8054 final project: Network Lasso via Splitting Methods

Chen-Wei Hua[*]
School of Statistics, University of Minnesota Twin Cities

May 6, 2025

Abstract

This project explores the use of convex clustering methodology through the Network Lasso framework, aiming to estimate latent positions of nodes within a graph-structured dataset. We implement two splitting-based optimization methods: the Alternating Direction Method of Multipliers (ADMM) and the Alternating Minimization Algorithm (AMA), following formulations in Hallac et al. (2015) and Chi and Lange (2015). Using simulated data generated from a Tau protein model—where brain region interactions are represented by noisy matrices—we evaluate and compare the convergence behavior, computational efficiency, and clustering performance of both methods. Our results demonstrate that AMA achieves faster convergence for small regularization parameters but struggles during the cluster merging phase, while ADMM maintains stability across a wider range of parameters. These findings highlight practical considerations for applying Network Lasso to real-world biomedical data.

Keywords: *Network Lasso, Convex Clustering, Splitting Methods, ADMM, AMA, Graph-structured Data*

1 Introduction

The analysis of graph-structured data has become increasingly important across fields such as bioinformatics, social network analysis, and neuroscience. In many applications, each node in a network is associated with a latent representation that captures hidden traits or functional roles. For instance, in neuroimaging studies, correlations between brain regions can be modeled as networks where nodes represent regions of interest (ROIs) and edges encode pairwise interactions. Estimating latent structures underlying such data can reveal meaningful patterns, such as patient subtypes or functional clusters.

Clustering of high-dimensional data points is a central task in unsupervised learning. Traditional clustering algorithms—including k-means, hierarchical clustering, and spectral clustering—are widely used to discover groups of similar nodes. However, these

[*]hua00076@umn.edu

methods typically require pre-specification of the number of clusters and may not fully exploit the network structure underlying the data. To address these limitations, **convex clustering** frameworks have been developed, which integrate both node-level observations and pairwise similarity information within a unified optimization framework. A key feature of convex clustering is that it generates a continuous clustering path: as the regularization parameter increases, clusters progressively merge, allowing practitioners to explore clustering structures at multiple resolutions without committing to a fixed number of clusters upfront.

Network Lasso can be seen as a generalized version of convex clustering. While convex clustering and the Network Lasso share a common optimization framework that fuses node representations to promote clustering, two key differences distinguish them. First, the Network Lasso allows for general convex loss functions in the fitting term $f_i(u_i)$, enabling flexibility to tailor the model to various types of data. In contrast, convex clustering typically assumes a squared error loss, making it well-suited for standard Euclidean data but less adaptable to other contexts.

Second, the treatment of the graph structure $G = (V, E)$ differs fundamentally between the two approaches. Convex clustering generally does not incorporate prior knowledge of the graph and often defaults to using a fully connected graph, applying fusion penalties across all pairs of nodes. This reflects an implicit assumption that every pair of points may potentially belong to the same cluster. On the other hand, the Network Lasso explicitly incorporates an application-specific graph structure, allowing for sparse or problem-driven connectivity patterns. This flexibility enables modeling of real-world scenarios where only specific pairs of nodes are meaningfully related, and it elevates the sparsity and design of the graph G to a central role in both clustering behavior and computational performance.

Although the optimization problem underlying both convex clustering and the Network Lasso is convex and guarantees a unique global solution, solving it efficiently poses significant computational challenges, especially as the size of the graph and the dimensionality of the data increase. The flexibility of the Network Lasso in allowing arbitrary convex loss functions and sparse graph structures, while advantageous for modeling real-world data, also introduces additional complexity. In particular, sparse graphs can lead to irregular communication patterns between nodes, and non-quadratic losses may complicate the structure of the optimization landscape.

