Network Lasso: Clustering and Optimization in Large Graphs

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

ABSTRACT

Convex optimization is an essential tool for modern data analysis, as it provides a framework to formulate and solve many problems in machine learning and data mining. However, general convex optimization solvers do not scale well, and scalable solvers are often specialized to only work on a narrow class of problems. Therefore, there is a need for simple, scalable algorithms that can solve many common optimization problems. In this paper, we introduce the *network lasso*, a generalization of the group lasso to a network setting that allows for simultaneous clustering and optimization on graphs. We develop an algorithm based on the Alternating Direction Method of Multipliers (ADMM) to solve this problem in a distributed and scalable manner, which allows for guaranteed global convergence even on large graphs. We also examine a non-convex extension of this approach. We then demonstrate that many types of problems can be expressed in our framework. We focus on three in particular — binary classification, predicting housing prices, and event detection in time series data — comparing the network lasso to baseline approaches and showing that it is both a fast and accurate method of solving large optimization problems.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database applications—*Data mining*
General Terms: Algorithms; Experimentation.
Keywords: Convex Optimization, ADMM, Network Lasso.

1. INTRODUCTION

Convex optimization has become an increasingly popular way of modeling problems in many different fields, ranging from finance [4, §4.4] to image processing [5]. However, as datasets get larger and more intricate, classical methods of convex analysis, which often rely on interior point methods, begin to fail due to a lack of scalability. In fact, without any known structure to the optimization problem, the convergence time will scale with the cube of the problem size [4]. The challenge of large-scale optimization lies in developing methods general enough to work well independent of the input and capable of scaling to the immense datasets that today's applications require. Presently, solving these problems in a scalable way requires developing problem-specific solvers to exploit structure in the model [27], often an infeasible assumption. Therefore, it is necessary to formulate general classes of optimization solvers that can apply to a variety of relevant problems, and to develop algorithms for obtaining reliable and efficient solutions.

Present Work: Formulation. Here, we focus on optimization problems posed on graphs. Consider the following problem on a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the vertex set and $\mathcal{E}$ the set of edges:

$$\text{minimize} \quad \sum_{i \in \mathcal{V}} f_i(x_i) + \sum_{(j,k) \in \mathcal{E}} g_{jk}(x_j, x_k). \tag{1}$$

The variables are $x_1, \ldots, x_m \in \mathbf{R}^p$, where $m = |\mathcal{V}|$. (The total number of scalar variables is mp.) Here $x_i \in \mathbf{R}^p$ is the variable at node i, $f_i : \mathbf{R}^p \to \mathbf{R} \cup \{\infty\}$ is the cost function at node i, and $g_{jk} : \mathbf{R}^p \times \mathbf{R}^p \to \mathbf{R} \cup \{\infty\}$ is the cost function associated with edge (j, k). We use extended (infinite) values of f_i and g_{jk} to describe constraints on the variables, or pairs of variables across an edge, respectively. Our focus will be on the special case in which the f_i are convex, and $g_{jk}(x_j, x_k) = \lambda w_{jk} \|x_j - x_k\|_2$, with $\lambda \geq 0$ and user-defined weights $w_{jk} \geq 0$:

$$\text{minimize} \quad \sum_{i \in \mathcal{V}} f_i(x_i) + \lambda \sum_{(j,k) \in \mathcal{E}} w_{jk} \|x_j - x_k\|_2. \tag{2}$$

The edge objectives penalize differences between the variables at adjacent nodes, where the edge between nodes i and j has weight λw_{ij}. Here we can think of w_{ij} as setting the relative weights among the edges of the network, and λ as an overall parameter that scales the edge objectives relative to the node objectives. We call problem (2) the *network lasso* problem, since the edge cost is a sum of norms of differences of the adjacent edge variables.

