Learning Graphical Models Using Junction Trees

Yiqian Gan, Vyas Ramasubramani, Akash Rastogi, Yutong Wang
EECS 545
University of Michigan
Ann Arbor, MI 48104

Abstract

In this project we consider a paper by Vats and Nowak [1] describing a framework for undirected graphical model selection (UGMS). This framework utilizes junction trees as a method to decompose the UGMS problem into smaller subproblems that can be solved efficiently using different UGMS algorithms. We evaluated this method by comparing the results of applying this method to a UGMS problem to applying a standalone UGMS algorithm directly. We tested all of these algorithms on both real and synthetic data. We find that while the junction tree algorithm provides significant gains in some cases, in others these gains are minimal and are offset by much higher computational costs.

1 Motivation

1.1 Probabilistic graphical models

A graph is a mathematical object that encodes relationships between a discrete set of objects. The objects are encoded as vertices, and the related pairs of vertices are called edges. Many probabilistic data sets can be represented graphically, with vertices corresponding to different features and edges encoding conditional independence relationships between the features. The edges of a graph can be directed, meaning that the order of the vertices matters. When the underlying graph of a probabilistic model is undirected, the result is called an undirected graphical model (UGM), also known as a Markov Random Field (MRF). MRFs have been applied in many fields, including computational biology [2], medical imaging [3], remote sensing [4], and information retrieval [5], to name a few. The standard problem in this field is the undirected graphical model selection (UGMS) problem, which is to estimate an undirected graphical model given some observations from the underlying data. A number of methods exist to solve the UGMS problem such as Graphical lasso (Glasso) and the PC algorithm, which are described in detail in section 3.

1.2 Gaussian markov random fields

The theory of *Gaussian* Markov random fields (GMRF), where each of the features is normally distributed, is well developed. GMRFs have numerous applications including time-series analysis and spatial statistics [6]. The theme behind many of these applications is that the relationships between features in high dimensional spaces often encode important structural information, and learning these structures can improve model and prediction accuracy. When the data is assumed to be Gaussian, we can use maximum likelihood estimation to solve the UGMS problem. Furthermore, if the underlying Markov graph is sparse, then we can consider the penalized the log likelihood by the lasso regularizer, often called "graphical lasso". In Friedman et al. [7], a fast block coordinate descent algorithm is given to optimize the lasso penalized log-likelihood function. We review this in appendix A.

1.3 Junction tree framework

UGMS algorithms like graphical lasso suffer from certain drawbacks. Note that each of these subproblems can be solved by any UGMS algorithm, including the graphical lasso. Thus, the junction tree framework provides a meta-method that can be used to augment the performance of UGMS algorithms. In this paper, we evaluate the efficacy of this approach as compared to just using standalone methods such as the graphical lasso to determine the advantages and disadvantages of each of these methods for solving the UGMS problem.

2 Problem statement

Consider the tuple of random variables $\mathbf{X} = (X^{(1)}, \ldots, X^{(p)})$ and a graph $G = (V, E)$ where the vertices $V = \{1, \ldots, p\}$ index the variables in $\mathbf{X}$. We say that $\mathbf{X}$ is an MRF on G if for all $i, j \in V$ such that i and j are not connected by an edge, we have $X^{(i)}$ is conditionally independent of $X^{(j)}$ given $X^{(k)}\ \forall k \in V \setminus \{i, j\}$. In this case, we say that G is the *Markov graph* of $\mathbf{X}$. Now, given realizations $\mathbf{x}_1, \ldots, \mathbf{x}_n$ of $\mathbf{X}$, the *model selection* problem is to estimate the Markov graph G based on these realizations.

3 UGMS methods

For this paper we consider two methods for the solution of the solution of the undirected graphical model selection problem. The graphical lasso problem, as described above, provides one method. For comparison we also used the PC algorithm, which provides an alternative methods to solve the same problem. It was also used by Vats and Nowak [1] in their paper.

3.1 Graphical lasso

We first introduce the basics of the graphical lasso. Consider a Gaussian random variable $\mathbf{X} = (X^{(1)}, \ldots, X^{(p)})$ with zero mean and covariance matrix Σ and let the precision matrix $\Theta = \Sigma^{-1}$. A crucial result in the field of MRFs is that the entries of the precision matrix encode conditional independence relationships between variables. Formally, for i, j distinct, we have that $X^{(i)}$ is conditionally independent of $X^{(j)}$ given $X^{(k)}\ \forall k \in V \setminus \{i, j\}$ if and only if $\Theta_{ij} = 0$ (cf. Speed and Kiiveri [8]). The edges of the Markov graph encode conditional independence relationships: thus, sparsity of Θ is equivalent to sparsity of the Markov graph G. This fact motivates using the lasso penalty for maximum likelihood estimation approach to learning Gaussian MRF. Below is the lasso penalized log likelihood function where S is the data covariance matrix.