To address these challenges, splitting methods such as the Alternating Direction Method of Multipliers (ADMM) and the Alternating Minimization Algorithm (AMA) have been developed. Both methods decompose the original problem into iterative subproblems to improve computational efficiency, particularly in large-scale or distributed settings. ADMM introduces explicit auxiliary variables and applies an augmented Lagrangian formulation, which incorporates penalty terms in both the primal and dual updates to enforce consistency between variables. AMA, by contrast, maintains auxiliary variables—typically dual variables—associated with edges, and while its auxiliary updates also involve penalty terms, its primal updates are notably simpler: they do not include augmented terms and directly solve the original optimization objective. This results in faster iterations for AMA; however, AMA requires additional conditions, such as appropriate step size selection and certain graph properties, to guarantee convergence. ADMM, although computationally heavier per iteration, offers broader convergence guarantees under more general conditions.

In this report, we apply both ADMM and AMA to solve a Tau protein model—a concrete example of convex clustering that arises in neuroimaging studies. We systematically compare the performance of the two methods in terms of convergence behavior, computational efficiency, and clustering results. Through this study, we aim to provide practical insights into the trade-offs between ADMM and AMA when applied to graph-structured biomedical data.

2 Network Lasso Formulation

We start with formulation of the problem. Given a graph(or network) $G = (V, E)$, where V is the node set (for simplicity, the set $V = [N] = \{1, 2, ..., N\}$) and E is the edge set. Assume that each node i has a latent position $u_i \in \mathbb{R}^d$ to be estimated. Consider the similarity weight w_{ij} defined in the edge set E. We believe that nodes with higher similarity(i.e. have large similarity weights) are likely to have similar or even same latene position u_i, so we penalties on the weighted difference $\gamma w_{ij}\|u_i - u_j\|$. As a result, we are solving the following optimization problem:

$$\min_{\{u_i\}} \sum_{i \in V} f_i(u_i) + \gamma \sum_{(i,j) \in E} w_{ij}\|u_i - u_j\| \tag{1}$$

where f_i is the local fitting loss for node i. (1) is called a network lasso because the second term can be seen as a cost on the edges. Convex clustering is a special case when $f_i(u_i) = \frac{1}{2}\|u_i - x_i\|_2^2$ is a square loss function. Here, x_i may be some noisy observation. The lasso parameter γ defines the trade-off between local fitting and agreeing with neighbors. At $\gamma = 0$, the solution at node i is the minimizer of f_i, in the convex setting it is the observed x_i. As $\gamma \to \infty$, the solution on each node will eventually merge together. Therefore, solutions 0 to ∞ characterize a merging path of nodes. Moreover, the solution in the merging phase is of our main interest.

3 Splitting Methods

On smaller graphs, directing solving all the variables may be possible via standard interior point methods (with aid of barrier methods). However, when N or d is large, the problem is not directly solvable in a acceptable runtime. Therefore, the aid of the splitting method is needed to break the problem into some easier subproblems. In this paper we will describe the steps in two splitting methods, ADMM and AMA, and compare their performance via a simulation study.

3.1 ADMM (Alternating Direction Method of Multipliers)

For ADMM, we follow [Hallac et al., 2015] to introduce two auxiliary variables for each edge (i, j):

- $z_{ij} \approx u_i$
- $z_{ji} \approx u_j$

with the constraint:

$$z_{ij} = u_i, \quad \forall i \in \mathcal{V},\ j \in \mathcal{N}(i)$$

The reformulated problem becomes:

$$\begin{aligned} \min_{\{u_i\},\{z_{ij}\}} & \left(\sum_{i \in \mathcal{V}} f_i(u_i) \ + \ \gamma \sum_{(i,j)\in\mathcal{E}} w_{ij} \|z_{ij} - z_{ji}\|_2 \right) \\ \text{subject to} & \quad u_i = z_{ij}, \quad \forall i \in \mathcal{V},\ j \in \mathcal{N}(i) \end{aligned} \tag{2}$$

The augmented Lagrangian is:

$$\begin{aligned} L_\rho(u, z, \lambda) = & \sum_{i \in \mathcal{V}} f_i(u_i) \\ & + \sum_{(i,j)\in\mathcal{E}} \Bigg(\gamma w_{ij} \|z_{ij} - z_{ji}\|_2 - \frac{\rho}{2} \left(\|\lambda_{ij}\|_2^2 + \|\lambda_{ji}\|_2^2 \right) \\ & \qquad + \frac{\rho}{2} \left(\|u_i - z_{ij} + \lambda_{ij}\|_2^2 + \|u_j - z_{ji} + \lambda_{ji}\|_2^2 \right) \Bigg) \end{aligned} \tag{3}$$