The network lasso problem is a convex optimization problem, and so in principle it can be solved efficiently. For small networks, generic (centralized) convex optimization methods can be used to solve it. But we are interested in problems with many variables, with p, $m = |\mathcal{V}|$, and $n = |\mathcal{E}|$ all potentially large. For such problems no adequate solver currently exists. Thus, we develop a distributed and scalable method for solving the network lasso problem, in which each vertex variable x_i is controlled by one "agent", and the agents exchange (small) messages over the graph to solve the problem iteratively. This approach provides global convergence for all problems that can be put into this form. We also analyze a non-convex extension of the network lasso, a slightly different way to model the problem, and give a similar algorithm that, although it does not guarantee optimality, tends to perform well in practice.

Present Work: Applications. There are many general settings in which the network lasso problem arises. In control systems, the nodes might represent the possible states of a system, and x_i the action or actions to take when we are in state i, so the collection of variables $(x_1, \ldots, x_m)$ describes a policy. The graph tells us about state transitions, and the weights express how much we care about the actions in neighboring states differing. Here the network lasso problem seeks a solution that minimizes the total cost, but also does not change much across adjacent states, allowing for a "simpler" policy. The parameter λ allows us to trade off the total cost (the node objective) versus a cost for the actions varying across the states (the edge objective).

Another general setting, one we focus on in this paper, relates to statistical learning, where the variables x_i are parameters in a statistical model of some data resident at, or associated with, node i. The objective term f_i represents the loss for the model over the data, possibly with some regularization added in. The edge terms are regularization that encourages adjacent nodes to have close (or the same) model parameters. In this setting, the network expresses our idea that adjacent nodes should have similar (or the same) models. We can imagine that this regularization allows us to build models at each node that borrow strength from the fact that neighboring nodes should have similar, or even identical, models.

It is critical to note that the edge terms in the network lasso problem involve the norm, not the norm squared, of the difference. If the norms were squared, the edge objective would reduce to (weighted) Laplacian regularization [25]. The sum-of-norms regularization that we use is like group lasso [28]; it encourages not just $x_i \approx x_j$ for edge $(i, j) \in \mathcal{E}$, but $x_i = x_j$, *i.e.*, *consensus* across the edge. Indeed, we will see that there is often a (finite) value of λ above which the solution has $x_1 = \cdots = x_m$, *i.e.*, all the vectors are in consensus. For smaller values of λ, the solution of the network lasso problem breaks into clusters of nodes, with x_i the same across all nodes in the cluster. In the policy setting, we can think of this as a combination of state aggregation or clustering, together with policy design. In the modeling setting, this is a combination of clustering the data collections and fitting a model to each cluster.

Present Work: Use Case. As a running example, which we later analyze in detail, consider the problem of predicting housing prices. One common approach is linear regression. That is, we learn the weights of each feature (number of bedrooms, square footage, etc...) and use these same weights for each house to estimate the price. However, due to location-based factors such as school district or distance to a highway, similar houses in different locations can have drastically different prices. These factors are often unknown a priori and difficult to quantify, so it is inconvenient to attempt to incorporate them as features in the regression. Therefore, standard linear regression will have large errors in price prediction, since it forces the entire dataset to agree on a single global model. What we actually want is to cluster the houses into "neighborhoods" which share a common regression model. First, we build a network where neighboring houses (nodes) are connected by edges. Then, each house solves for its own regression model (based on its own features and price). We use the network lasso penalty to encourage nearby houses to share the same regression parameters, in essence helping each house determine which neighborhood it is part of, and learning relevant information from this group of neighbors to improve its own prediction. The size and shape of these neighborhoods, though, are difficult to know beforehand and often depend on a variety of factors, including the amount of available data. The network lasso solution empirically determines the neighborhoods, so that each house can share a common model with houses in its cluster, without having to agree with the potentially misleading information from other locations.

Summary of Contributions. The main contributions of this paper are as follows:

- We formally define the *network lasso*, a specific type of optimization problem on networks.
- We develop a fast, scalable, and distributed solver for any problem of this form. This algorithm is also capable of choosing the right regularization parameter λ.
- We show that many common and useful problems can be formulated as an instance of the network lasso.

