>
<td></td>
</tr>
<tr>
<td>14*</td>
<td>Immune response</td>
<td>314</td>
<td>3.96 × 10^{-38}</td>
<td>106 / 1673</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Leukocyte activation involved in immune response</td>
<td>314</td>
<td>9.04 × 10^{-31}</td>
<td>62 / 607</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Granulocyte activation</td>
<td>314</td>
<td>1.24 × 10^{-26}</td>
<td>53 / 495</td>
<td></td>
</tr>
<tr>
<td>15*</td>
<td>Cell activation in immune response</td>
<td>313</td>
<td>4.95 × 10^{-32}</td>
<td>65 / 611</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Myeloid leukocyte activation</td>
<td>313</td>
<td>5.28 × 10^{-32}</td>
<td>63 / 566</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Immune system process</td>
<td>313</td>
<td>1.01 × 10^{-31}</td>
<td>124 / 2552</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Cellular process</td>
<td>290</td>
<td>3.73 × 10^{-3}</td>
<td>132 / 15013</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Regulation of macromolecule metabolic process</td>
<td>290</td>
<td>1.58 × 10^{-3}</td>
<td>74 / 6142</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Organonitrogen compound metabolic process</td>
<td>282</td>
<td>1.97 × 10^{-2}</td>
<td>60 / 5523</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Cell cycle</td>
<td>285</td>
<td>1.50 × 10^{-8}</td>
<td>37 / 1355</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Cellular component organization or biogenesis</td>
<td>295</td>
<td>4.34 × 10^{-3}</td>
<td>66 / 5525</td>
<td></td>
</tr>
</tbody>
</table>

* The number of genes in the overlap / the total number of genes in the GO category

Among all 20 modules, modules 13-15 had a statistically significant enrichment of GO terms related to immune system function, which also corresponded to the most significant enrichments across all modules (Table 4). Even though modules 16-20 did not have any significant enrichment of asthma-related GO
categories, many of the genes in these modules were connected to genes in modules 13-15 in the posterior gene network \( \Lambda_{y|x,z} \) (Figure 20), and thus this subset of genes in modules 16-20 may be also involved with immune system function. To see if this is indeed the case, we obtained the significantly enriched GO categories in the 374 genes in modules 16-20 that are connected to modules 13-15 in the posterior network \( \Lambda_{y|x,z} \) \((p\text{-value} < 0.05\) after Bonferroni correction\). This set of genes was significantly enriched for several GO categories related to immune system processes, including cellular response to stress \((p\text{-value} = 2.90 \times 10^{-2}\) with overlap of 35 genes out of 1599 genes in the category\), regulation of defense response to virus \((p\text{-value} = 4.36 \times 10^{-2}\) with overlap of 6 genes out of 71 genes in the category\), and regulation of immune effector process \((p\text{-value} = 4.42 \times 10^{-2}\) with overlap of 14 genes out of 409 genes in the category\).

Thus, all of the modules that influence phenotypes, modules 13-20, showed enrichments in immune-related genes, with significant enrichment for modules 13-15 and weaker but still significant enrichment for modules 16-20. Since asthma is an immune disorder, the enrichment of immune-related genes in the trait-perturbing modules provides evidence that these modules are likely to play an important role in asthma patients.
Figure 18: The Perturb-Net model estimated from asthma data. The parameters of the sparse Gaussian chain graph model estimated from the asthma data are shown. (A) Asthma phenotype network $\Lambda_{zz}$. The phenotypes were ordered by hierarchical clustering applied to within each of the two groups of phenotypes, lung function traits (yellow) and blood test traits (purple). (B) Gene network $\Lambda_{yy}$. The gene network is annotated with 20 modules obtained from applying a network clustering algorithm METIS [45] to $\Lambda_{yy}$. (C) The influence of gene expression levels on phenotypes $\Theta_{yz}$. (D) SNP perturbation of gene expression levels $\Theta_{xy}$ for the top 1,000 eQTL hotspots, ordered by genomic location and labeled by chromosomes. In each panel, non-zero elements of the estimated parameters are shown as blue for positive interactions and red for negative interactions.
Figure 19: SNP effects on gene modules and gene-module effects on phenotypes. Given the estimated $\Theta_{yz}$ and inferred $B_{yz}$ for gene-expression effects on phenotypes from the Perturb-Net model, we show the gene-module effects on each group of phenotypes for (A) lung and (B) blood, computed as the sum of absolute effect sizes across all genes within the module and across all phenotypes in the group. (C) Given the estimated $\Theta_{xy}$ and inferred $B_{xy}$ for SNP effects on gene network, we show the SNP effects on each gene module, summarized as the sum of absolute effect sizes across all SNPs and all genes within the module.

