Network Inference via the Time-Varying Graphical Lasso

David Hallac, Youngsuk Park, Stephen Boyd, Jure Leskovec
Stanford University
{hallac,youngsuk,boyd,jure}@stanford.edu

ABSTRACT

Many important problems can be modeled as a system of interconnected entities, where each entity is recording time-dependent observations or measurements. In order to spot trends, detect anomalies, and interpret the temporal dynamics of such data, it is essential to understand the relationships between the different entities and how these relationships evolve over time. In this paper, we introduce the *time-varying graphical lasso (TVGL)*, a method of inferring time-varying networks from raw time series data. We cast the problem in terms of estimating a sparse time-varying inverse covariance matrix, which reveals a dynamic network of interdependencies between the entities. Since dynamic network inference is a computationally expensive task, we derive a scalable message-passing algorithm based on the Alternating Direction Method of Multipliers (ADMM) to solve this problem in an efficient way. We also discuss several extensions, including a streaming algorithm to update the model and incorporate new observations in real time. Finally, we evaluate our TVGL algorithm on both real and synthetic datasets, obtaining interpretable results and outperforming state-of-the-art baselines in terms of both accuracy and scalability.

1 INTRODUCTION

Applications in many settings, ranging from neurological connectivity patterns [21] to financial markets [23] and social network analysis [1, 22], contain massive sequences of multivariate timestamped observations. Such data can often be modeled as a network of interacting entities, where each entity is a node associated with a time series of data points. In these dependency networks, also known as Markov random fields (MRFs) [16, 26, 33], an edge represents a partial correlation, or a direct effect (holding all other nodes constant) between two entities. An important problem that arises in many applications is using observational data to infer these relationships (*i.e.*, edges) and their evolution over time. In particular, it is necessary to understand how the structure of these complex systems changes over a period of interest (Figure 1). For example, in financial markets, companies can be represented as nodes, and each acts like a "sensor" recording a time series of its stock price. By understanding the relationships within the network and their evolution over time, one can detect anomalies, spot trends, classify events, forecast future behavior, and solve many other problems at the intersection of time series analysis and network science.

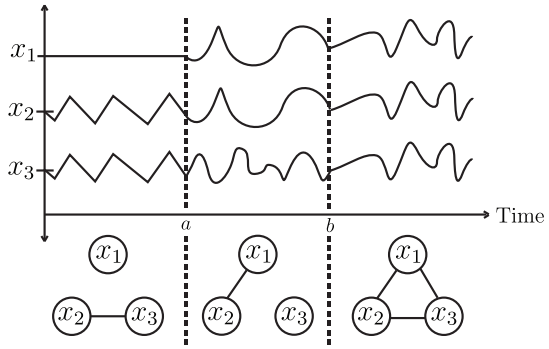


Figure 1: Three sensors with associated time series readings. Based on such data, we infer a time-varying network that reveals 1) the dependencies between the different sensors, and 2) when and how these dependencies change over time.

To learn these dynamic networks, one can model the relationships between the entities through an underlying inverse covariance matrix that changes over time. Doing so allows for inference of a dynamic undirected network, with nodes representing the different entities and edges defining the coupling between them. More precisely, given a multivariate sequence of readings, one can estimate the true inverse covariance matrix $\Sigma^{-1}(t)$ (which changes over time), assuming a Gaussian distribution. The focus is specifically on the inverse covariance because of its increased interpretability: if $\Sigma^{-1}_{ij}(t) = 0$ then, given the values of all the other entities (*i.e.*, nodes), i and j are conditionally independent at time t [17]. Therefore, the inferred network has an edge between i and j at time t if $\Sigma^{-1}_{ij}(t) \neq 0$, denoting a structural dependency between these two entities at that moment in time. In the static case, where Σ^{-1} is constant, this inference is known as the graphical lasso problem [7, 35]. While many efficient algorithms exist for the graphical lasso [2, 14], such methods do not generalize to the time-varying case.

Inferring dynamic networks is challenging mainly because it is difficult to simultaneously estimate both the network itself and the change in its structure over time. This is in part due to the fact that networks can exhibit many different types of changes. The range of possibilities includes a sudden shift of the entire network structure, a single node rewiring all of its connections, or even just one or two edges changing in the whole network. Therefore, any method must be general enough to discover many types of evolutionary patterns, while also being powerful enough to learn this temporal structure over very long time series. As such, solving for time-varying networks is computationally expensive, especially compared to time-invariant inference [6, 19]. There are more parameters, additional coupling, and more complicated dynamics.

KDD '17, August 13-17, 2017, Halifax, NS, Canada

ACM ISBN 978-1-4503-4887-4/17/08…$15.00
https://doi.org/10.1145/3097983.3098037

Standard methods have trouble scaling to large examples, so novel algorithms and techniques are required.

Present Work. In this paper, we formulate time-varying network inference as a convex optimization problem. Thus, our two primary contributions are in formally defining this problem, which we call the *time-varying graphical lasso* (TVGL), and in deriving a scalable optimization method to solve it. Given a sequence of multivariate observations, the TVGL estimates the inverse covariance $\Sigma^{-1}(t)$ of the data to simultaneously achieve three goals: (1) matching the empirical observations, (2) sparsity, and (3) temporal consistency. Matching the empirical observations ensures that our estimated network is well-supported by the data. A sparse inverse covariance prevents overfitting and provides interpretable results via a sparse graphical representation [2, 7]. Temporal consistency is the idea that, most of the time, adjacent timestamps should have very similar (or even identical) estimations of the network. We impose a temporal penalty to limit how the network can evolve, where different penalties induce different dynamics. We suggest five specific penalties that allow us to model different types of temporal variation (or evolutionary patterns) in networks: e.g., smoothly varying networks, rare-but-large-scale shifts, or a single node rewiring its connections. Then, since no scalable methods exist for solving our TVGL problem, we develop a message-passing algorithm using the alternating direction method of multipliers (ADMM) [3]. We derive closed-form solutions for the ADMM subproblems, including one for each of the five unique penalty types, to further speed up the runtime. We also discuss several extensions, including one to convert our approach into a streaming algorithm that can quickly incorporate new data and update our estimate in real time.

We then apply our TVGL method to both real and synthetic datasets. First, we test accuracy and scalability on synthetic examples with known ground truth networks. Our TVGL method leads to accuracy improvements of up to 92% over two state-of-the-art baselines. Furthermore, our ADMM-based implementation is several orders of magnitude faster than other solution methods, able to solve for 5 million unknown variables in under 12 minutes (while other solvers take several hours for even 50 thousand unknowns). Finally, we analyze two real-world case studies, using financial and automobile sensor data, to demonstrate how the TVGL approach can find meaningful insights, understandable structure, and different types of evolutionary patterns in time series data.

Related Work. This work relates to recent advancements in both graphical models and convex optimization. Inferring static networks via the graphical lasso is a well-studied topic [2, 6, 7, 35]. However, previous work on dynamic inference has only focused on a kernel method [36] or an ℓ_1-fused penalty [15, 21, 31]. One of the main contributions of our approach is that it is able to model many different types of network evolutionary patterns, for example a small set of edges rewiring, a single node changing all its edges, or the entire network restructuring at a single time step. This opens up a variety of new applications, several of which we examine in Sections 6 and 7. Whereas previous work allows for only one time-varying pattern [15, 21, 31], we show how the selection of the proper evolutionary penalty is a very important parameter for obtaining accurate results. We also propose several extensions, including a streaming approach, that to the best of our knowledge have not been explored in the literature.