Related Work. The network lasso can be thought of as a special case of certain methods (Bayesian inference, general convex optimization) and a generalization of others (fused lasso [23], total variation [24, 26]). It occupies a unique point on the trade-off curve between generality and scalability that, to the best of our knowledge, has not yet been formally analyzed. Our approach provides a unified view of a diverse class of optimization problems, but is still capable of solving large-scale examples. For example, convex clustering [7, 14, 22], an alternative to the K-means algorithm, is a well-studied instance of the network lasso. However, convex clustering requires f_i to be the square loss from some observation a_i, and often assumes a fully connected graph since there is no prior information about which nodes may be clustered together. In contrast, generalizing to any shape of network with any convex objectives (including allowing constraints) allows our approach to be applied to new topics, such as control systems and event detection. Furthermore, we elect to focus on the ℓ_2-norm because of its intuitive network-based rationale in that it leads to node stratification.

The network lasso is also related to probabilistic graphical models (PGMs). Problem (2) is a type of Bayesian inference where we learn a set of models or dependencies based on latent clustering. The network lasso penalty, a form of regularization, allows for one type of "relationship" between nodes, a weighted prior belief that the connected variables should be equal. The clustering that our model accomplishes is similar to a latent variable mixture model [20], where cluster membership is indicated by some latent variable. With this, certain network lasso problems can be rewritten as a maximum likelihood estimation problem where a conditional distribution is learned for each cluster. However, many examples are difficult to encode and scale in this way. Additionally, there has been much research on optimal decomposition and splitting methods for these types of problems [8, 19]. Hinge-loss Markov random fields, which are PGMs defined over continuous variables for MAP inference, use a similar ADMM-based approach to ours [1], though the hinge-loss potentials they support do not include the norm-based lasso that we utilize to induce the clustering. However, unlike many of these other frameworks [1, 16, 29], which often use a probabilistic approach, we formulate it as a single, very large, convex optimization problem that we solve by splitting it across a graph. This focus on the specific topic of simultaneous clustering and optimization enables us to provide a clean formalism and scalable approach, with guaranteed convergence, for solving a wide class of problems, all using the exact same algorithm.

2. CONVEX PROBLEM DEFINITION

We now look more closely at the network lasso problem,

$$\text{minimize} \quad \sum_{i \in \mathcal{V}} f_i(x_i) + \lambda \sum_{(j,k) \in \mathcal{E}} w_{jk} \|x_j - x_k\|_2.$$

This problem is convex in the variable $x = (x_1, \ldots, x_m) \in \mathbf{R}^{mp}$, and we let $x^\star$ denote an optimal solution.

Local Variables. It is worth noting that there can be local private optimization variables at each node that are not part of the lasso penalty. More formally, the network lasso problem can be defined as

$$\text{minimize} \quad \sum_{i \in \mathcal{V}} \tilde{f}_i(x_i, \varepsilon_i) + \lambda \sum_{(j,k) \in \mathcal{E}} w_{jk} \|x_j - x_k\|_2, \tag{3}$$

where ε_i are potential dummy variables at node i (the size can vary at each node). However, using partial minimization, if we let

$$f_i(x_i) = \min_{\varepsilon_i} \tilde{f}_i(x_i, \varepsilon_i),$$

we get the original problem, defined in (2). For simplicity, we therefore use problem (2) throughout the paper, with the implicit understanding that there may be private variables at each node.