Figure 20: Asthma posterior gene network. The posterior gene network $\Lambda_{y|x,z}$ after taking into account the phenotype data.
4.3.3 Local SNP perturbations and direct SNP perturbations have larger eQTL effect sizes

Next, we examined the effect sizes of eQTLs identified by our method as perturbing the gene network. Since the indirect perturbations represent the direct perturbation effects propagating to other parts of the network, the direct SNP perturbation effects in $\Theta_{xy}$ may be stronger than the indirect SNP perturbation effects in $B_{xy}$. We compared the distribution of SNP perturbation effect sizes between $\Theta_{xy}$ and $B_{xy}$. We obtained the distribution of perturbation effect sizes as a histogram of non-zero elements in $\Theta_{xy}$ and in $B_{xy}$, after normalizing the frequencies with the total number of non-zero elements (99,055 in $\Theta_{xy}$ and 326,455 in $B_{xy}$ after thresholding $B_{xy}$ at 0.001 to rule out the perturbations with small effect sizes). The histograms in Fig 21A confirm our hypothesis that overall, direct SNP perturbation effects are stronger than indirect perturbation effects (rank sum test $p$-value = 0 within machine precision; 1,656 and 12,049 elements in $\Theta_{xy}$ with effect sizes greater than 0.1 and 0.05, and 0 and 9 elements in $B_{xy}$ in the corresponding ranges).

Among the direct perturbations in $\Theta_{xy}$, the perturbations with stronger effects may be more likely to be local eQTLs. We compared the distributions of the effect sizes of local and distal eQTLs in $\Theta_{xy}$, where the local eQTLs are defined as eQTLs located within 50kb from the coding regions of perturbed genes, after normalizing the frequencies with the total number of eQTLs in each category (279 for local eQTLs and 98,776 for distal eQTLs). The histograms in Fig 21B show that overall, local eQTLs have larger effect sizes than distal eQTLs (rank sum test $p$-value=2.0930 $\times$ 10$^{-43}$), even though the number of local eQTLs with large effect sizes was smaller than that of distal eQTLs (43 local eQTLs and 163 distal eQTLs with effect size>0.2, 11 local eQTLs and 42 distal eQTLs with effect size>0.4). Overall, our results indicate that strong perturbation effects tend to arise from local eQTLs.

![Figure 21: SNP perturbations of the asthma gene network. (A) Comparison of the distributions of the effect sizes for direct and indirect SNP perturbations of the gene network in the sparse Gaussian chain graph model estimated from the asthma data. Histograms of the effect sizes in the estimated $\Theta_{xy}$ and the inferred $B_{xy}$ are shown after the frequencies were normalized by the total numbers of SNP perturbations in each perturbation type. (B) Comparison of the distributions of local and distal eQTL effect sizes among the direct perturbations in $\Theta_{xy}$. The local eQTLs are defined as 50kb from the coding region of the gene for the corresponding expression trait.](image)