To solve our TVGL problem, we develop an ADMM-based algorithm [3] so that the same framework can incorporate all of the different penalty types that we study. This is necessary because, even though many problem-specific methods exist to solve the standard (static) graphical lasso [7, 14], no ready-to-use methods exist for the time-varying case. To make our algorithm more scalable, we rewrite the ADMM subproblems in terms of proximal operators [6, 21, 27]. This allows us to take advantage of known properties and solution methods to derive closed-form ADMM updates [4, 25], which speed up our solver by several orders of magnitude over a naive ADMM implementation (Section 6.2).

Another common method of time series analysis is the Kalman filter [9], a special type of dynamic factor model [20]. Although used in similar domains, these models are fundamentally different from our network inference approach. They are typically used to predict values of unknown variables, given all previous data. Our algorithm instead learns the underlying dynamic graphical structure of the variables [16, 17]. Thus, our method helps remove the effects of noisy data and adds interpretability to the results. Additionally, we note the difference between our work here and information cascade-based network inference [8, 22], which assumes a viral spreading process over the nodes and aims to infer the links based on the node infection times. We instead aim to uncover the dependency structure (*i.e.*, inverse covariance matrix) based on multivariate time series observations.

2 PROBLEM DEFINITION

Consider a sequence of multivariate observations in $\mathbf{R}^p$ sampled from a distribution $x \sim N(0, \Sigma(t))$. Observations come in at times $0 \leq t_1 \leq \cdots \leq t_T$, where at each time t_i, there are $n_i \geq 1$ different observation vectors over the readings of all the nodes. (For now we assume that the readings are synchronous, where $t_i - t_{i-1}$ is a constant value for all i, but in Section 3.1 we extend this approach to asynchronous observations.) With these samples, we aim to estimate the underlying covariance matrix $\Sigma(t)$, which can change over time. That is, given a changing underlying distribution, we attempt to estimate it based on a sequence of observations. Here, we formulate a convex optimization problem to infer a sequence of sparse inverse covariance matrices, $\Theta_i = \Sigma(t_i)^{-1}$, one for each t_i. These estimates are based on local empirical observation vector(s), as well as coupling constraints with neighboring timestamps' covariance estimates. Recall that a sparse inverse covariance allows us to encode conditional independence between different variables [16].

Inferring Static Networks. We first consider static inference, which is equivalent to the graphical lasso problem [2, 7, 35], and then build on it to extend to dynamic networks. In the static case, $\Sigma(t)$ is constant for all t. Given a series of multivariate readings, this can be written as

$$\text{minimize} \; -l(\Theta) + \lambda \left\|\Theta\right\|_{\text{od},1}, \tag{1}$$

where $\|\Theta\|_{\text{od},1}$ is a seminorm of Θ, the off-diagonal ℓ_1-norm, $\sum_{i \neq j} |\Theta_{ij}|$. This lasso penalty enforces element-wise sparsity in our solution for Θ, regulated by the trade-off parameter $\lambda \geq 0$. In Problem (1),

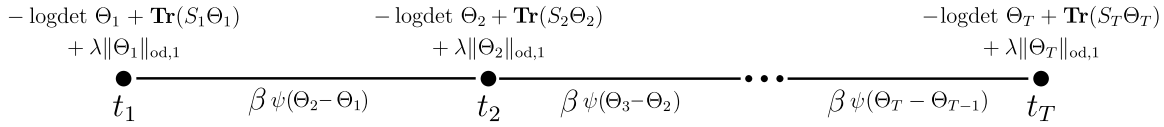


Figure 2: Dynamic network inference can be thought of as an optimization problem on a chain graph, where each node objective solves for a *network slice* at each timestamp, and edge objectives define the penalties that enforce temporal consistency.

$l(\Theta)$ is the log likelihood of Θ (up to a constant and scale) [29, 35],

$$l(\Theta) = n(\log\det\Theta - \mathbf{Tr}(S\Theta)),$$

where Θ must be symmetric positive-definite ($\mathbb{S}^p_{++}$), S is the empirical covariance $\frac{1}{n}\sum_{i=1}^n x_i x_i^T$, n is the number of observations, and x_i are the different samples. When S is invertible, $l(\Theta)$ encourages Θ to be close to S^{-1}.

Inferring Dynamic Networks. In order to infer a time-varying sequence of networks, we extend the above approach and allow $\Sigma(t)$ to vary over time. To do so we set up a sequence of graphical lasso problems, each coupled together in a chain to penalize deviations in the estimations, which we call the *time-varying graphical lasso* (TVGL). We solve for $\Theta = (\Theta_1, \ldots, \Theta_T)$, our estimated inverse covariance matrices at times $t_1, \ldots, t_T$,

$$\underset{\Theta\in\mathbb{S}^p_{++}}{\text{minimize}} \quad \sum_{i=1}^{T} -l_i(\Theta_i) + \lambda\,\|\Theta_i\|_{\text{od},1} + \beta\sum_{i=2}^{T}\psi(\Theta_i - \Theta_{i-1}). \tag{2}$$

Here, $l_i(\Theta_i) = n_i(\log\det\Theta_i - \mathbf{Tr}(S_i\Theta_i))$, $\beta \geq 0$, and $\psi(\Theta_i - \Theta_{i-1})$ is a convex penalty function, minimized at $\psi(0)$, which encourages similarity between Θ_{t-1} and Θ_t. We examine in Section 2.1 how different ψ's enforce different behaviors. Note that Problem (2) can be viewed as an optimization problem defined on a chain graph, where each element solves for a different Θ_i, as shown in Figure 2.

Empirical Covariance. The log-likelihood l_i depends on S_i, the empirical covariance at time t_i. In high-dimensional settings, where the number of dimensions p is larger than the number of observations n_i, S_i will be rank deficient and therefore non-invertible. However, our method is well-suited to overcome this problem of an insufficient number of observations. By enforcing structural similarity, each Θ_i borrows strength from the fact that neighboring network estimates should be similar, or even identical, across time. In the extreme case, we are able to estimate a network at a time where there is only one observation. The empirical covariance, $x_i x_i^T$, is rank 1, but between the sparsity and the temporal consistency penalties, our approach can still infer an accurate estimate of the network at that snapshot in time.

Regularization Parameters λ and β. Problem (2) has two parameters, λ and β, which define important values on two trade-off curves. λ determines the sparsity level of the network: small values will better match the empirical data, but will lead to very dense networks, which are less interpretable and often overfit. β determines how strongly correlated neighboring covariance estimations should be. A small β will lead to Θ's which fluctuate from estimate-to-estimate, whereas large β's lead to smoother estimates over time. As $\beta \to \infty$, the temporal deviation penalty gets so large that Problem (2) turns into the original graphical lasso, since a constant Θ will be the solution across the entire time series.

2.1 Encoding Network Evolutionary Patterns

Different types of penalty functions ψ allow us to enforce different behaviors in the evolution of the network structure. In particular, if we have an expectation about how the underlying network may change over time, we are able to encode it into ψ. Here, we define several common temporal patterns and their associated penalty functions. In Sections 6 and 7, we use these to analyze the time-varying dynamics of multiple real and synthetic datasets. For these penalties, note that ψ can sometimes be split into a sum of column-norms. As such, we refer to the j-th column of a matrix X as $[X]_j$.