Regularization Path. Although the regularization parameter λ in problem (2) can be incorporated into the w_{ij}'s by scaling the edge weights, it is best viewed separately as a single parameter which is tuned to yield different global results. λ defines a trade-off for the nodes between minimizing its own objective and agreeing with its neighbors. At $\lambda = 0$, $x_i^\star$, the solution at node i, is simply a minimizer of f_i. This can be computed locally at each node, since when $\lambda = 0$ the edges of the network have no effect. At the other extreme, as $\lambda \to \infty$, problem (2) turns into

$$\text{minimize} \quad \sum_{i \in \mathcal{V}} f_i(\tilde{x}), \tag{4}$$

since a common $\tilde{x}$ must be the solution at every node. This is solved by $x^{\text{cons}} \in \mathbf{R}^p$. We refer to (4) as the *consensus problem* and to x^{cons} as the *consensus solution*. If a solution to (4) exists, it can be shown that there is a finite $\lambda_{\text{critical}}$ such that for any $\lambda \geq \lambda_{\text{critical}}$, the consensus solution holds. That is, beyond this $\lambda_{\text{critical}}$, increasing λ has no effect on the solution. For λ's in between $\lambda = 0$ and $\lambda_{\text{critical}}$, the family of solutions is known as the *regularization path*, though it is sometimes referred to as the clusterpath [14].

Network Lasso and Clustering. The ℓ_2-norm penalty over the edge difference, $\|x_j - x_k\|_2$, defines the *network lasso*. It incentivizes the differences between connected nodes to be exactly zero, rather than just close to zero, yet it does not penalize large outliers (in this case, node values being very different) too severely. An edge difference of zero means that $x_j = x_k$. When many edges are in consensus like this, we have grouped the nodes into sets with equal values of x_i. Each set of nodes, or cluster, has a common solution for the variable x_i. The outliers then refer to edges between nodes in different clusters. Cluster size tends to get larger as λ increases, until at $\lambda_{\text{critical}}$ the consensus solution can be thought of as a single cluster for the entire network. Even though increasing λ is most often agglomerative, cluster fission may occur, meaning two nodes in the same cluster may break apart at a higher λ. Therefore, the clustering pattern is not strictly hierarchical [22].

Inference on New Nodes. After we have solved for $x^\star$, we can interpolate the solution to estimate the value of x_j on a new node j, for example during cross-validation on a test set. Given j, all we need is its location within the network; that is, the neighbors of j and the edge weights. With this information, we treat j like a dummy node, with $f_j(x_j) = 0$. We solve for x_j just like in problem (2) except without the objective function f_j, so the optimization problem becomes

$$\text{minimize} \quad \sum_{k \in N(j)} w_{jk} \|x_j - x_k^\star\|_2, \tag{5}$$

where $N(j)$ is the set of neighbors of node j. This estimate of x_j can be thought of as a weighted median of j's neighbors' solutions. This is called the Weber problem, and it involves finding the point which minimizes the weighted sum of distances to a set of other points [2]. It has no analytical solution when j has more than two neighbors, but it can be readily computed even for large problems. For example, when one of the dimensions is much larger than the other (number of neighbors vs. size of each x_k), the problem can be solved in linear time with respect to the larger dimension [4].

3. PROPOSED SOLUTION

On smaller graphs, the network lasso problem can be solved using standard interior point methods. This paper focuses on large problems, where solving everything at once is infeasible. This is especially true when we solve for a span of λ's across the entire regularization path, since we will need to solve a separate problem for each λ. A distributed solution is necessary so that computational and storage limits do not constrain the scope of potential applications. We propose an easy-to-implement algorithm based on the Alternating Direction Method of Multipliers (ADMM) [3, 21], a well-established method for distributed convex optimization. With ADMM, each individual component solves its own private objective function, passes this solution to its neighbors, and repeats the process until the entire network converges. There is no need for global coordination except for iteration synchronization.