4.3.4 SNPs perturbing asthma phenotypes overlap with SNPs perturbing immune modules

The SNP perturbation of the gene modules above (Figure 19C) may or may not result in a change in phenotypes. To see if the SNPs perturbing each gene module have an impact on the lung and blood phenotypes, we compared the top module-specific eQTLs in $\Theta_{xy}$ with the SNPs with the strongest effects on the lung or blood phenotypes in $B_{xz}$ inferred from our sparse Gaussian chain graph model. The SNPs with the strongest
effects on the lung (or blood) phenotypes were determined based on the sum over the SNP effects on all lung (or blood) phenotypes in $|B_{xz}|$. Similarly, the top module-specific eQTLs were determined based on the sum over the SNP effect sizes on all expression levels in each module in $|\Theta_{xy}|$. We obtained the overlap between the SNPs perturbing the phenotype network and the SNPs perturbing the gene network, considering the top 100 and 200 module-specific eQTLs and top 200 SNPs perturbing each phenotype group (the cutoff for top 200 SNPs shown as the magenta line at SNP effect size 0.01325 for lung traits in Figure 22A and at SNP effect size 0.00375 for blood traits in Figure 22B). Using Fisher’s exact test, we also assessed the significance of these overlaps within the set of SNPs with non-zero effects in $\Theta_{xy}$.

In our comparison, only a subset of the eQTLs influenced phenotypes, but the eQTLs perturbing the immune modules, modules 13-20, were more likely to perturb the phenotypes than the eQTLs for the other modules (Figures 23A and 23B). Among the top 100 module-specific eQTLs, only a fraction of those SNPs overlapped with top 200 SNPs perturbing phenotypes (ranging from 0% to 18% of eQTLs across modules for an overlap with SNPs perturbing lung phenotypes and ranging from 2% to 20% of eQTLs across modules for an overlap with SNPs perturbing blood phenotypes). These fractions increased as we considered more eQTLs as in top 200 and all module-specific eQTLs. This matches with the observations from previous
Figure 23: Overlap between SNPs perturbing phenotype network and SNPs perturbing gene network. For each gene module and each group of lung and blood phenotypes, we found the overlap between the top 200 SNPs perturbing the phenotype subnetwork and each of the top 100, 200, and all eQTLs perturbing the gene module. The number of SNPs in the overlap is shown for (A) lung phenotypes and (B) blood phenotypes. Statistical significance of the overlap is shown for (C) lung phenotypes and (D) blood phenotypes.
studies that not all of the eQTLs affect higher-level phenotypes [65] and that trait-associated SNPs are likely to be eQTLs [60]. However, in our analysis, the eQTLs for immune-related modules, modules 13-20, tended to have larger overlaps than the other modules (Figures 23A and 23B). Furthermore, we found these overlaps are statistically significant for all of the immune modules, modules 13-20, but not for all of the other modules, and the most statistically significant overlaps were from the immune modules (Figures 23C and 23D). This suggests that eQTLs that perturb the modules that influence phenotypes are more likely to perturb phenotypes than eQTLs that perturb other gene modules.

4.3.5 The immune modules mediate SNP perturbation of phenotypes

To understand the molecular mechanisms that underlie the SNPs perturbing phenotypes beyond the simple overlap of SNPs perturbing the phenotype network and SNPs perturbing the gene network, we used the Perturb-Net inference procedure to obtain the decomposition of the SNP effects on phenotypes $B_{xz}$ into the component SNP effects on phenotypes $B_{xz}^{M_1}, \ldots, B_{xz}^{M_{20}}$ mediated by each of the 20 gene modules. We examined this decomposition for the 50 SNPs with the strongest effects on each group of lung and blood phenotypes (the cutoff for top 50 SNPs is shown as the green line at SNP effect size 0.04 for the lung phenotypes in Figure 22A and at SNP effect size 0.011 on blood phenotypes in Figure 22B).

For each set of 50 SNPs with the strongest perturbation effects on lung or blood phenotypes, nearly all of their effects on phenotypes were mediated by modules 12 through 20. The decomposition of the SNP effects on lung phenotypes (Figure 24A) into the 20 components (Figure 24B) shows that only the components for modules 12 through 20 contain non-zero SNP effects on the lung phenotypes, except for module 6, which mediates the effects of SNP rs1008932. We further summarized the component SNP effects by summing across all lung phenotypes for each SNP ($\sum_{j \in \text{Lung}} |B_{xz}^{M_i}|, j$ for module $M$ and SNP $i$; Figure 24C). In Figure 24C, for 45 out of the 50 SNPs the SNP effect on the lung phenotypes is mediated by a single module from modules 12-20. For the other 5 SNPs, although their effects on phenotypes were mediated by two or three modules, the module with the strongest mediator effect had effect size at least 5 times as large as the other modules. Although only 20 SNPs overlapped between the two sets of top 50 SNPs for lung and blood phenotypes, the SNP effects on blood phenotypes were also mediated by modules 12-20 (Figure 25). This indicates that modules 12-20 can potentially explain the molecular mechanisms behind the SNP perturbations of asthma phenotypes.