Updates

(1) u-update For each node i, we solve:

$$u_i^{k+1} = \arg\min_{u_i} \left[f_i(u_i) + \frac{\rho}{2} \sum_{j \in N(i)} \|u_i - z_{ij}^k + \lambda_{ij}^k\|_2^2 \right] \tag{4}$$

When $f_i(u_i) = \frac{1}{2}\|u_i - x_i\|_2^2$, the closed-form solution is:

$$u_i^{k+1} = \frac{x_i + \rho \sum_{j\in N(i)} (z_{ij}^k - \lambda_{ij}^k)}{1 + \rho |N(i)|} \tag{5}$$

(2) z-update For each edge (i, j), we jointly update (z_{ij}, z_{ji}) by solving:

$$(z_{ij}^{k+1}, z_{ji}^{k+1}) = \arg\min_{z_{ij}, z_{ji}} \left[\gamma w_{ij} \|z_{ij} - z_{ji}\|_2 + \frac{\rho}{2} \left(\|u_i^{k+1} - z_{ij} + \lambda_{ij}^k\|_2^2 + \|u_j^{k+1} - z_{ji} + \lambda_{ji}^k\|_2^2 \right) \right] \tag{6}$$

We define:

$$v_i = u_i^{k+1} + \lambda_{ij}^k, \quad v_j = u_j^{k+1} + \lambda_{ji}^k$$

The solution is:

$$\begin{aligned} z_{ij}^{k+1} &= \theta v_i + (1-\theta) v_j \\ z_{ji}^{k+1} &= (1-\theta) v_i + \theta v_j \end{aligned} \tag{7}$$

where

$$\theta = \max\left(1 - \frac{\gamma w_{ij}}{\rho \|v_i - v_j\|_2}, 0.5\right)$$

(3) λ-update For each edge (i,j):

$$\begin{aligned} \lambda_{ij}^{k+1} &= \lambda_{ij}^k + u_i^{k+1} - z_{ij}^{k+1} \\ \lambda_{ji}^{k+1} &= \lambda_{ji}^k + u_j^{k+1} - z_{ji}^{k+1} \end{aligned} \tag{8}$$

3.2 AMA (Alternating Minimization Algorithm)

For AMA we follow the splitting in [Chi and Lange, 2015], which introduces an auxiliary variable v_{ij} for each edge (i,j) and reformulates the problem as:

$$\min_{u,v} \sum_{i\in\mathcal{V}} f_i(u_i) + \gamma \sum_{(i,j)\in E} w_{ij}\|v_{ij}\|_2 \quad \text{subject to} \quad v_{ij} = u_i - u_j. \tag{9}$$

Updates

(1) u-update The u-update solves:

$$u^{(k+1)} = \arg\min_u \sum_i f_i(u_i) + \sum_{(i,j)\in\mathcal{E}} \lambda_{ij}^{(k)\top}(u_i - u_j).$$

For the squared loss $f_i(u_i) = \frac{1}{2}\|u_i - x_i\|_2^2$, the closed-form solution is:

$$u_i^{(k+1)} = x_i - \sum_{j\in\mathcal{N}(i)} \left(\lambda_{ij}^{(k)} - \lambda_{ji}^{(k)}\right). \tag{10}$$

(2) v-update (Corrected Formula) The v_{ij}-update is:

$$v_{ij}^{(k+1)} = \arg\min_{v_{ij}} \left(\gamma w_{ij}\|v_{ij}\|_2 - \lambda_{ij}^{(k)\top} v_{ij} + \frac{\nu}{2}\|u_i^{(k+1)} - u_j^{(k+1)} - v_{ij}\|_2^2\right).$$

The closed-form solution is:

$$\boxed{v_{ij}^{(k+1)} = \left(1 - \frac{\gamma w_{ij}}{\nu\,\|s_{ij}\|_2}\right)_+ s_{ij} \quad \text{where} \quad s_{ij} = u_i^{(k+1)} - u_j^{(k+1)} + \frac{1}{\nu}\lambda_{ij}^{(k)}} \tag{11}$$

(3) λ-update For each edge (i,j):

$$\lambda_{ij}^{(k+1)} = \lambda_{ij}^{(k)} + \nu\left(u_i^{(k+1)} - u_j^{(k+1)} - v_{ij}^{(k+1)}\right). \tag{12}$$

Note : no v needed in the algorithm

Since the u-update does not depend on v, we can merge (11) and (12) to one step and implement the algorithm by only updating u and λ, making it more memory-efficient.