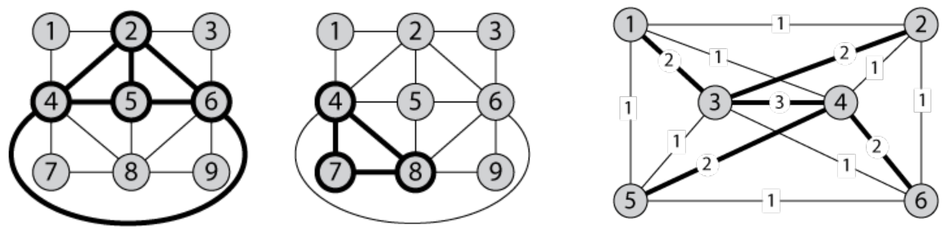
$$\log \det \Theta - \operatorname{Tr}(\Theta S) - \rho \|\Theta\|_1 \tag{1}$$

Here, $\Theta = \Sigma^{-1}$ is the *inverse* of the estimated covariance matrix. Without the penalty, the maximum achieving Θ is simply S^{-1}. The optimization technique used to solve this often termed *block coordinate descent*. The details are discussed in appendix A following the development in Friedman et al. [7].

3.2 PC algorithm

The PC algorithm provides another method to estimate graphical models. Unlike graphical lasso, it can also be used for directed models, but we do not take advantage of that in this paper ([9], [10]). For this paper, only part of the PC algorithm is used for the junction tree model of the undirected graph. The correlation of X and Y conditional upon Z is

$$\rho_{XY \cdot Z} = (1 - \rho_{XZ}^2)^{-1/2} (1 - \rho_{ZY}^2)^{-1/2} (\rho_{XY} - \rho_{XZ}\rho_{ZY}), \tag{2}$$



(a) Clique of size 4 (b) Clique of size 3 (c) Maximal spanning tree example

Figure 1: Some basic graphical concepts

where ρ_{XY} is the correlation of X and Y. The core idea is that conditional correlation coefficient($\rho_{XY \cdot Z}$) is zero if X and Y are conditionally independent($X \perp Y|Z$). Using the same idea, one can determine the edges of an undirected graphical model by applying a threshold λ_n to the conditional correlation of each each edge based on the related observations, i.e., $\hat{\rho}_{ij|S} < \lambda_n \implies X_i \perp X_j|X_S$. As described in algorithm 4 in appendix B, Vats and Nowak [1] use the PC parameter κ to lower computational complexity by constraining the size of the resulting region graph.

4 Junction tree framework

Given UGMS methods like graphical lasso or PC, the core of the junction tree framework is an iterative algorithm used to repeatedly decompose the UGMS problem into smaller subproblems that are then solved by the provided UGMS method. A critical feature of the junction tree algorithm is that it requires as input a graph H that is a supergraph of the true graph G, i.e. given data that is distributed according to a MRF $G(V, E)$, the junction tree framework assumes that a method exists to estimate a graph $H(V, \bar{E})$ where $E \subseteq \bar{E}$. Obtaining the graph H is a non-trivial problem. It can come from either prior knowledge or conditional independence testing, which is beyond the scope of this project. Our focus, following Vats and Nowak [1], is to assume that H is given, then investigate how the graphical structure of H can be exploited to help us produce more accurate estimates.

4.1 Graph theory background

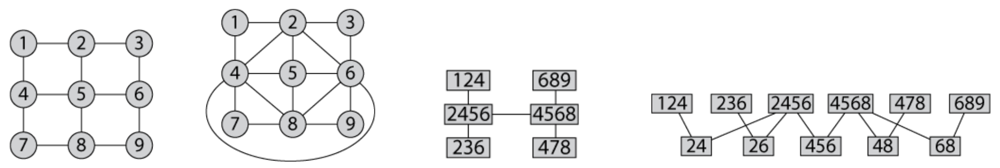
Before delving too deeply into the junction tree method process, we first introduce some fundamental graph theoretic notions. For an in-depth treatment of graph theory, see [11].

Below, let $G(V, E)$ be a graph. A *path* on G is an alternating sequence of distinct edges and vertices. A graph is *connected* if there is a path between every pair of vertices. A graph is *complete* if there is an edge between every pair of vertices. A *cycle* is a sequence of edges and vertices that are all distinct except that the two endpoints vertices are the same. An *acyclic* graph contains no cycles, otherwise the graph is said to be *cyclic*.

A *tree* is a graph that is acyclic and connected. If G is a tree and v_1, v_2, v_3 are vertices such that every path from v_1 to v_3 passes through v_2, then we say that v_2 *separates* v_1 and v_2. A *spanning tree* of a graph G is a subgraph of G that is a tree containing all vertices of G.