4.3.6 Module 13 explains the molecular mechanism of the previously known association between SNP rs63340 and asthma susceptibility

We performed an in-depth analysis of module 13, its influence on asthma phenotypes, and its perturbation by SNP rs63340, one of the SNPs with the strongest effects on this module and also on phenotypes (Figure 26). SNP rs63340 ranked third for its effect on module 13 and 41st for its effect on phenotypes. The genome region 16q21, where SNP rs63340 is located, has been previously found to be linked to asthma and atopy in several previous genome-wide screenings [89, 61, 36, 50, 2], though the mechanism behind this association has not been fully elucidated. Our model indicated that this locus directly perturbs the expression levels of NRP1, DCANP1, EPHB1, NLRP7 and GZMB. Several of these genes have been previously linked to asthma. NRP1 is known to be a part of one of important mediators involved in the pathogenesis of asthma [77]. A promoter nucleotide variant in DCANP1 was previously associated with serum IgE levels among asthmatics [46]. EPHB1 has been previously linked to lung function traits in asthma [54]. Our model found EPHA2, KCNA5, NRP1, and CLEC4C as key mediator genes, determined by the row of $B_{xz}^{m}$ for SNP rs63340 and for gene $m$ in module 13 summed across all phenotypes, that mediate the effects of SNP rs63340 on asthma phenotypes. Among these genes, KCNA5 has been known to be connected to pulmonary vasoconstriction [66, 7] and a SNP near KCNA5 was significantly associated with asthma [82]. In addition,
Figure 24: Top 50 SNPs perturbing lung phenotypes and their perturbations effects on phenotypes mediated by gene modules. For the top 50 SNPs perturbing lung phenotypes, we show (A) their effect sizes on phenotypes $B_{xz}$ and (B) the decomposition of $B_{xz}$ into component effects $B_{xz}^{M1}, \ldots, B_{xz}^{M20}$ mediated by each of the 20 gene modules. The sum over all component effects in Panel (B) is equal to the overall effects in Panel (A). (C) We summarize each component SNP effect $B_{xz}^{m}$ for module $m$ in Panel (B) as a row-wise sum of $B_{xz}^{m}$, shown as the $m$th column in the figure. The SNPs are ordered according to their overall effect sizes on the lung phenotypes.
Figure 25: Top 50 SNPs perturbing blood phenotypes and their perturbations effects on phenotypes mediated by gene modules. For the top 50 SNPs perturbing blood phenotypes, we show (A) their effect sizes on phenotypes $B_{xz}$ and (B) the decomposition of $B_{xz}$ into component effects $B_{m_{xz}}^1, \ldots, B_{m_{xz}}^{20}$ mediated by each of the 20 gene modules. The sum over all component effects in Panel (B) is equal to the overall effects in Panel (A). (C) We summarize each component SNP effect $B_{m_{xz}}$ for module $m$ in Panel (B) as a row-wise sum of $B_{m_{xz}}$, shown as the $m$th column in the figure. The SNPs are ordered according to their overall effect sizes on blood phenotypes.
Figure 26: The gene network for module 13, its influence on asthma phenotypes, and its perturbation by SNP rs63340. The asthma phenotype network $\Lambda_{zz}$ in our estimated model is shown in the green box and the gene network $\Lambda_{yy}$ for module 13 is shown outside of the green box. The edges across the two networks correspond to direct influence of expression levels on phenotypes $\Theta_{yz}$. The five genes ($NRP1$, $DCANP1$, $EPHB1$, $NLRP7$, and $GZMB$) whose expression is directly perturbed by SNP rs63340 with effect size $>0.05$ in $\Theta_{xy}$ are labeled with arrows, colored red to indicate positive eQTL effects. Node colors depict the indirect effects of this eQTL on gene expression levels $B_{xy}$ and phenotypes $B_{xz}$, with red for up-regulation and blue for down-regulation. Node size of genes depicts the component of the eQTL effects on phenotypes mediated by the given gene, the row of $B_{xz}^m$ for SNP rs63340 and for gene $m$ in module 13 summed across all phenotypes.
CLEC4C has been known to be involved in immune response \cite{13, 69}. Thus, the results from Perturb-Net are well supported by the previous findings in the literature and provide insights into the gene network underlying the previously reported association between the locus and asthma phenotypes.