AMA (Algorithm 2 in [Chi and Lange, 2015])

```
1: Initialize $\lambda^0$.
2: for $m = 1, 2, 3, \ldots$ do
3:     for $i = 1, \ldots, n$ do
4:         $\Delta_i^{(m)} = \sum_{l_1=i} \lambda_l^{(m-1)} - \sum_{l_2=i} \lambda_l^{(m-1)}$
5:     end for
6:     for all $l$ do
7:         $g_l^{(m)} = u_{l_1}^{(m)} - u_{l_2}^{(m)} + \Delta_{l_1}^{(m)} - \Delta_{l_2}^{(m)}$
8:         $\lambda_l^{(m)} = \mathcal{P}_{C_l}\left(\lambda_l^{(m-1)} - \nu g_l^{(m)}\right)$
9:     end for
10: end for
```

Remark

It is important to note that the u-update is independent of v, meaning that in practice, only u and λ need to be updated. In the presentation I stated the v-update incorrectly, but it does not affect the functionality of the AMA algorithm described above.

4 Simulation Study: Tau Model

In this section, I will implement the two splitting methods to a tau-model simulation study. We will compare the convergence behavior in different regularizations.

4.1 Setup

Tau protein is a neuron-specific protein often used to study interactions between brain regions. Assume there are p regions of interests(ROIs) to be studied. We consider N patients, each with:

- $y^{(k)} \in \mathbb{R}^p$: a binary vector indicating the tau contamination status for patient k. Each entry corresponds to a brain region (ROI), with $y_i^{(k)} = 1$ if region i is infected, and 0 otherwise.

- $\Gamma^{(k)} \in \mathbb{R}^{p\times p}$: a noisy observation of the underlying connectivity matrix $\Omega^{(k)}$, perturbed by $y^{(k)}$-dependent terms.

Patients belonging to the same subtype are assumed to share the same Ω and exhibit similar y patterns.

4.2 Model and Network Lasso Formulation

The tau-model is defined as:

$$\Gamma_{ij}^{(k)} = \Omega_{ij}^{(k)} + \left|y_i^{(k)} - y_j^{(k)}\right| + \left|y_i^{(k)} + y_j^{(k)}\right| + \varepsilon_{ij}^{(k)}, \tag{13}$$

where $\varepsilon_{ij}^{(k)}$ denotes random noise. The Network Lasso optimization problem is:

$$\begin{aligned}\min_{\{\Omega^{(k)}\}} \ \frac{1}{2}\sum_{k=1}^{N}\sum_{i,j}\left(\Omega_{ij}^{(k)} + \left|y_i^{(k)} - y_j^{(k)}\right| + \left|y_i^{(k)} + y_j^{(k)}\right| - \Gamma_{ij}^{(k)}\right)^2 \\ +\lambda \sum_{k<k'} \rho(y^{(k)}, y^{(k')}) \left\|\Omega^{(k)} - \Omega^{(k')}\right\|_F\end{aligned} \tag{14}$$

where:

- $\rho(y^{(k)}, y^{(k')})$ measures the similarity between infection status vectors
- $\lambda > 0$ controls the regularization strength. Here, instead of γ, I set the lasso parameter to be λ

4.3 Data generation and Parameter Settings

Synthetic Data Generation

For each subtype $k = 1, \ldots, K = 3$, we generate data according to the following steps:

1. **Latent vector generation:** For each brain region $i = 1, \ldots, p$, we generate a latent vector $u_i \in \mathbb{R}^d$ from:

$$u_i \sim \mathcal{N}(0, I_d)$$

   with parameters:

   - $p = 68$ (number of brain regions)
   - $d = 3$ (dimension of the latent space)

2. **Baseline connectivity matrix Ω:** We define:

$$\Omega_{ij} = \langle u_i, u_j \rangle,$$

   and set $\Omega_{ii} = 0$.