- **A few edges changing at a time** — Setting $\psi(X) = \sum_{i,j}|X_{i,j}|$, an element-wise ℓ_1 penalty, encourages neighboring graphs to be identical [6, 34]. When the ij-th edge of the network "breaks", or is different at two neighboring times, this penalty still encourages the rest of the graph to remain *exactly* the same. As a result, this penalty is best used in cases where we expect only a handful of edges — at most — to change at a time.
- **Global restructuring** — Setting $\psi(X) = \sum_j \|[X]_j\|_2$, a group lasso ℓ_2 penalty, causes the entire graph to restructure at a select few timestamps, while at all other times, the graph remains piecewise constant [6, 10]. This is useful for event detection and time-series segmentation, as it finds exact times where there is a regime-change, or shift, in the underlying covariance matrix.
- **Smoothly varying over time** — Setting $\psi(X) = \sum_{i,j} X_{i,j}^2$, a Laplacian penalty, causes smooth transitions of the graphical model from timestamp to timestamp. Adjacent graphs will often differ by small amounts, but severe deviations are largely penalized so they rarely occur [30]. This penalty is best used when we want to come up with a smoothly varying estimate over time.
- **Block-wise restructuring** — Setting $\psi(X)$ to an ℓ_∞ penalty, $\psi(X) = \sum_j \left(\max_i |X_{i,j}|\right)$, implies that, when an element in the inverse covariance matrix changes by ϵ at a given time, other elements are free to change by up to that same amount with no additional penalty (except for the original $\ell_{\text{od},1}$ sparsity condition). This is best used when a cluster of nodes suddenly changes its internal edge structure, the rest of the network does not change.
- **Perturbed node** — Setting ψ to the row-column overlap penalty [18], defined as $\psi(X) = \min_{V:V+V^T=X} \sum_j \|[V]_j\|_2$, yields an interesting behavior. When node i has a reweighting of one edge at time t, this says that the node can rewire all its edges at that same time with a minimal penalty. However the rest of the graph is strongly encouraged to remain the exact same. It is best used in situations where we are looking for single nodes re-locating themselves to a new set of neighbors within the network.

3 EXTENSIONS

Here, we develop three extensions of the basic time-varying graphical lasso. First, we update our approach to allow for asynchronous observations of the data. Then, we derive a method of inferring an estimate at *any* time, even if there is no temporally-local data. Finally, we extend the algorithm so it can be deployed in an application with streaming data and real-time update constraints.

Asynchronous Observations. Problem (2) can be extended to asynchronous settings, where samples are observed at irregularly-spaced intervals. We rewrite the problem as

$$\underset{\Theta \in \mathbf{S}^p_{++}}{\text{minimize}} \quad \sum_{i=1}^{T} -l_i(\Theta_i) + \lambda \|\Theta_i\|_{\text{od},1} + \beta \sum_{i=2}^{T} h_i \psi\left(\frac{\Theta_i - \Theta_{i-1}}{h_i}\right),$$

where $h_i = t_i - t_{i-1}$, the interval of time between sample $i-1$ and sample i. This is similar to putting h_i different intermediate Θ's between the two samples (with no loss, since there is no data). Since ψ is convex, we know that the loss is minimized by evenly spreading the change in $\Theta_i - \Theta_{i-1}$ across these h_i steps. We also know from convexity that this penalty gets smaller as h_i gets larger. That is, the temporal consistency penalty is less important when consecutive samples have a large time gap between them. Of course, we can scale this penalty through the regularization parameter β to account for the average frequency of the data.

Inferring Intermediate Networks. With this approach, it is possible to infer the covariance estimation at *any* time, even if there are no observations at that moment in time. This can be very useful if, for example, we want to get a granular estimate of a sharp breakpoint. To infer a network estimate at intermediate time s, we create a dummy node at s and merge it into the chain graph by connecting it to the nearest observation in either direction, $j-1$ and j. To get an estimate of Θ_s, we solve for

$$\underset{\Theta_s \in \mathbf{S}^p_{++}}{\text{minimize}} \quad w(s - t_{j-1})\psi(\Theta_s - \Theta_{j-1}) + w(t_j - s)\psi(\Theta_j - \Theta_s).$$

For most common ψ's, this problem has a closed-form solution. With this we can efficiently "upsample" and predict underlying covariances as frequently as our application requires, even if the data has a slower sampling frequency.

Streaming Algorithm. In many applications, it is necessary to deploy a network inference scheme that updates in real time as new observations continue to arrive. Therefore, we require a streaming algorithm to quickly incorporate new data into our model. That is, given that we have solved for a problem with i timestamps, we look for a fast way to update the solution when we receive a new observation at time t_{i+1}.

One approach is to use a warm-start ADMM. With this method, we solve for the $i+1$ covariance matrices using the standard ADMM approach, but the first i Θ's initialize themselves at the value of the solution to the previous problem. However, there are no guarantees as to how many iterations warm-start ADMM may take to converge. In particular, as i gets large, there is a risk that one single additional reading takes a very long time as the new information needs to propagate across the entire time series. Therefore, it may take longer to incorporate the 1000th reading than the 100th. If we want a real-time implementation of this algorithm, we need a way

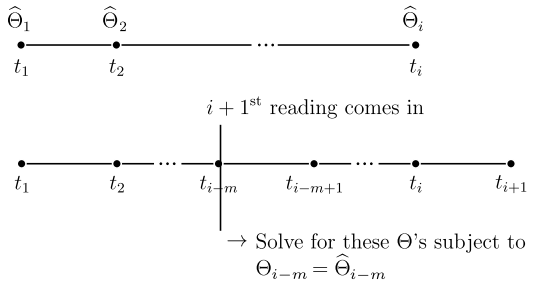


Figure 3: To quickly update our estimates when the observation at t_{i+1} arrives, we re-solve for the most recent m timestamps, while enforcing that Θ_{i-m} must remain the same.

of guaranteeing that it takes the same amount of time to solve, regardless of the current timestamp.

We do so using a small approximation where we fix the result a certain distance in the past, and only solve for the m most recent nodes (Figure 3). Therefore, if the i + 1-st reading comes in, we only solve for nodes $i-m$ to $i+1$, subject to the constraint that $\Theta_{i-m} = \hat{\Theta}_{i-m}$, where $\hat{\Theta}$ is the solution from when there were only i timestamps. We can pick m based on memory limitations, previous breakpoints, or domain expertise.

4 PROPOSED ALGORITHM

When inferring small networks, or those without time-varying dynamics, Problem (2) can be solved using standard interior-point methods. However, our paper focuses on larger examples, where it is infeasible to solve the whole problem at once. Here, we propose the time-varying graphical lasso (TVGL) algorithm, based on the alternating direction method of multipliers (ADMM) [3], a well-established distributed convex optimization approach. With ADMM, we split the problem up into a series of subproblems and use a message-passing algorithm to converge on the globally optimal solution. In this section, we analyze the separable subproblems and develop analytical solutions, which are fast and easy to implement, for every step in the ADMM process. To do so, we rewrite terms in the form of proximal operators [25], which are defined for a matrix $A \in \mathbf{R}^{m\times n}$ and the real-valued function $f(X)$ as

$$\mathbf{prox}_{\eta f}(A) = \underset{X \in \mathbf{R}^{m\times n}}{\text{argmin}} \left(f(X) + 1/(2\eta)\|X - A\|_F^2\right). \tag{3}$$

The proximal operator defines a trade-off for X between minimizing f and being near A. Writing the problems in this form allows us to take advantage of well known properties to find closed-form updates for each of the ADMM subproblems.