3.1 ADMM

To solve via ADMM, we introduce a copy of x_i, called z_{ij}, at each edge ij. Note that the same edge also has a z_{ji}, a copy of x_j. We rewrite problem (2) as an equivalent problem,

$$\begin{array}{ll} \text{minimize} & \sum_{i \in \mathcal{V}} f_i(x_i) + \lambda \sum_{(j,k) \in \mathcal{E}} w_{jk} \|z_{jk} - z_{kj}\|_2 \\ \text{subject to} & x_i = z_{ij}, \quad i = 1, \ldots, m, \quad j \in N(i). \end{array}$$

We then derive its augmented Lagrangian [13], which gives us

$$\begin{aligned} L_\rho(x, z, u) = & \sum_{i \in \mathcal{V}} f_i(x_i) + \sum_{(j,k) \in \mathcal{E}} \Big(\lambda w_{jk} \|z_{jk} - z_{kj}\|_2 - \\ & (\rho/2) \left(\|u_{jk}\|_2^2 + \|u_{kj}\|_2^2 \right) + \\ & (\rho/2) \left(\|x_j - z_{jk} + u_{jk}\|_2^2 + \|x_k - z_{kj} + u_{kj}\|_2^2 \right) \Big), \end{aligned}$$

where u is the scaled dual variable and $\rho > 0$ is the penalty parameter [3, §3.1.1]. ADMM consists of the following steps, with k denoting the iteration number:

$$\begin{aligned} x^{k+1} &= \underset{x}{\operatorname{argmin}}\, L_\rho(x, z^k, u^k) \\ z^{k+1} &= \underset{z}{\operatorname{argmin}}\, L_\rho(x^{k+1}, z, u^k) \\ u^{k+1} &= u^k + (x^{k+1} - z^{k+1}). \end{aligned}$$

Let us examine each of these steps in more detail.

x-Update. In the x-update we minimize a separable sum of functions, one per node, so it can be calculated independently at each node and solved in parallel. At node i, this is

$$x_i^{k+1} = \underset{x_i}{\operatorname{argmin}} \left(f_i(x_i) + \sum_{j \in N(i)} (\rho/2) \|x_i - z_{ij}^k + u_{ij}^k\|_2^2 \right).$$

z-Update. The z-update is separable across the edges. Note that for edge ij, we need to jointly update z_{ij} and z_{ji}. This becomes

$$\begin{aligned} z_{ij}^{k+1}, z_{ji}^{k+1} = & \underset{z_{ij}, z_{ji}}{\operatorname{argmin}} \Big(\lambda w_{ij} \|z_{ij} - z_{ji}\|_2 + \\ & (\rho/2) \left(\|x_i^{k+1} - z_{ij} + u_{ij}^k\|_2^2 + \|x_j^{k+1} - z_{ji} + u_{ji}^k\|_2^2 \right) \Big). \end{aligned}$$

This problem has a closed-form analytical solution, which we derive in Appendix A. It is

$$z_{ij}^\star = \theta(x_i + u_{ij}) + (1-\theta)(x_j + u_{ji})$$
$$z_{ji}^\star = (1-\theta)(x_i + u_{ij}) + \theta(x_j + u_{ji}),$$

where

$$\theta = \max\left(1 - \frac{\lambda w_{ij}}{\rho \|x_i + u_{ij} - (x_j + u_{ji})\|_2}, 0.5\right). \quad (6)$$

u-Update. The u-update is also edge-separable. For each variable, this looks like

$$u_{ij}^{k+1} = u_{ij}^k + (x_i^{k+1} - z_{ij}^{k+1}).$$

Global Convergence. Because the problem is convex, ADMM is guaranteed to converge to a global optimum. The stopping criterion can be based on the primal and dual residuals, commonly defined as r and s, being below given threshold values; see [3]. This allows us to stop when x^k and z^k are close, and when x^k (or z^k) does not change much in one iteration. As is typical for ADMM, the algorithm tends to attain modest accuracy relatively quickly, and high accuracy (which in many applications is not needed) only slowly.