3. **Graph G generation:** For each pair (i,j), we sample:

$$G_{ij} \sim \text{Bernoulli}(p_{ij}),$$

   where p_{ij} is a probability determined based on Ω_{ij} plus a shift. The shift is chosen such that the density of G is approximately 0.2.

4. **Selection of infection source s_0:** A brain region is randomly selected as the initial infection source for the SI model.

5. For each data point $i = 1, \ldots, n_{\text{data}} = 20$:

   (5.1) **Simulating infection duration T:**

$$T \sim \text{Poisson}(\mu), \quad \mu = 0.2 \times p.$$

   (5.2) **Generating infection status $y^{(i)}$:** We run the SI model $\text{SI}(G, s_0, T)$, starting from s_0, until T infections have occurred. The result is a binary vector $y^{(i)} \in \{0,1\}^p$.

   (5.3) **Generating observation matrix $\Gamma^{(i)}$:** For each (i,j), we compute:

$$\Gamma_{ij}^{(i)} = \Omega_{ij} + |y_i^{(i)} - y_j^{(i)}| + |y_i^{(i)} + y_j^{(i)}| + \varepsilon_{ij}^{(i)},$$

   where $\varepsilon_{ij}^{(i)} \sim \mathcal{N}(0, \sigma^2)$ and $\sigma^2 = 0.5$. Under this setting, the ratio of within-cluster distance to between-center distance is approximately 0.3.

4.4 Similarity Definition

The similarity between sample i and sample j is defined as:

$$\rho(y^{(i)}, y^{(j)}) = \frac{\|y^{(i)} \cdot y^{(j)}\|_0}{\max(\|y^{(i)}\|_0, \|y^{(j)}\|_0)},$$

where:

- $\cdot$ denotes element-wise multiplication,
- $\|\cdot\|_0$ counts the number of non-zero elements (i.e., the number of infected brain regions),
- if both $y^{(i)}$ and $y^{(j)}$ are zero vectors, we set $\rho = 0$.

5 Results and Discussion

First, we choose $\lambda = 17,400$ to compare the performance of ADMM and AMA. For a fair comparison, I set the initialization of both methods to be the same: the primal starts from the observed values, while the auxiliary and dual variables are initialized to zero vectors.

For evaluation I plot the objective value vs runtime, so that the stopping criterion does not influence the comparison.

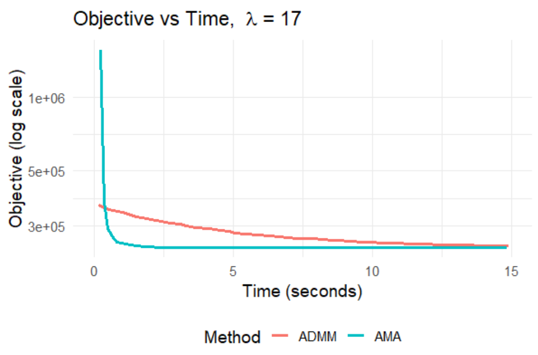

Figure 1: Objective with $\lambda = 17$

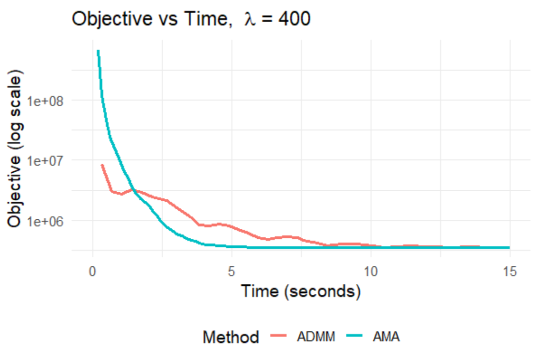

Figure 2: Objective with $\lambda = 400$

In both graphs, we observe that the blue curve (AMA) outperforms the red curve (ADMM) during the early runtime. However, both regularization parameters lie outside the main range of interest: 17 corresponds to the phase before merging begins, while 400 is well after the merging phase. Next, we pick 100 different $\lambda's$ ranging from 10 to 1000, and solve the sequence of problem via both methods.