4.1 ADMM Solution

To split Problem (2) into a separable form, we introduce a consensus variable $Z = \{Z_0, Z_1, Z_2\} = \{(Z_{1,0}, \ldots, Z_{T,0}), (Z_{1,1}, \ldots, Z_{T-1,1}), (Z_{2,2}, \ldots, Z_{T,2})\}$. With this, we can rewrite Problem (2) as its equivalent problem,

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{T} -l_i(\Theta_i) + \lambda\|Z_{i,0}\|_{\text{od},1} + \beta\sum_{i=2}^{T}\psi(Z_{i,2} - Z_{i-1,1}) \\ \text{subject to} \quad & Z_{i,0} = \Theta_i,\ \Theta_i \in \mathbf{S}^p_{++} \quad \text{for} \quad i = 1, \ldots, T \\ & (Z_{i-1,1}, Z_{i,2}) = (\Theta_{i-1}, \Theta_i) \quad \text{for} \quad i = 2, \ldots, T. \end{aligned}$$

The corresponding augmented Lagrangian [13] becomes

$$\mathcal{L}_\rho(\Theta, Z, U) = \sum_{i=1}^{T} -l(\Theta_i) + \lambda \left\|Z_{i,0}\right\|_{\text{od},1} + \beta \sum_{i=2}^{T} \psi(Z_{i,2} - Z_{i-1,1})$$
$$+ (\rho/2) \sum_{i=1}^{T} \left(\|\Theta_i - Z_{i,0} + U_{i,0}\|_F^2 - \|U_{i,0}\|_F^2 \right)$$
$$+ (\rho/2) \sum_{i=2}^{T} \Big(\|\Theta_{i-1} - Z_{i-1,1} + U_{i-1,1}\|_F^2 - \|U_{i-1,1}\|_F^2$$
$$+ \|\Theta_i - Z_{i,2} + U_{i,2}\|_F^2 - \|U_{i,2}\|_F^2 \Big), \tag{4}$$

where $U = \{U_0, U_1, U_2\} = \{(U_{1,0}, \ldots, U_{T,0}), (U_{1,1}, \ldots, U_{T-1,1}), (U_{2,2}, \ldots, U_{T,2})\}$ is the scaled dual variable and $\rho > 0$ is the ADMM penalty parameter [3, §3.1.1]. ADMM consists of the following updates, where k denotes the iteration number:

$$(a) \quad \Theta^{k+1} := \underset{\Theta \in \mathbb{S}_{++}^p}{\operatorname{argmin}} \mathcal{L}_\rho\left(\Theta, Z^k, U^k\right)$$

$$(b) \quad Z^{k+1} = \begin{bmatrix} Z_0^{k+1} \\ Z_1^{k+1} \\ Z_2^{k+1} \end{bmatrix} := \underset{Z_0, Z_1, Z_2}{\operatorname{argmin}} \mathcal{L}_\rho\left(\Theta^{k+1}, Z, U^k\right)$$

$$(c) \quad U^{k+1} = \begin{bmatrix} U_0^{k+1} \\ U_1^{k+1} \\ U_2^{k+1} \end{bmatrix} := \begin{bmatrix} U_0^{k} \\ U_1^{k} \\ U_2^{k} \end{bmatrix} + \begin{bmatrix} \Theta^{k+1} - Z_0^{k+1} \\ (\Theta_1^{k+1}, \ldots, \Theta_{T-1}^{k+1}) - Z_1^{k+1} \\ (\Theta_2^{k+1}, \ldots, \Theta_T^{k+1}) - Z_2^{k+1} \end{bmatrix}.$$

Global Convergence. By separating Problem (2) into two blocks of variables, Θ and Z, our ADMM approach is guaranteed to converge to the global optimum. Our iterative algorithm uses a stopping criterion based on the primal and dual residual values being below specified thresholds; see [3].

4.2 Θ-Update

The Θ-step can be split into separate updates for each Θ_i, which can then be solved in parallel:

$$\Theta_i^{k+1} \overset{(a)}{=} \underset{\Theta_i \in \mathbb{S}_{++}^p}{\operatorname{argmin}} \; -\log\det(\Theta_i) + \mathbf{Tr}(S_i\Theta_i) + \frac{1}{2\eta}\|\Theta_i - A\|_F^2$$
$$\overset{(b)}{=} \underset{\Theta_i \in \mathbb{S}_{++}^p}{\operatorname{argmin}} \; -\log\det(\Theta_i) + \mathbf{Tr}(S_i\Theta_i) + \frac{1}{2\eta}\left\|\Theta_i - \frac{A + A^T}{2}\right\|_F^2$$

where (a) holds for $A = \frac{Z_{i,0}^k + Z_{i,1}^k + Z_{i,2}^k - U_{i,0}^k - U_{i,1}^k - U_{i,2}^k}{3}$ and $\eta = \frac{n_i}{3\rho}$, and (b) holds due to the symmetry of Θ_i. This can be rewritten as a proximal operator,

$$\Theta_i^{k+1} = \mathbf{prox}_{\eta(-\log\det(\cdot) + \mathbf{Tr}(S_i\cdot))}\left((A + A^T)/2\right).$$

Since $\frac{A+A^T}{2}$ is symmetric, this has an analytical solution,

$$\Theta_i^{k+1} := \frac{1}{2\eta^{-1}} Q\left(D + \sqrt{D^2 + 4\eta^{-1}I}\right)Q^T, \tag{5}$$

where QDQ^T is the eigendecomposition of $\eta^{-1}\frac{A+A^T}{2} - S_i$ [6, 32]. This is the most computationally expensive task in our algorithm, as decomposing a $p \times p$ matrix has an $O(p^3)$ runtime.

4.3 Z-Update

The Z-update can be split into two parts: Z_0, which refers to the $\|\Theta\|_{\text{od},1}$-penalty that enforces sparsity in the inverse covariance matrices, and (Z_1, Z_2), which denotes the ψ-penalty that minimizes deviations across timestamps. These two updates can be solved simultaneously, and each part can be parallelized even further to speed up computation.

4.3.1 Part 1: Z_0-Update. Each $Z_{i,0}$ can be written as the proximal operator of the $\ell_{\text{od},1}$-norm, which has a known closed-form solution [25]

$$Z_{i,0}^{k+1} = \mathbf{prox}_{\frac{\lambda}{\rho}\|\cdot\|_{\text{od},1}}(\Theta_i^{k+1} + U_{i,0}^k) = S_{\frac{\lambda}{\rho}}(\Theta_i^{k+1} + U_{i,0}^k),$$

where $S_{\frac{\lambda}{\rho}}(\cdot)$ is the element-wise soft-threshold function, for $1 \le i \ne j \le p$. The (i,j)-th element of this update is

$$\left(S_{\frac{\lambda}{\rho}}(A)\right)_{ij} = \begin{cases} 0 & |A_{ij}| \le \frac{\lambda}{\rho} \\ \mathbf{sgn}(A_{ij})(|A_{ij}| - \frac{\lambda}{\rho}) & \text{otherwise.} \end{cases}$$

4.3.2 Part 2: (Z_1, Z_2)-Update. $Z_{i-1,1}$ and $Z_{i,2}$ are coupled together in the augmented Lagrangian, so they must be jointly updated. In order to derive the closed-form solution, we define

$$\tilde{\psi}\left(\begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix}\right) = \psi(Z_2 - Z_1).$$