An *edge-weighted* graph, or simply a *weighted graph*, is a graph $G(V, E, W)$ where a weight $w \in W$ is associated with every edge $e \in E$. A *minimal (resp. maximal) weight spanning tree* of a weighted graph G is a spanning tree with minimum (resp. maximal) weight, where the *weight of the tree* is determined by the sum of the weights of its edges. Figure 1c shows a maximal spanning tree of weight 11.

A *junction tree* a graph $G(V, E)$ is a tree J where each vertex $j \in J$ is a subset of V, and the following properties hold: (1) Every $v \in V$ is in some vertex $j \in J$; (2) For each $e \in E$, there is a vertex $j \in J$ that contains both of the endpoints of e; and (3) For $j_1, j_2, j_3 \in J$, if j_2 separates j_1 and j_3, then $j_1 \cap j_3 \subset j_2$ (the running intersection property)



(a) H graph (b) H triangulated (c) Junction Tree (d) Region Graph.

Figure 2: In the region graph, the layer number increases further down in the graph.

4.2 Constructing the junction tree

With the above definitions, we are now in a position to discuss the core results of Vats and Nowak [1]. We assume that the supergraph H has already estimated and is ready to be used to generate a junction tree. Constructing the junction tree is a non-trivial task. Below, we will give an overview of this process. For more in-depth treatment, we refer the reader to [12].

For illustrative purposes, we will use the graph in figure 2a as our H graph. The first step toward constructing the junction tree of H is to *triangulate* H. A graph is called a *triangulated* (or chordal) graph if every cycle consisting of at least 4 nodes contains a *chord*, which is an edge that is not part of the cycle. Given a graph H, a triangulation of H is a supergraph of H that is a triangulated graph. For example, figure 2b is the triangulation of figure 2a. In our project, we implemented the `GreedyDegree` algorithm for triangulation. See Bodlaender and Koster [12] for details.

After triangulating H, our next task is to find the set of *maximal cliques* of the graph. A *clique* C of a graph H is a complete subgraph of H. Every vertex is trivially a clique, as is every pair of vertices that is containing. For instance, see figures 1a and 1b. A clique C is called *maximal* if it is not a subgraph of another clique. One can see by inspection that both of the two example cliques are maximal, since adding any other vertex to these cliques will result in a set of vertices where at least one pair in the set is not connected by an edge.

To find all maximal cliques, we use the Bron-Kerbosch algorithm with partial pivoting and a vertex ordering. The central idea of the algorithm is to recursively grow cliques and determine whether they are maximal at each step. For details on this algorithm, we refer the reader to Eppstein et al. [13].

Now, let $\mathcal{C}$ be the set of all maximal cliques in H. The goal is to find a tree decomposition of the graph H. We now define the *intersection graph* of any set $\mathcal{C}$, which is the graph formed by first taking each set $S \in \mathcal{C}$ as a vertex, then adding an edge between every pair of sets with a nonempty intersection. Let $\mathcal{C}$ be the set of all maximal cliques in a graph. The *clique graph* of H is the intersection graph of $\mathcal{C}$. If we take the clique graph and weight each of the edges (S_1, S_2) by the cardinality of the intersection $S_1 \cap S_2$, the resulting graph is known as the *junction graph* of H.

Once we have the junction graph of H, we are nearly done. The final step in constructing junction trees uses the following fact: a maximal spanning tree of the junction graph is a junction tree for H. One can easily confirm that the junction tree of the graph H in figure 1b is isomorphic to the minimal spanning tree in figure 1c. To find a maximal spanning tree, we implemented Kruskal's algorithm, a standard algorithm for determining a maximal spanning tree on a graph. For our running example, we check that figure 2c is a maximal spanning tree. This concludes the construction of a junction tree given a graph H.

4.3 The region graph of a junction tree

The next major development in this paper is their definition of a region graph. Here, we will describe the region graphs. The full pseudocode is included in algorithm 5 in appendix C.

The region graph is split into different layers with the top layers R^1 just being the clusters of the junction tree. For example, consider figure 2c and figure 5. Then we take the intersections of clusters that are adjacent in the junction to form the second layer R^2. If a region in R^1 contains the region in R^2, then there is an directed edge point from R^1 to R^2 between the two regions. For R^ℓ where $\ell > 2$, suppose that $R^{\ell-1}$ has been constructed. We take pairwise intersection of sets in the previous layer and again connect regions in $R^{\ell-1}$ that contains the region in R^ℓ with an edge. We say that $R^{\ell-1}$ is the *parent* layer of R^ℓ and R^ℓ is the *child* layer of $R^{\ell-1}$.

Note that the edges in the region graph are directed, pointing from lower to higher layers. We define the children of a region $\mathrm{ch}(R)$ as the set $\{S : (R, S) \in E(J)\}$. We then call R an ancestor of S. We extend this definition of ancestor to consider all ancestors of R as ancestors of S as well, making the ancestral relationship recursive. $\overline{R}$ is the union of R and its ancestors. $H[R]$ is induced graph of R containing all the edges in H over R. The graph $H'(R)$ is then defined as set difference of $H[R]$ and the complete graph over all vertices contained in the children of R. We then use $H'(R)$ to ensure that the UGMS algorithm is only applied once to every edge in H.