### 4.4 Comparison with other methods

<table>
<thead>
<tr>
<th>Prediction task</th>
<th>Lasso</th>
<th>Our Model</th>
<th>Our Model with Semi-supervised Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y</td>
<td>x$</td>
<td>0.76494</td>
<td>0.75322</td>
</tr>
<tr>
<td>$z</td>
<td>y$</td>
<td>1.03486</td>
<td>0.97068</td>
</tr>
<tr>
<td>$y</td>
<td>x, z$</td>
<td>0.78161</td>
<td>0.75346</td>
</tr>
<tr>
<td>$z</td>
<td>x$</td>
<td>0.85785</td>
<td>0.85795</td>
</tr>
</tbody>
</table>

We compared our method with the two-layer Lasso both qualitatively by visual inspection of the estimated parameters and quantitatively by assessing the predictive power of different methods.

#### Comparison of the estimated models

We compared the results from our approach and the two-layer Lasso by visually inspecting the estimated SNP effects on gene modules and the estimated gene module effects on phenotypes. For the top 50 SNPs perturbing the lung phenotypes (Figure 24), we examined the overall SNP effects on each gene module based on $\Theta_{xy}$ and $B_{xy}$ from our model and $A_{xy}$ from the two-layer Lasso ($\sum_{j \in M} [\Theta_{xy}]_{i,j}, \sum_{j \in M} [B_{xy}]_{i,j}$, and $\sum_{j \in M} [A_{xy}]_{i,j}$ for SNP $i$ and module $M$). To see how each gene module influences phenotypes, we computed the magnitudes of overall gene module effects on each phenotype from $\Theta_{yz}$ and $B_{yz}$ in our model and $A_{yz}$ in the two-layer Lasso ($\sum_{j \in M} [\Theta_{yz}]_{j,k}, \sum_{j \in M} [B_{yz}]_{j,k}$, and $\sum_{j \in M} [A_{yz}]_{j,k}$ for module $M$ and phenotype $k$).

Unlike the Perturb-Net model, the two-layer Lasso does not model direct and indirect perturbation effects separately but attempts to capture both types of effects in a single set of parameters. Thus, the perturbation effects captured by the two-layer Lasso appeared to be a compromise between the direct and indirect perturbation effects captured by Perturb-Net (Figure 27). However, the SNP effects appeared to be similar across $\Theta_{xy}$, $B_{xy}$, and $A_{xy}$ in the module-level summaries (Figures 27A-27C), because the direct SNP perturbation effects tended to propagate to other genes only within each module, but not to genes in other modules. On the other hand, the module effects on phenotypes showed a distinct pattern across $\Theta_{yz}$, $B_{yz}$, and $A_{yz}$ (Figures 27D-27F), because in our model, the direct influence of gene expression levels on a phenotype induces the indirect influence on other correlated phenotypes, whereas the Lasso parameter tries to capture both types of information in a single parameter.

#### Comparison of prediction accuracy

We assess the ability to make predictions about new asthma patients based on the estimated Perturb-Net model and compare the results with those from the two-layer Lasso. We split the data into train and test sets and obtained the prediction accuracy using the test set after training a model on the train set. We set aside 25 samples as a test set and used the remaining 115 fully-observed samples and 34 partially observed samples to train a sparse Gaussian chain graph model with our semi-supervised learning method. We also trained a model, using only the 115 fully observed samples with the supervised learning method, and compared the results from the two-layer Lasso, also trained from the fully observed samples. Given the estimated models, we performed prediction tasks and obtained the prediction error as the squared difference between the observed and predicted values averaged across samples in test set.