We now solve a separate update for each (Z_1, Z_2) pair,

$$\begin{bmatrix} Z_{i-1,1}^{k+1} \\ Z_{i,2}^{k+1} \end{bmatrix} = \mathbf{prox}_{\frac{\beta}{\rho}\tilde{\psi}(\cdot)}\left(\begin{bmatrix} \Theta_{i-1}^{k+1} + U_{i-1,1}^k \\ \Theta_i^{k+1} + U_{i,2}^k \end{bmatrix}\right). \tag{6}$$

Part 2-1: (Z_1, Z_2)-Updates for the Sum of Column Norms. For the ℓ_1, ℓ_2, ℓ_2^2, and ℓ_∞ penalties defined in Section 2.1, we solve the proximal operator by utilizing the following two properties [25, 28]:

- If a function f is a composition of another function g with an orthogonal affine transformation, i.e., $f(x) = g(Cx + D)$ and $CC^T = (1/\alpha)I$, then
$$\mathbf{prox}_f(x) = (I - \alpha C^T C)x + \alpha C^T(\mathbf{prox}_{\frac{1}{\alpha}g}(Cx + D) - D).$$
- If f is block-separable, i.e., $f(x) = \sum_j f_j(x_j)$ where $x = (x_1, x_2, \ldots)$, then $\left(\mathbf{prox}_f(v)\right)_j = \mathbf{prox}_{f_j}(v_j)$.

Recall that our goal is to get the analytical solution to (6). To do so, we apply the first property with $f = \tilde{\psi}$, $g = \psi$, $C = \begin{bmatrix} -I & I \end{bmatrix}$, $D = 0$, and $\alpha = \frac{1}{2}$, which converts the (Z_1, Z_2)-update into

$$\begin{bmatrix} Z_{i-1,1}^{k+1} \\ Z_{i,2}^{k+1} \end{bmatrix} \overset{(a)}{=} \frac{1}{2}\begin{bmatrix} \Theta_{i-1}^{k+1} + \Theta_i^{k+1} + U_{i-1,1}^k + U_{i,2}^k \\ \Theta_{i-1}^{k+1} + \Theta_i^{k+1} + U_{i-1,1}^k + U_{i,2}^k \end{bmatrix} + \frac{1}{2}\begin{bmatrix} -E \\ E \end{bmatrix},$$

where (a) holds for

$$E = \mathbf{prox}_{\frac{2\beta}{\rho}\psi}\left(\left[\Theta_i^{k+1} - \Theta_{i-1}^{k+1} + U_{i,2}^k - U_{i-1,1}^k\right]\right).$$

Now, for each penalty function ψ, we simply need to solve for the corresponding E. We denote $A = \left[\Theta_{i-1}^{k+1} - \Theta_i^{k+1} + U_{i-1,1}^k - U_{i,2}^k\right]$ and $\eta = \frac{2\beta}{\rho}$. Then, since ψ is just the sum of column-norms and thus block-separable, we simplify E as

$$[E]_j = \left(\mathbf{prox}_{\eta\psi}(A)\right)_j \overset{(b)}{=} \mathbf{prox}_{\eta\phi}([A]_j), \tag{7}$$

where ϕ is the column-norm for ℓ_1, ℓ_2, ℓ_2^2, and ℓ_∞. Note that we have narrowed the $(Z_{i-1,1}, Z_{i,2})$-update in (6) down to just finding E_j in (7), expressed by the proximal operator of ϕ defined on a vector, each of which has a closed form solution as follows:

Element-wise ℓ_1 Penalty. The ℓ_1 proximal operator is

$$[E]_j = \mathbf{prox}_{\eta\|\cdot\|_1}([A]_j) = S_\eta(A_j).$$

This is just the element-wise soft threshold,

$$E_{ij} = S_\eta(A_{ij}) = \begin{cases} 0 & |A_{ij}| \le \eta \\ \mathbf{sgn}(A_{ij})(|A_{ij}| - \eta) & \text{otherwise.} \end{cases}$$

Group Lasso ℓ_2 Penalty. The proximal operator for the ℓ_2-norm is a block-wise soft thresholding,

$$[E]_j = \mathbf{prox}_{\eta\|\cdot\|_2}([A]_j) = \begin{cases} 0 & \|[A]_j\|_2 \le \eta \\ (1 - \eta/\|[A]_j\|_2)[A]_j & \text{otherwise.} \end{cases}$$

Laplacian Penalty. The proximal operator for the ℓ_2^2-norm, Laplacian regularization, is given by

$$[E]_j = \mathbf{prox}_{\eta\|\cdot\|_2^2}([A]_j) = (1 + 2\eta)^{-1}([A]_j).$$

This can be rewritten in element-wise form

$$E_{ij} = (1 + 2\eta)^{-1}(A_{ij}).$$

ℓ_∞ Penalty. The proximal operator for the ℓ_∞-norm is

$$[E]_j = \mathbf{prox}_{\eta\|\cdot\|_\infty}([A]_j) = \begin{cases} 0 & \|[A]_j\|_1 \le \eta \\ [A]_j - \eta S_\sigma([A]_j/\eta) & \text{otherwise,} \end{cases}$$

where σ is the solution to $\sum_{i=1}^n \max\{A_{ij}/\eta - \sigma, 0\} = 1$, which has no closed-form solution but can be solved via bisection.

Part 2-2: (Z_1, Z_2)-Update for Perturbed Node Penalty. The perturbed node proximal operator does not have an efficient analytical solution. However, we can solve this new problem by deriving a second ADMM algorithm. Here, in each iteration of our original ADMM solution, we now call this second ADMM solver.

In order to avoid notational conflict, we denote the minimization variables $(Z_{i-1,1}, Z_{i,2})$ as (Y_1, Y_2). We then introduce an additional variable $V = W^T$, and the augmented Lagrangian $\mathcal{L}_\rho$ becomes

$$\mathcal{L}_\rho(V, W, Y_1, Y_2) = \beta\|V\|_2 + \frac{\rho}{2}\left\|\begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} - \begin{bmatrix} \Theta_{i-1}^{k+1} + U_{i-1,1}^k \\ \Theta_i^{k+1} + U_{i,2}^k \end{bmatrix}\right\|_F^2 + \frac{\rho}{2}\left\|V + W - (Y_1 - Y_2) + \tilde{U}_1\right\|_F^2 + \frac{\rho}{2}\left\|V - W^T + \tilde{U}_2\right\|_F^2,$$

where $(\tilde{U}_1, \tilde{U}_2)$ is the scaled dual variable and ρ is the same ADMM penalty parameter as outer ADMM. At the l-th iteration, the three steps in the ADMM update are as follows:

$$(a) \quad V^{l+1} = \mathbf{prox}_{\frac{\beta}{2\rho}\|\cdot\|_2}\left(\frac{Y_1^l - Y_2^l - W^l - \tilde{U}_1^l + ((W^l)^T - \tilde{U}_2^l)^T}{2}\right),$$

which has the following closed form solution for the jth column, with $A = \frac{Y_1^l - Y_2^l - W^l - \tilde{U}_1^l + ((W^l)^T - \tilde{U}_2^l)^T}{2}$,

$$[V^{l+1}]_j = \begin{cases} 0 & \|[A]_j\|_2 \le \frac{\beta}{2\rho} \\ (1 - 1/(2\rho\|[A]_j\|_2))[A]_j & \text{otherwise.} \end{cases}$$

$$(b) \begin{bmatrix} W^{l+1} \\ Y_1^{l+1} \\ Y_2^{l+1} \end{bmatrix} = (C^TC + 2I)^{-1}\left(2\begin{bmatrix} (V^l + \tilde{U}_2^l)^T \\ \Theta_{i-1}^{k+1} + U_{i-1,1}^k \\ \Theta_i^{k+1} + U_{i,2}^k \end{bmatrix} - C^TD\right),$$

where $C = \begin{bmatrix} I & -I & I \end{bmatrix}$, and $D = (V^l + \tilde{U}_1^l)$.