This region graph construct is the central structure in the junction tree framework. The region graph provides a natural way to build up the graph incrementally by estimating cliques, then estimating the edges connecting them, and slowly moving up a hierarchy to estimate more and more of the graph. The crux of the junction tree framework is to select some algorithm that solves the UGMS problem, then iteratively construct a region graph for the graph using algorithm 5, estimate the regions using the UGMS algorithm as shown in 6, then reconstruct the region graph and repeat until some termination criterion is reached. Algorithm 6 applies the picked UGMS to "unique edges"($H'(R)$) of region R using observations of all its ancestors($\overline{R}$).

Algorithm 1

```
1: procedure JUNCTION TREE FRAMEWORK FOR UGMS($RG$)
2:     Initialize $\widehat{G}$ so that $E(\widehat{G}) = \emptyset$ and find the region graph $\mathfrak{G}$ of $H$ using algorithm 5.
3:     Find the smallest $l$ such that there exists a region $R \in R^l$ such that $E(H'_R) \neq \emptyset$.
4:     Apply Algorithm 6 to each region in $R^l$
5:     Add all estimated edges to $\widehat{G}$ and remove edges from $H$ that have been estimated.
       Now $H \cup \widehat{G}$ contains all the edges in $G^*$.
6:     Compute a new junction tree and region graph $G$ using the graph $\widehat{G} \cup H$.
7:     If $E(H) = \emptyset$, stop the algorithm, else go to Step 2.
8: end procedure
```

5 Analysis

5.1 Metrics

To evaluate performance, several metric are defined in [1]: False Discovery Rate (FDR), True Positive Rate (TPR), True Discovery Rate (TDR), Edit Distance (ED). We compute these metrics in a number of cases.

$$FDR = \frac{\text{\# of edges in } \widehat{G} \setminus G^*}{\text{\# of edges in } \widehat{G}}, \quad TPR = \frac{\text{\# of edges in } \widehat{G} \cap G^*}{\text{\# of edges in } G^*}, \quad TDR = (1 - FDR) \times TPR,$$

$$ED = (\text{\# of edges in } \widehat{G} \setminus G^*) + (\text{\# of edges in } G^* \setminus \widehat{G})$$

5.2 Synthetic data

For synthetic data, we consider three classes of graphs G: cycles, chains, and grids. A *cycle graph* is a graph that is just a single loop. A *chain graph* is a graph that is a single path. A *grid graph* is a square lattice where each vertex is connected to four neighbors, except the vertices at the boundary.

Our feature dimension is $|G| = 25$. We then let the precision matrix $\Theta = G + nI$ where I is the identity and n is the smallest integer such that Θ is positive definite. Next, we randomly generate $\mathbf{x}_i$ for $i = 1, \ldots, 200$ according to a zero mean multivariate Gaussian with covariance matrix Θ^{-1}.

We then added noise randomly to G (while keeping a certain sparsity of the precision matrix) to obtain the H graph as the input to the junction tree UGMS algorithm, whose goal is to recover G from the input H and the data matrix $\mathbf{X}$. The results are shown in Tables 1 and 2. We compared the standalone graphical lasso (glasso) algorithm with the graphical lasso within the junction tree framework for these three synthetic data sets above using two fine-tuned regularization parameters, the regularization parameter ρ and a threshold. For the PC algorithm and junction tree - PC, thresholding is kept as 0.2 for all cases (no regularization is needed in PC) as recommended in Vats and Nowak [1].

Table 1: Results for JT Glasso and glasso algorithm with p =25, n = 100

		FDR	TPR	ED	TDR	ρ	Threshold
JT Glasso	Cycle graph	0.6563	0.2933	47.5	0.1008	0.0156	0.1
	Grid graph	0.1707	0.85	13	0.7049	0.0313	0.005
	Chain graph	0.6136	0.2297	42	0.0888	0.0156	0.2
Glasso	Cycle graph	0.8333	0.1333	57.5	0.0222	0.0313	0.2
	Grid graph	0.6563	0.825	70	0.2836	0.0156	0.2
	Chain graph	0.8108	0.1892	60	0.0358	0.0313	0.2

As we can see in the table above, the junction tree - glasso algorithm performs better than the glasso algorithm in all four performance metrics, especially the FDR metric. Junction tree - PC is almost the same as PC algorithm with current thresholding. Among the three categories of the synthetic data, we find that the best results are for the grid graph, followed by chain and cycle graph. This trend seems consistent across all four algorithms (including JT-PC and PC).