Algorithm 1 ADMM Steps

```
**repeat**
    $x_i^{k+1} = \underset{x_i}{\text{argmin}}\left(f_i(x_i) + \sum_{j \in N(i)} (\rho/2)\|x_i - z_{ij}^k + u_{ij}^k\|_2^2\right)$
    $z_{ij}^{k+1} = \theta(x_i + u_{ij}) + (1-\theta)(x_j + u_{ji})$
    $z_{ji}^{k+1} = (1-\theta)(x_i + u_{ij}) + \theta(x_j + u_{ji})$
    $u_{ij}^{k+1} = u_{ij}^k + (x_i^{k+1} - z_{ij}^{k+1})$
**until** $\|r^k\|_2 \le \epsilon^{\text{pri}}$; $\|s^k\|_2 \le \epsilon^{\text{dual}}$.
```

3.2 Regularization Path

It is often useful to compute the regularization path as a function of λ to gain insight into the network structure. For specific applications, this may also help decide the correct value of λ to use, for example by choosing λ to minimize the cross-validation error.

We begin the regularization path at $\lambda = 0$ and solve for an increasing sequence of λ's ($\lambda := \alpha\lambda$, $\alpha > 1$). We know when we have reached $\lambda_{\text{critical}}$ because a single x^{cons} will be the optimal solution at every node, and increasing λ no longer affects the solution. This may lead to a stopping point slightly above the actual $\lambda_{\text{critical}}$, which we denote as $\tilde{\lambda}_{\text{critical}}$. There is no harm if $\tilde{\lambda}_{\text{critical}} > \lambda_{\text{critical}}$, since they will both yield the same result, the consensus solution. To account for the case where no consensus solution exists, we can also stop when the new solution has changed by less than some ϵ, since even without consensus, the problem converges to some solution.

A big advantage of the regularization path, as opposed to computing each value of $x^\star(\lambda)$ in parallel, is that we begin with a warm start towards the new solution at each step. For each λ, the unknown variables are already close to the new $x^\star$, $u^\star$, and $z^\star$ by virtue of starting at the solution for the last λ. In fact, when f_i is strictly convex, the solution $x^\star$ is continuous in λ. Without any prior knowledge, for example initializing everything to 0 for each λ, we start far from the actual solution, so it will often (although not always) take longer to converge via ADMM. The only other required variable is λ_{initial}, the initial non-zero value of λ, which depends on the variable scaling. The hope is that $x^\star$ does not change too much between $\lambda = 0$ and this initial value, and a rough estimate of λ_{initial} can be found using the following heuristic:

1. Pick edge ij at random and find $x_i^\star$, $x_j^\star$ at $\lambda = 0$.
2. Evaluate the gradients of $f_i(x)$ and $f_j(x)$ at $x = (x_i^\star + x_j^\star)/2$.
3. Set $\lambda_{\text{initial}} := 0.01\left(\frac{\|\nabla f_i(x)\|_2 + \|\nabla f_j(x)\|_2}{2w_{ij}}\right)$.

To get a more robust estimate, repeat the above steps picking different edges each time, and choose the smallest solution for λ_{initial}. Given these variables, we are now able to solve for the entire regularization path. Our method is outlined in Algorithm 2.

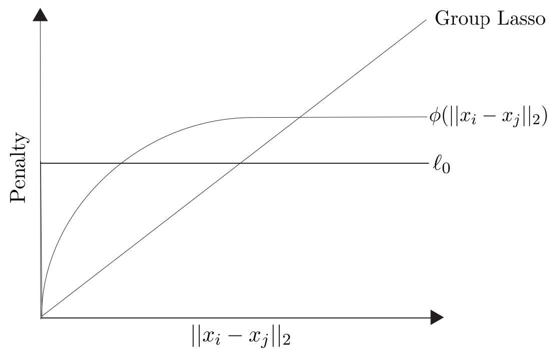


Figure 1: Comparison of Group Lasso, ℓ_0, and Non-Convex ϕ.