$$(c) \begin{bmatrix} \tilde{U}_1^{l+1} \\ \tilde{U}_2^{l+1} \end{bmatrix} = \begin{bmatrix} \tilde{U}_1^l \\ \tilde{U}_2^l \end{bmatrix} + \begin{bmatrix} (V^{l+1} + W^{l+1}) - (Y_1^{l+1} - Y_2^{l+1})) \\ V^{l+1} - (W^{l+1})^T \end{bmatrix}.$$

5 IMPLEMENTATION

We have built a custom TVGL Python solver[1] on top of SnapVX [11], an open-source convex optimization package. Our solver takes as inputs the multivariate observations, the regularization parameters, and the type of penalty (ℓ_1, ℓ_2, Laplacian, ℓ_∞, or perturbed node), and it returns the time-varying network. Although TVGL is capable of being distributed across many machines, we instead distribute it across multiple cores of a single large-memory machine.

6 EXPERIMENTS

Here, we run several experiments on synthetic data, where there are clear ground truth networks, to test the accuracy and scalability of our network inference approach. First, we compare our TVGL method to two state-of-the-art baselines to measure accuracy, and we demonstrate the importance of using appropriate penalties for different types of temporal evolutions. Next, we vary the problem size over several orders of magnitude and test our ADMM-based algorithm's scalability compared to three other solution methods.

6.1 Accuracy on Synthetic Data

We first analyze a synthetic problem in which the observations are generated from a changing underlying covariance matrix. This provides a known ground truth network, which we can use to verify the accuracy of our network inference approach.

Experimental Setup. We evaluate two different types of temporal evolutions: a global shift, where the entire structure of the network changes at some time t, and a single node perturbation (which we refer to as a local shift), where one node rewires its connections all at once but the rest of the graph remains the same. We randomly generate the ground truth covariance and subsequent samples using the method outlined by Mohan et al. [19]. For both examples, we generate data in $\mathbf{R}^{10}$ over 100 timestamps, where the shift (either global or local) occurs at time $t = 50$. At each t, we observe 10 independent samples from the true distribution.

From this time series of observations, we then solve for Θ_t, $t = 1, \ldots, 100$, our estimate of the dynamic network across this time period. Here, we set the regularization parameters λ and β as the values that minimize the Akaike Information Criteria (AIC) [12] (on a separate, independently generated training set).

Baseline Methods. We compare our approach to two different baselines: the static graphical lasso [7] and the kernel method [36]. For the static graphical lasso, we treat each t_i as an independent network. Since our data has 100 time steps, this means we solve 100 independent graphical lasso problems and infer 100 separate networks. For the kernel method, we modify the empirical covariances

[1] Code and solver can be found at http://snap.stanford.edu/tvgl/.

True Shift	Score	Static GL	Kernel	TVGL (ℓ_1)	TVGL (ℓ_2)	TVGL (Perturbed Node)
Local	F_1	0.646	0.768	0.819	0.817	**0.853**
	TD ratio	2.02	2.41	27.9	23.3	**55.5**
Global	F_1	0.496	0.688	0.939	**0.952**	0.943
	TD ratio	1.06	1.80	**47.6**	38.6	36.2

Table 1: F_1 score and Temporal Deviation (TD) ratio for a local and global shift, using the two baselines and three different TVGL evolutionary penalties.

and weight them according to a non-negative kernel function. We set the kernel width to the theoretically guaranteed optimum using the method proposed by Zhou et al. [36].

Performance Measures. We introduce two metrics to measure the accuracy of our estimate:

- **F_1 score:** This measures how closely we capture the true edge structure of the network (*i.e.*, how many of the non-zero elements in the inverse covariance we correctly identify as non-zero). This score is the harmonic mean of the precision and recall.
- **Temporal deviation (TD) ratio:** The temporal deviation, $\|\Theta_i - \Theta_{i-1}\|_F$, shows how much the estimate has changed at each timestamp. This score is the ratio of the temporal deviation at $t = 50$ (where the one "true" shift happened) to the average temporal deviation value across the 100 timestamps.

Experimental Results. We show results for both the local and global shift with several different TVGL penalties in Table 1. In terms of both F_1 score and temporal deviation, our TVGL approach significantly outperforms the two baselines. The TVGL F_1 score is up to 38.4% higher than the kernel method and 91.9% higher than the static graphical lasso. Regardless of penalty type, the TVGL also always has a temporal deviation (TD) ratio at least 9.7 times larger than any of the baseline methods. In fact, for both baselines and both shift types, the largest temporal deviation peak in the time series does *not* occur at $t = 50$. This means that the baselines do not detect that there is a large shift in the network at this time, whereas this sudden change is clearly discovered by TVGL (where the largest peak always occurs at $t = 50$). We later use this idea in Section 7 to detect significant events in real-world time series data.

Selection of Penalty Type. While the TVGL outperformed the two baselines regardless of the penalty type, even greater gains can be achieved by selecting the correct evolutionary penalty. In real world cases, this parameter can be selected by cross-validation or by incorporating domain knowledge, using the descriptions in Section 2.1 to choose the proper penalty based on exactly what type of temporal evolution one is looking for in the data. As shown in Table 1, there are clear benefits from using certain penalties in certain situations. For example, with a local shift, which is well-suited to be analyzed by a perturbed node penalty, choosing this penalty leads to a 5% higher F_1 score and a temporal deviation ratio that is more than twice as large as both the ℓ_1 or ℓ_2 cases. For the Global shift, the ℓ_2 penalty does the best job at reconstructing the time-varying network (largest F_1 score), though the ℓ_1 is better able to identify the sudden shift at $t = 50$ (with its TD ratio 23% larger than either of the other two penalty types). This additional selection parameter, which to the best of our knowledge has not been previously explored in this context, expands the reach of time-varying inference methods by allowing us to model various types of network evolutions with high precision.

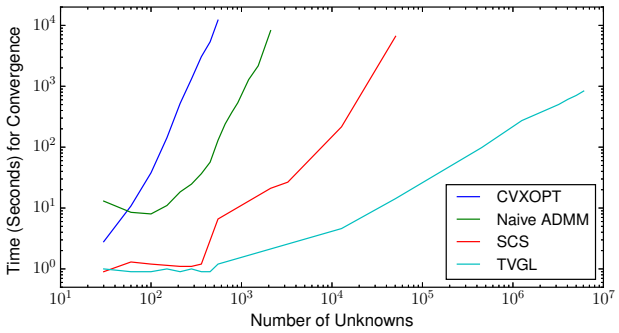


Figure 4: Scalability comparison between our TVGL solver and three other solution methods.

6.2 Scalability of TVGL

Next, we examine the scalability of our TVGL algorithm. We run our TVGL solver on data generated by the method outlined in Section 6.1, and we vary our problem size over several orders of magnitude. Here, we estimate m slices of a n-node network, for a total of $m\frac{n(n+1)}{2}$ unknown variables.