Table 2: Results for JT-PC and PC algorithm with p = 25, n = 100

		FDR	TPR	ED	TDR
JT-PC	Cycle graph	0.6111	0.0933	39.5	0.0363
	Grid graph	0.1071	0.625	18	0.558
	Chain graph	0.5	0.0811	37	0.0405
PC	Cycle graph	0.5385	0.16	38.5	0.0738
	Grid graph	0.0345	0.7	13	0.6759
	Chain graph	0.55	0.1216	38	0.0547

5.3 Cross validation

For cross validation, we followed the methods in Vats and Nowak [1] and use the extended Bayesian Information Criterion Chen and Chen [14]. The method for choosing the lasso penalty parameter in (1) is to select ρ to maximize

$$n[\log\det\Theta_\rho - \mathrm{Tr}(S\Theta_\rho)] + |E(\widehat{G}_\rho)|(\log n + 2\log p) \tag{3}$$

where $\widehat{G}_\rho$ and Θ_ρ are the graph and matrix recovered using ρ in graphical lasso. Here, S is the empirical data covariance matrix.

5.4 Real world data

We tested all four algorithms on the ARA dataset from https://github.com/vats-div/JunctionTreeUGMS/tree/master/RealData . Since, the analysis for this data set is not mentioned in Vats and Nowak [1], we thought to extend the framework to this data set. To tune the parameters for real world analysis, we use 5-fold cross validation 5.3.

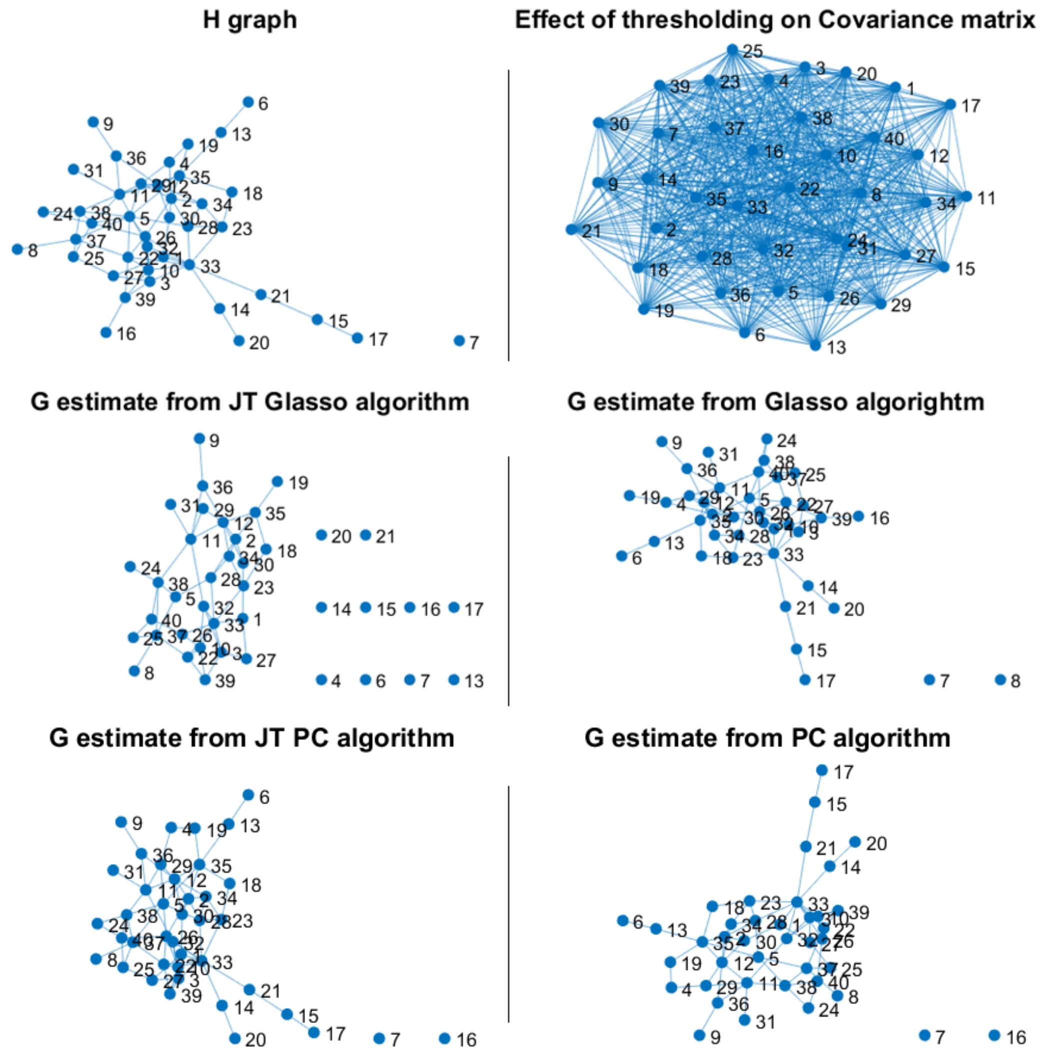

H graph
Effect of thresholding on Covariance matrix
G estimate from JT Glasso algorithm
G estimate from Glasso algorightm
G estimate from JT PC algorithm
G estimate from PC algorithm

Figure 3: H graph and estimated graphs for the ARA data

From the figure above, we can observe that inspite of optimum threshold being low, JT Glasso does a better job at retrieving a more sparse and better estimate for the true graph in comparison to glasso algorithm. In the estimated graph from glasso there are just two nodes which are not connected to any other node whereas in the estimated graph from JT glasso algorithm, there are 10 such nodes. The estimated graphs from PC and JT-PC algorithms shows no such significant difference between the two.