Algorithm 2 Regularization Path

```
**initialize** Solve for $x^\star$, $u^\star$, $z^\star$ at $\lambda = 0$.
**set** $\lambda := \lambda_{\text{initial}}$; $\alpha > 1$; $u := u^\star$; $z := z^\star$.
**repeat**
    Use ADMM to solve for $x^\star(\lambda)$ (see Algorithm 1)
    *Stopping Criterion.* **quit** if $x^\star(\lambda) = x^\star(\lambda_{\text{previous}})$
    Set $\lambda := \alpha\lambda$.
**return** $x^\star(\lambda)$ for $\lambda$ from 0 to $\tilde{\lambda}_{\text{critical}}$.
```

4. NON-CONVEX EXTENSION

In many applications, we are using the group lasso as an approximation of the ℓ_0-norm [6]. That is, we are looking for a sparse solution where relatively few edge differences are non-zero. However, once $\|x_i - x_j\|_2$ becomes non-zero, we do not care about its magnitude, since we already know that i and j are in different clusters. The lasso has a proportional penalty, which is the closest that a convex function can come to approximating the ℓ_0-norm. Once we have found the true clusters, though, this will "pull" the different clusters towards each other through their mutual edges. If we replace the group lasso penalty with a monotonically nondecreasing concave function $\phi(u)$, where $\phi(0) = 0$ and whose domain is $u \ge 0$, we come even closer to the ℓ_0, as shown in Figure 1. However, this new optimization problem,

$$\text{minimize} \quad \sum_{i \in \mathcal{V}} f_i(x_i) + \lambda \sum_{(j,k) \in \mathcal{E}} w_{jk}\phi\left(\|x_j - x_k\|_2\right), \quad (7)$$

is not convex. ADMM is not guaranteed to converge, and even if it does, it need not be to a global optimum. It is in some sense a "riskier" approach. In fact, different initial conditions on x, u, z, and ρ can yield quite different solutions. However, as a heuristic, a slight modification to ADMM empirically performs very well. Since the algorithm might not converge, it is necessary to keep track of the iteration which yields the minimum objective, and to return that as the solution instead of the most recent step. The primal and dual residuals are not guaranteed to go to 0, so we instead run our algorithm for a set number of iterations for each λ.

Non-Convex z-Update. Compared to the convex case, the only difference in the ADMM solution is the z-update, which is now

$$\begin{array}{ll}\text{minimize} & \lambda w_{ij}\phi\left(\|z_{ij}-z_{ji}\|_2\right)+(\rho/2)\big(\|x_i^{k+1}-z_{ij}+u_{ij}^k\|_2^2+\\ & \|x_j^{k+1}-z_{ji}+u_{ji}^k\|_2^2\big).\end{array} \tag{8}$$

For simplicity, we define

$$a=x_i^{k+1}+u_{ij}^k,\quad b=x_j^{k+1}+u_{ji}^k,$$
$$c=\lambda w_{ij},\quad d=\|a-b\|_2,$$

so problem (8) turns into

$$\text{minimize}\quad c\phi\left(\|z_{ij}-z_{ji}\|_2\right)+(\rho/2)\left(\|a-z_{ij}\|_2^2+\|b-z_{ji}\|_2^2\right).$$

There are two possible cases for the solution to problem (8): $z_{ij}^\star=z_{ji}^\star$ or $z_{ij}^\star\neq z_{ji}^\star$. When the two solutions are identical, then $\phi\left(\|z_{ij}-z_{ji}\|_2\right)=\phi(0)=0$, so the only terms remaining are

$$(\rho/2)\left(\|a-z_{ij}\|_2^2+\|b-z_{ji}\|_2^2\right).$$

Minimizing over the constraint that $z_{ij}=z_{ji}$ yields $z_{ij}^\star=z_{ji}^\star=(1/2)(a+b)$ and an objective of $(\rho/4)\|a-b\|_2^2$.