While many algorithms exist to efficiently solve the static graphical lasso problem (*e.g.*, [7, 14]), these cannot be directly applied to our problem because of the time-varying penalties coupling the variables together. Instead, we compare our runtime against three alternative methods that can solve our time-varying problem: two semidefinite programming solvers (CVXOPT [5] and SCS [24]) and a naive ADMM method (without the closed-form updates that we developed in §4). We experiment on a single 40-core CPU where the entire problem fits into memory. We set $m = 10$ and modify n to vary the problem size. Even though our TVGL algorithm can solve for problems with much larger values of m, it is intractable to scale the other methods beyond this point, and we run the experiments in identical conditions to isolate our algorithm's effect on scalability.

We compare the performance of the four solvers in Figure 4. As shown, problems which take hours for the other solvers can be solved in seconds using TVGL. For example, to solve for 50,000 unknowns, TVGL is over 400 times faster than SCS, the second fastest solver (14.3 vs. 6,611 seconds). The main difference in solution time is due to the closed-form ADMM solutions we derived in Section 4.1 for each of the ADMM subproblems. Our problem has a semidefinite programming (SDP) constraint, which is particularly difficult to solve via standard methods. To infer a single n-by-n covariance, the largest per-iteration cost in our algorithm is $O(n^3)$, the cost of an eigendecomposition during the Θ-update. For the same problem, general interior-point methods have a runtime of $O(n^6)$ [19]. These numbers empirically appear to hold true in Figure 4 (recall that the total number of unknowns in the x-axis scales with $O(n^2)$).

7 CASE STUDIES

We next apply the TVGL to two real-world domains to illustrate several basic examples of how our approach can be used to learn meaningful insights from multivariate time series data.

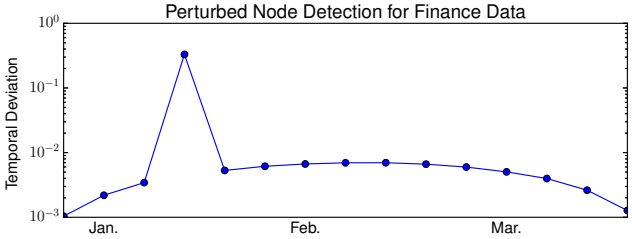
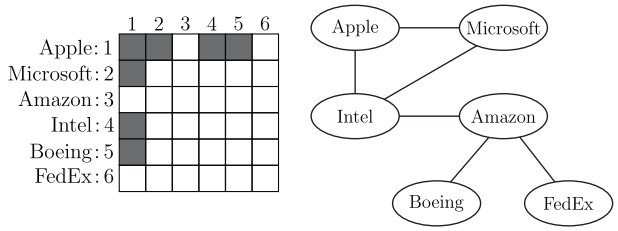


Figure 5: Plot of the temporal deviation of our estimated stock network, which detects a local shift in late January.



(a) (b)

Figure 6: Sparsity structure of (a) the network change during the local shift, indicating that Apple is the perturbed node, (b) the relationships in the financial network.

7.1 Applications in Financial Data

By examining historical stock prices, we can infer a financial network to model relationships between different companies. Learning the structure of this network is useful because it allows us to devise models of how certain stocks are related, which can be used to predict future prices, understand economic trends, or even diversify a portfolio by avoiding highly correlated stocks. We infer this network by examining the stock prices of several large companies in 2010. We apply our method by treating the closing price of each stock as a daily "sensor" observation. Our inferred graphical representation then shows how the stock prices affect each other.

Identifying Single-Stock Perturbations. We observe daily stock prices for six large American companies: Apple, Google, Amazon, Intel, Boeing, and FedEx. Generally, large companies are well-established in the market, so we would expect their correlation network to change only slightly over time. However, events occasionally occur which cause a sudden shift in the network structure. We look at the dependencies of these companies while using a perturbed node penalty. This enforces a temporal dynamic where single nodes may occasionally reweigh all their connections at once. This typically reflects that something happened to affect just one company, while leaving the rest of the network unchanged. We solve the TVGL optimization problem with a perturbed node penalty and discover that one event stood out as having the *largest single-node effect* on the dynamics between these stocks.

After running our TVGL method, we plot the temporal deviation, $\|\Theta_i - \Theta_{i-1}\|_F$, in Figure 5. We discover that there is a large spike in the temporal deviation score during the last week of January. This represents a significant "shift" in the network at this specific time. We show the network change at this shift in Figure 6(a), where we see that the perturbed stock was Apple. At this timestamp, only Apple's edges were affected, reflecting a local rather than global change in the network. We examined the media to understand what may have caused this shift, and we found that on January 27th, Apple first introduced the original iPad to the public. This corresponds to the exact time that our TVGL method captured a structural change in the network, where it was also able to identify Apple as the cause of this shift.

We also plot the post-announcement network in Figure 6(b). Analyzing the correlation structure yields some insightful relationships. The four technology companies—Apple, Google, Amazon, and Intel—are closely related. The shipping company (FedEx) and the airplane manufacturer (Boeing) are both connected to only one company, Amazon. Amazon heavily depends on both companies to deliver products on time, while both rely on Amazon for a large portion of their business. Interestingly, our model predicts that FedEx and Boeing's stock prices are conditionally independent given Amazon, a trend that holds true across the entire dataset.

Detecting Large-Scale Events. Time-varying inference can also be used to detect significant events that affected the entire network, not just single entities within it. We run a similar experiment to the previous example, except we now include every company in the S&P 500, which covers 500 of the largest US-based public companies. Since we are focusing on macro-level event detection, we look for the maximum temporal deviation with an ℓ_1 penalty. This point in time represents the largest "shock" to the network, the timestamp where our TVGL model detected a large and sudden change. We discover that a shift happens during the week of May 6th, 2010. Surprisingly, we saw that there was in fact a "Flash Crash" that day, where the entire market dropped 9% in a matter of seconds, only to rebound back just minutes later. Our results imply that, even though the market recovered from the crash in terms of stock price values, there were tangible long-term effects on the correlation network between companies in the S&P 500.

7.2 Application to Automobile Sensors

Inferring relationships between interrelated entities is of particular interest to industries with large amounts of sensor data. One such industry is automobiles, where modern cars contain hundreds of sensors measuring everything from the car's velocity to the slope of the road. Inferring a time-varying network of relationships between these sensors is an informative way to understand driving habits, as each driver creates a unique "signature", or dynamic sensor network, while driving. This network can be used to model behavior, compare driver ability, or even detect impaired drivers.