6 Conclusion

From our analysis, we find that the junction tree while the algorithm does offer significant improvements in some cases, it does not offer a universal improvement on other UGMS algorithms. For example, in the three types of synthetic graphs we construct, the junction tree method performs much better on grid graphs than on cycle or chain graphs. This simplistic example clearly suggests that the structure of the graph plays a large role in the effectiveness of the junction tree method. We posit that this is likely due to the treewidth characteristics of specific types of graphs; lower treewidth graphs have smaller cliques, meaning that the tree decomposition is more favorable.

This conclusion is further borne out by the fact that for relatively sparse graphs the method performs at least as well as standard methods while adding little computational overhead. On large, dense graphs, however, the method introduces prohibitive computational overhead in the computation of junction trees and region graphs. This overhead is particularly large when any form of model (parameter) selection is considered. The selection of optimal parameters for the algorithm requires running the entire algorithm many times, and this is especially intractable if parameters like the threshold are set in such a way that the resulting estimated graph is extremely dense. One of the the major advantages mentioned by Vats and Nowak [1] regarding this method is that it allows the use of different regularization parameters for different clusters, using local information to increase the accuracy of prediction. Unfortunately, for insufficiently sparse graphs this method is also prohibitively expensive. Given our resources, parameter estimation was therefore only feasible on the overall algorithm, not on each individual subproblem generated by the junction tree decomposition.

An important point largely glossed over by Vats and Nowak [1] is that the method is heavily dependent on having a good H graph. With no prior information, the natural assumption for the H graph is simply the complete graph on the vertices of G, but using the complete graph typically makes the junction tree approach produce results that are no better than just a standalone UGMS algorithm (due to the fact that the entire graph is one clique, preventing any decomposition). While Vats and Nowak [1] assert that screening methods exist for the estimation of the H graph, our attempts suggest that these are highly case-specific. Therefore, we conclude that the effectiveness of this method is highly dependent on having an H graph that is constructed based on prior data. Having a subject matter expert establish some subset of independence relationships ahead of time would be far more reliable than any screening method, and is also likely to result in a much sparser graph. Returning to the protein expression example, a biologist should be able to generate a reasonable H graph by simply indicating when any pair of proteins should have completely uncorrelated expression patterns. These are the conditions under which the junction tree framework provides substantial improvements.

Despite these issues, we do find that the junction tree framework improves upon standalone graphical lasso estimates in our tests. This suggests that while the junction tree framework could offer substantial improvement in terms of accuracy, the computational load it introduces should be heavily weighed against this improvement. Ultimately, the choice of using this framework should be based on whether or not an accurate and sufficiently sparse H graph can be provided to take advantage of the strengths of the junction tree framework.

7 Group Roles

Vyas wrote the majority of the graphical algorithms, including triangulation, the Bron-Kerbosch algorithm, Kruskal's algorithm, and the region graph algorithm. Additionally, Vyas wrote the majority of the final paper and edited the proposal heavily. Yutong was responsible for project ideation and the majority of the initial literature search, as well as authoring most of the initial proposal. Yutong also wrote the glasso and junction graph algorithms as well as the main script and some cross-validation. Akash wrote the algorithm to optimize over the regions of a region graph as well writing the metrics. Akash was also responsible for writing and running the majority of the data analysis code. Yiqian wrote the PC algorithm and helped write and debug various other parts of our code.

References

[1] Divyanshu Vats and Robert D Nowak. A Junction Tree Framework for Undirected Graphical Model Selection. *Journal of Machine Learning Research*, 15:147191, 2014. ISSN 15337928. URL http://jmlr.org/papers/v15/vats14a.html.

[2] Min Chen, Judy Cho, and Hongyu Zhao. Incorporating biological pathways via a markov random field model in genome-wide association studies. *PLoS Genet*, 7(4): e1001353, 2011.

[3] Y. Zhang, M. Brady, and S. Smith. Segmentation of brain MR images through a hidden Markov random field model and the expectation-maximization algorithm. *IEEE Transactions on Medical Imaging*, 20(1):45–57, 2001. ISSN 02780062. doi: 10.1109/42.906424. URL http://www.scopus.com/inward/record.url?eid=2-s2.0-0034745001{\&}partnerID=tZOtx3y1.