The Perturb-Net model estimated from all data had the smallest prediction error for all of the prediction tasks (Table 5). In particular, our model with semi-supervised learning performed better than our model...
with supervised learning, demonstrating that leveraging partially observed data can help learn a model with greater predictive power. For supervised learning, our model outperformed Lasso. This demonstrates that taking into account the network structure in expression levels and clinical phenotypes increases the performance on prediction tasks.

4.5 Discussion of Perturb-Net for Asthma

We introduced a statistical framework called Perturb-Net for learning a gene network underlying phenotypes using SNP perturbations and for identifying SNPs that perturb this network, given population genotype, expression, and phenotype data. Compared to many of the previous methods that focused on the co-localization of eQTLs and genetic association signals for phenotypes [29, 38, 26], using multi-stage methods [21, 72, 73], our approach combines all available data in a single statistical analysis and directly models the multiple layers of a biological system with a cascade of influence from SNPs to expression levels to phenotypes, while modeling each layer as a network. Our probabilistic graphical model framework allows to model eQTLs with or without an impact on phenotypes for an investigation of co-localization of SNPs perturbing expression levels and SNPs perturbing phenotypes and to extract rich information on the molecular mechanisms that explains the influence of SNPs on phenotypes. We developed fast learning algorithms called Fast-sCGGM and Mega-sCGGM for learning sparse CGGM components of the Perturb-Net model, which serve as the key subroutine of our Perturb-Net learning method, to enable analysis of human genome scale data within a few hours.

Our results from applying Perturb-Net to asthma data confirmed the observations from the previous studies, including GWAS, eQTL mapping, and gene network modeling [72, 60]. Our results confirmed the finding from previous studies on combining the results of GWAS and eQTL mapping [60] that there is a partial overlap between SNPs perturbing expression levels and SNPs perturbing phenotypes. In addition, this overlap was more significant for eQTLs that perturb trait-associated modules than eQTLs that perturb other parts of the gene network, as was previously reported [65].

The analysis of the asthma data with Perturb-Net provided new insights. Perturb-Net was able to systematically reveal the gene network that lies between the SNPs and phenotypes and to uncover how different parts of this gene network modulate the SNP effects on phenotypes in a statistically principled manner. Often, there are genetic loci that have been previously known to be linked to the disease susceptibility, though little is known about the underlying molecular mechanism. In such cases, the Perturb-Net analysis of asthma data demonstrated the potential to reveal the molecular pathway that are perturbed by previously known trait-associated loci.

Perturb-Net provides a flexible tool that can be extended in several different ways in a straightforward manner. Because the sparse Gaussian chain graph model in Perturb-Net uses sparse CGGMs as building blocks, the sparse CGGM component models can be threaded in different ways to form sparse Gaussian chain graph models with different structures. For example, if expression data from multiple tissue types are available for a patient cohort along with genome sequence and phenotype data, a sparse Gaussian chain graph model can be set up with multiple component sparse CGGMs, each corresponding to the gene network under SNP perturbation in each tissue type, linked to another sparse CGGM for modeling expression levels influencing phenotypes. Models like this can reveal SNPs that perturb phenotypes through different tissue types and through different modules in each tissue-specific gene network. Another possible extension is to thread more than two component sparse CGGMs within a sparse Gaussian chain graph model to model more than two layers in a biological system, including epigenomes, metabolomes, and proteomes.
Figure 27: Comparison of different methods for learning the cascaded influence of SNPs to gene modules to phenotypes. For the top 50 SNPs perturbing lung phenotypes (Figure 24), the effects of these SNPs on each of the gene modules are shown for (A) $\Theta_{yz}$ from our model, (B) $B_{yz}$ inferred from our model, and (C) $A_{yz}$ from the two-layer Lasso. The effects of the expression levels in each gene module on lung phenotypes are shown for (D) $\Theta_{yz}$ from our model, (E) $B_{yz}$ inferred from our model, and (F) $A_{yz}$ from the two-layer Lasso. The effect sizes in each model parameter matrix above were summed across all genes within each module after taking absolute values.
References