As a second case study, we analyze an automobile sensor dataset, provided by a large car company. We look at a short session where a driver goes down a straight road, makes a right turn, and continues on a new road. We observe eight different sensors every 0.1 seconds: steering wheel angle, steering wheel velocity, vehicle velocity, brake pedal, gas pedal, forward acceleration, lateral acceleration, and engine RPM. Since we do not expect any sudden shifts in this network, we use a Laplacian penalty to enforce a smoothly varying network across time. We run our TVGL algorithm and plot three snapshots of the sensor network in Figure 7: one on the straightaway before the turn, one in the middle of turning, and one after the turn. Note that the "before" and "after" networks look quite similar, since the driver is doing a similar activity (driving straight) on both. The "during" network looks much different, however. The most

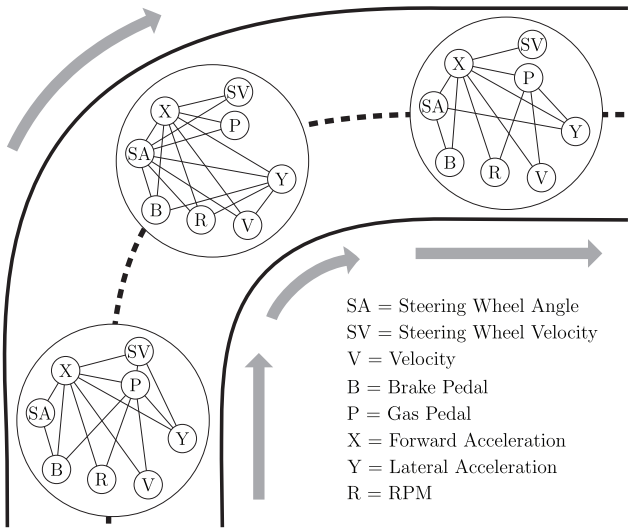


Figure 7: Three snapshots of the automobile sensor network measuring eight sensors, taken (1) before, (2) during, and (3) after a standard right turn.

noticeable difference is that the steering wheel angle sensor (SA) is now centrally located within the network, connected to almost every other node. This means that variations in the other sensors' observations can be largely explained by the steering wheel angle during the turn. For straightaways, on the other hand, the steering angle is on the network periphery, meaning that it is not well suited to explain the other sensor readings.

8 CONCLUSION AND FUTURE WORK

In this paper, we have defined a general method for inferring dynamic networks from timestamped observational data. Our approach, the time-varying graphical lasso, provides a scalable algorithm capable of encoding many different temporal dependencies. This type of modeling allows us to infer structure from large-scale "sensor" deployments. We leave for future work the examination of additional penalty functions to enforce different behaviors in the evolution of the network structure, along with their closed-form proximal operator solutions. There are also new extensions, beyond what was discussed in Section 3, which could be further analyzed. For example, we currently assume that the underlying distribution is zero mean. However, we could model a problem where observations come from a distribution $x \sim \mathcal{N}(\mu(t), \Sigma(t))$, and attempt to simultaneously estimate both the mean and covariance as they vary over time. Finally, this work could also be extended to infer correlations across different timestamps. Currently, our algorithm is suited to find simultaneously-related entities, *i.e.*, stocks whose prices move up and down in unison. However, there are other applications where the correlations are not immediate. For example, in brain connectivity networks, neuron A firing at timestamp i could cause neuron B to fire at time $i + 1$. Each of these additions would open up our framework to new potential applications, providing additional benefits to future research in this topic.

Acknowledgements. This work was supported by NSF IIS-1149837, NIH BD2K, DARPA SIMPLEX, DARPA XDATA, Chan Zuckerberg Biohub, SDSI, Boeing, Bosch, and Volkswagen.

REFERENCES

[1] A. Ahmed and E. P. Xing. Recovering time-varying networks of dependencies in social and biological studies. *PNAS*, 2009.
[2] O. Banerjee, L. El Ghaoui, and A. d'Aspremont. Model selection through sparse maximum likelihood estimation for multivariate Gaussian or binary data. *JMLR*, 2008.
[3] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine learning*, 2011.
[4] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.
[5] J. Dahl and L. Vandenberghe. CVXOPT: A Python package for convex optimization. In *Proc. Eur. Conf. Op. Res*, 2006.
[6] P. Danaher, P. Wang, and D. Witten. The joint graphical lasso for inverse covariance estimation across multiple classes. *JRSS: Series B*, 2014.
[7] J. Friedman, T. Hastie, and R. Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 2008.
[8] M. Gomez Rodriguez, J. Leskovec, and A. Krause. Inferring networks of diffusion and influence. In *KDD*. ACM, 2010.
[9] M. S. Grewal. *Kalman filtering*. Springer, 2011.
[10] D. Hallac, J. Leskovec, and S. Boyd. Network lasso: Clustering and optimization in large graphs. In *KDD*, 2015.
[11] D. Hallac, C. Wong, S. Diamond, R. Sosič, S. Boyd, and J. Leskovec. SnapVX: A network-based convex optimization solver. *JMLR (To Appear)*, 2017.
[12] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2009.
[13] M. R. Hestenes. Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, 1969.
[14] C.-J. Hsieh, M. A. Sustik, I. S. Dhillon, P. K. Ravikumar, and R. Poldrack. BIG & QUIC: Sparse inverse covariance estimation for a million variables. In *NIPS*, 2013.
[15] M. Kolar, L. Song, A. Ahmed, and E. Xing. Estimating time-varying networks. *The Annals of Applied Statistics*, 2010.
[16] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT press, 2009.
[17] S. L. Lauritzen. *Graphical models*. Clarendon Press, 1996.
[18] K. Mohan, M. Chung, S. Han, D. Witten, S.-I. Lee, and M. Fazel. Structured learning of Gaussian graphical models. In *NIPS*, 2012.
[19] K. Mohan, P. London, M. Fazel, D. Witten, and S.-I. Lee. Node-based learning of multiple Gaussian graphical models. *JMLR*, 2014.
[20] P. C. Molenaar. A dynamic factor model for the analysis of multivariate time series. *Psychometrika*, 1985.
[21] R. P. Monti, P. Hellyer, D. Sharp, R. Leech, C. Anagnostopoulos, and G. Montana. Estimating time-varying brain connectivity networks from functional MRI time series. *Neuroimage*, 2014.
[22] S. Myers and J. Leskovec. On the convexity of latent social network inference. In *NIPS*, 2010.
[23] A. Namaki, A. Shirazi, R. Raei, and G. Jafari. Network analysis of a financial market based on genuine correlation and threshold method. *Physica A: Stat. Mech. Apps.*, 2011.
[24] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 2016.
[25] N. Parikh and S. Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 2014.
[26] H. Rue and L. Held. *Gaussian Markov Random Fields: Theory and Applications*. CRC Press, 2005.
[27] K. Scheinberg, S. Ma, and D. Goldfarb. Sparse inverse covariance selection via alternating linearization methods. In *NIPS*, 2010.
[28] L. Vanderberghe. Proximal mapping lecture notes. http://seas.ucla.edu/~vandenbe/236C/lectures/proxop.pdf, 2010.
[29] M. J. Wainwright and M. I. Jordan. Log-determinant relaxation for approximate inference in discrete Markov random fields. *IEEE Tr. on Signal Processing*, 2006.
[30] K. Weinberger, F. Sha, Q. Zhu, and L. Saul. Graph Laplacian regularization for large-scale semidefinite programming. In *NIPS*, 2006.
[31] E. Wit and A. Abbruzzo. Inferring slowly-changing dynamic gene-regulatory networks. *BMC Bioinformatics*, 2015.
[32] D. Witten and R. Tibshirani. Covariance-regularized regression and classification for high dimensional problems. *JRSS: Series B*, 2009.
[33] M. Wytock and J. Z. Kolter. Sparse Gaussian conditional random fields: Algorithms, theory, and application to energy forecasting. *ICML*, 2013.
[34] S. Yang, Z. Lu, X. Shen, P. Wonka, and J. Ye. Fused multiple graphical lasso. *SIAM*, 2015.
[35] M. Yuan and Y. Lin. Model selection and estimation in the Gaussian graphical model. *Biometrika*, 2007.
[36] S. Zhou, J. Lafferty, and L. Wasserman. Time varying undirected graphs. *Machine Learning*, 2010.