[4] T Kasetkasem, M Arora, and P Varshney. Super-resolution land cover mapping using a Markov random field based approach. *Remote Sensing of Environment*, 96 (3-4):302–314, 2005. ISSN 00344257. doi: 10.1016/j.rse.2005.02.006. URL http://www.sciencedirect.com/science/article/pii/S0034425705000866.

[5] Donald A Metzler. Automatic Feature Selection in the Markov Random Field Model for Information Retrieval. In *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, pages 253–262, 2007. ISBN 978-1-59593-803-9. doi: 10.1145/1321440.1321478. URL http://doi.acm.org/10.1145/1321440.1321478.

[6] Havard Rue. Gaussian Markov Random Fields: Theory and Applications. *Hand The*, 104(1960):263 p., 2005. ISSN 0026-1335. doi: 10.1007/s00184-007-0162-3. URL http://www.amazon.com/dp/1584884320.

[7] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 2008. ISSN 14654644. doi: 10.1093/biostatistics/kxm045.

[8] TP Speed and HT Kiiveri. Gaussian markov distributions over finite graphs. *The Annals of Statistics*, pages 138–150, 1986.

[9] Clark Glymour. An algorithm for fast recovery of sparse causal graphs. 1990.

[10] Adam J Rothman, Peter J Bickel, Elizaveta Levina, Ji Zhu, et al. Sparse permutation invariant covariance estimation. *Electronic Journal of Statistics*, 2:494–515, 2008.

[11] John Adrian Bondy and Uppaluri Siva Ramachandra Murty. *Graph theory with applications*, volume 290. Citeseer, 1976.

[12] Hans L Bodlaender and Arie MCA Koster. Treewidth computations i. upper bounds. *Information and Computation*, 208(3):259–275, 2010.

[13] David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In *International Symposium on Algorithms and Computation*, pages 403–414. Springer, 2010.

[14] Jiahua Chen and Zehua Chen. Extended bayesian information criteria for model selection with large model spaces. *Biometrika*, 95(3):759–771, 2008.

[15] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 2008. ISSN 14654644. doi: 10.1093/biostatistics/kxm045.

[16] Jean Baptiste Pettit, Raju Tomer, Kaia Achim, Sylvia Richardson, Lamiae Azizi, and John Marioni. Identifying Cell Types from Spatially Referenced Single-Cell Expression Datasets. *PLoS Computational Biology*, 10(9), 2014. ISSN 15537358. doi: 10.1371/journal.pcbi.1003824.

Appendix A Block coordinate descent for graphical lasso

We following [7]. Let $x_1, \ldots, x_n \in \mathbb{R}^d$ be the training data with zero mean. Suppose that we know the data come from a Gaussian Markov random field with covariance matrix Σ over a *sparse* graph $G = (V, E)$. The goal is to recover the graph G. Now, let $\Theta = \Sigma^{-1}$ and consider

$$\ell(\Theta) := \log p(x_1, \ldots, x_n \mid \Theta) = \log \prod_{i=1}^{n} \sqrt{\frac{\det \Theta}{(2\pi)^d}} \exp\left(-\frac{x_i^T \Theta x_i}{2}\right) \tag{4}$$

Using the fact that $u^T M u = \mathrm{Tr}(uu^T M)$ when M is square symmetric, we have that

$$\ell(\Theta) \propto \log\det\Theta - \mathrm{Tr}(S\Theta) \tag{5}$$

where $S = \frac{1}{n} x_i x_i^T$ is the data covariance matrix. To ensure a sparse estimate, we use ℓ_1 regularization

$$\max_{\Theta \succeq 0} \log\det\Theta - \mathrm{Tr}(\Theta S) - \rho\|\Theta\|_1 \tag{6}$$

Maximization of (6) is done using **coordinate descent algorithm**. Let $\Theta = W^{-1}$. The subgradient equation is

$$W - S - \rho\Gamma = 0, \quad \text{where } \Gamma = \mathrm{sign}(\Theta) \tag{7}$$

which implies

$$w_{12} - s_{12} - \rho\gamma_{12} = 0, \quad \text{where } \gamma_{12} = \mathrm{sign}(\theta_{12}) \tag{8}$$

Consider a different optimization problem

$$\min_{\beta} \frac{1}{2}\|W_{11}^{1/2}\beta - b\|^2 + \rho\|\beta\|_1, \quad \text{where } b = W_{11}^{-1/2} s_{12} \tag{9}$$

The subgradient equation is

$$W_{11}\beta - s_{12} + \rho\nu = 0, \quad \text{where } \nu = \mathrm{sign}(\beta) \tag{10}$$