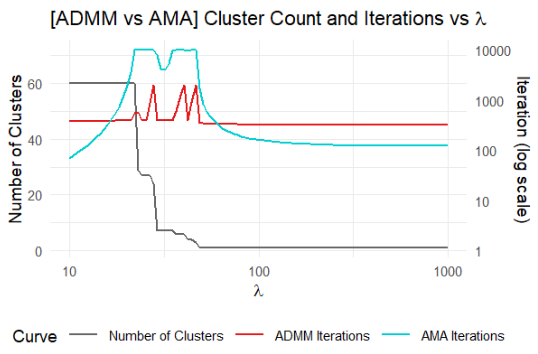

Figure 3: non-Warm-start version

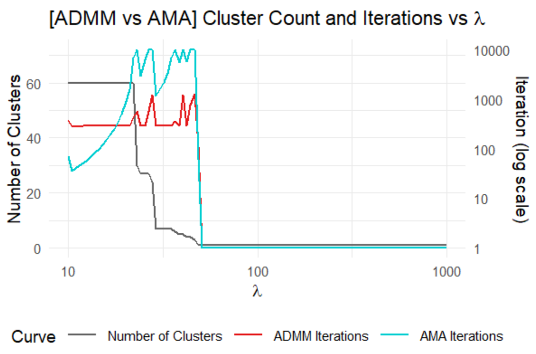

Figure 4: Warm-start version

The number of individuals is $N = 3 * 20 = 60$. For reference, the gray curve shows the number of clusters (here, I use hierarchical clustering with a very small cut-off to address numerical issues). The blue and red curves indicate the number of iterations used in AMA and ADMM, respectively. We can see that AMA outperforms ADMM for small and large values of λ, but it encounters convergence issues during the intermediate merging phase. I set the maximum number of iterations to 10,000; thus, a value of 10,000 indicates failure to converge. Each of the iterations starts from the observed point.

We propose a warm start version RHS. For each iteration, we pick the final primal, auxiliary, and dual of the last iteration to be the starting place. The warm start speeds up both algorithms, but it still cannot resolve the convergence issues of AMA.

6 Conclusion

This study investigated the application of Network Lasso to synthetic data generated from a Tau model, focusing on solving the associated convex clustering problem using two splitting methods: ADMM and AMA. Through extensive simulations, we evaluated their convergence behavior and computational efficiency across a range of regularization parameters.

Our results show that AMA achieves faster convergence in the early runtime for both small and large λ, outperforming ADMM in these regimes. However, AMA consistently encounters convergence issues in the merging phase, whereas ADMM maintains stable convergence across the entire regularization path. Incorporating warm starts further accelerates both methods but does not fully resolve AMA's limitations.

These findings highlight practical trade-offs between the two algorithms. AMA may be preferred when the primary interest lies in fast approximation for extreme λ values, while ADMM provides a more robust option for exploring the full clustering path, especially around critical merging phases.

Future directions include several promising avenues. First, in the current setup, we did not explicitly control the sparsity of the weight matrix; incorporating sparsity constraints may improve computational efficiency, particularly for large-scale problems. Second, while we followed the ADMM formulation described in the[Chi and Lange, 2015], it remains an open question whether this is equivalent to the version introduced in the original Network Lasso paper. Investigating their equivalence and benchmarking their computational performance would provide valuable insights. Finally, the [Chi and Lange, 2015] also proposes an accelerated version of AMA, which could be explored and compared in future experiments to assess its practical benefits.

References

[Chi and Lange, 2015] Chi, E. C. and Lange, K. (2015). Splitting methods for convex clustering. *Journal of Computational and Graphical Statistics*, 24(4):994–1020.

[Hallac et al., 2015] Hallac, D., Leskovec, J., and Boyd, S. (2015). Network lasso: Clustering and optimization in large graphs. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 387–396. ACM.