Note that if (W, Γ) solves (7), then (8) holds. So if we let $\beta = W_{11}^{-1} w_{12}$ and $\nu = -\gamma_{12}$, then (10) also holds. However, we must check that $\nu = \mathrm{sign}(\beta)$ is not violated. To see this, recall that $W\Theta = I$ which implies that

$$W_{11}\theta_{12} + w_{12}\theta_{22} = 0$$

Multiplying the above equation on both side by W_{11}^{-1} and rearranging, we get

$$\boxed{\theta_{12} = -\theta_{22} W_{11}^{-1} w_{12} = -\theta_{22}\beta} \implies \boxed{\nu = \mathrm{sign}(\beta) = -\mathrm{sign}(\theta_{12}) = -\gamma_{12}}$$

Note that we used the fact that $\theta_{22} \geq 0$ since Θ is PSD. In general, the diagonal entries of a PSD matrix are non-negative. The pseudocode for this block coordinate descent is as follow:

Algorithm 2

```
1: procedure GraphicalLasso($S, \rho$)
2:     $W \leftarrow S + \rho I$
3:     while not converged do
4:         for j = 1,...,d do
5:             $W_{11} \leftarrow$ the submatrix of $W$ by removing the $j$-th column and row from $W$.
6:             $s_{12} \leftarrow$ the $j$-th column of $S$ with the $j$-th row removed.
7:             $\beta \leftarrow$ Solver($W_{11}, s_{12}$) (algorithm 3 below.)
8:             Assign $W_{11}\beta$ to the corresponding row and column of $W$.
9:         end for
10:    end while
11:    return $W$
12: end procedure
```

Algorithm 3

```
1: procedure SOLVER($V, u, \rho$)
2:     Initialize $\beta$
3:     while not converged do
4:         for j = 1,. . . , d do
5:             $\beta_j \leftarrow S(u_j - \sum_{k \neq j} V_{kj}\beta_k, \rho)/V_{jj}$
6:         end for
7:     end while
8: end procedure
```

where $S(x, t) = \text{sign}(x) \max(|x| - t, 0)$.

Appendix B PC algorithm Pseudocode

Algorithm 4

```
1: procedure PC-ALGORITHM FOR UGMS($\kappa, \mathfrak{X}^n, H, L$)
   Inputs :
       $\kappa$: An integer that controls the computational complexity of PC.
       $\mathfrak{X}^n$: $n$ i.i.d. observations.
       $H$: A graph that contains all the true edges $G^*$.
       $L$: A graph that contains the edges that need to be estimated.
   Outputs: A graph $\widehat{G}$ that contains edges in $L$ that are estimated to be in $G^*$.
2:     $\widehat{G} \leftarrow L$
3:     for each $k \in 0, 1, ..., \kappa$ do
4:         for each $(i, j)$ $E(\widehat{G})$ do
5:             $S_{ij} \leftarrow$ Neighbors of $i$ or $j$ in $H$ depending on which one has lower cardinality.
6:             if $\exists S \subset S_{ij}, |S| = k$, s.t. $X_i \perp X_j | X_S$ (computed using $\mathfrak{X}^n$) then
7:                 Delete edge $(i, j)$ from $\widehat{G}$ and $H$.
8:             end if
9:         end for
10:    end for
11:
12:    return $\widehat{G}$.
13: end procedure
```

Appendix C Algorithm for constructing the region graph

Algorithm 5

```
1: procedure CONSTRUCTING REGION GRAPHS($JH(H)$)
2:     Input : A junction tree $J = (C, E(J))$ of a graph $H$.
3:     Output : A region graph $G = (\mathcal{R}, E\ (G))$.
4:     Let $R^1$ be the set $C$. Let $R^2$ be all the separators of $J$ , that is, $R^2 =$
       $S_{uv} = C_u \cap C_v : (C_u, C_v) \in E(J)$.
5:     To construct $R^3$, find all possible pairwise intersections of regions in $R^2$. Add all
       intersecting regions with cardinality greater than one to $R^3$.
6:     Repeat previous step to construct $R^4, ..., R^L$ until there are no more intersecting re-
       gions of cardinality greater than one.
7:     For $R \in R^l$ and $S \in R^{l+1}$, add the edge $(R, S)$ to $E\ (G)$ if $S \subseteq R$.
8:     Let $\mathcal{R} = \{R^1, ..., R^L\}$.
9: end procedure
```

Appendix D Algorithm for performing UGMS in a given region

Algorithm 6

```
1: procedure UGMS OVER REGIONS IN A REGION GRAPH($\mathfrak{G}, R, \mathfrak{X}^n, \Psi$)
2:     Inputs : Region graph $\mathfrak{G} = (R, E(G))$, a region $R$, observations $\mathfrak{X}^n$, and a UGMS
       algorithm $\Psi$.
3:     Compute $H'_R$ and $\overline{R}$
4:     Apply $\Psi$ to $\mathfrak{X}^n_{\overline{R}}$ to estimate edges in $H'_R$.
5:     return the estimated edges $\widehat{E}_R$.
6: end procedure
```