When the two solutions are not equal, $z_{ij}^\star$ and $z_{ji}^\star$ must lie on the line segment between a and b. If $z_{ij}^\star$ and/or $z_{ji}^\star$ are not on the line segment, projecting them onto this segment is nonincreasing in $\phi\left(\|z_{ij}-z_{ji}\|_2\right)$ and decreasing in $(\rho/2)\left(\|a-z_{ij}\|_2^2+\|b-z_{ji}\|_2^2\right)$, so the total objective function is guaranteed to decrease. Therefore, we know that

$$z_{ij}^\star=\theta_1 a+(1-\theta_1)b,\quad \theta_1\in[0,1]$$
$$z_{ji}^\star=\theta_2 a+(1-\theta_2)b,\quad \theta_2\in[0,1]$$

and that

$$\|z_{ij}^\star-z_{ji}^\star\|_2=\|a-b\|_2\left(|\theta_1-\theta_2|\right)=d|\theta_1-\theta_2|.$$

Note that the solution for $z_{ij}^\star=z_{ji}^\star$ is just $\theta_1=\theta_2=\frac{1}{2}$. We also know that $\theta_1\geq\theta_2$. If $\theta_1<\theta_2$, we could swap θ_1 and θ_2 and $\phi\left(\|z_{ij}-z_{ji}\|_2\right)$ would remain constant, but the rest of the objective, $(\rho/2)\left(\|a-z_{ij}\|_2^2+\|b-z_{ji}\|_2^2\right)$, would decrease. Therefore, we can rewrite the norm of the difference as

$$\|z_{ij}^\star-z_{ji}^\star\|_2=d(\theta_1-\theta_2),$$

and the objective becomes

$$c\phi\left(d(\theta_1-\theta_2)\right)+(\rho d^2/2)\left((1-\theta_1)^2+\theta_2^2\right).$$

When $z_{ij}^\star\neq z_{ji}^\star$, we know that $\theta_1>\theta_2$, and thus $d(\theta_1-\theta_2)>0$. When ϕ is differentiable at $d(\theta_1-\theta_2)$, we set the gradient to zero:

$$\frac{\partial}{\partial\theta_1}=cd\phi'(d(\theta_1-\theta_2))-\rho d^2(1-\theta_1)=0$$
$$\frac{\partial}{\partial\theta_2}=-cd\phi'(d(\theta_1-\theta_2))+\rho d^2\theta_2=0.$$

We see that

$$\rho d^2(1-\theta_1)=cd\phi'(d(\theta_1-\theta_2))=\rho d^2\theta_2,$$

or

$$\theta_2=1-\theta_1.$$

This puts the entire optimization problem in terms of one variable, $\theta=\theta_2$. Since $\theta_1+\theta_2=1$ and $\theta_1\geq\theta_2$, we know that $\theta\leq\frac{1}{2}$, so the final problem becomes

$$\begin{array}{ll}\text{minimize} & c\phi\left(d(1-2\theta)\right)+\rho d^2\theta^2\\ \text{subject to} & 0\leq\theta\leq\frac{1}{2}.\end{array} \tag{9}$$

It is of course necessary to find all solutions to this problem, since there may be several or none, and to compare the resulting objective to $(\rho/4)\|a-b\|_2^2$, when $z_{ij}^\star=z_{ji}^\star$. Of these solutions, pick the z's which minimize the overall objective function.

Log Function. We will now look at the specific case where $\phi(u)=\log(1+\frac{u}{\epsilon})$, where ϵ is a constant scaling factor. The objective function in problem (9) turns into

$$\text{minimize}\quad c\log(1+\tfrac{d(1-2\theta)}{\epsilon})+\rho d^2\theta^2.$$

Setting the derivative equal to zero, we get

$$-\frac{2cd}{d-2d\theta+\epsilon}+2\rho d^2\theta=0.$$

We simplify to

$$2\rho d^2\theta^2-\rho d(d+\epsilon)\theta+c=0$$

and see that this is a simple quadratic equation in θ, solved by

$$\theta=\frac{\rho(d+\epsilon)\pm\sqrt{\rho^2(d+\epsilon)^2-8\rho c}}{4\rho d}.$$

The z-update then involves comparing the resulting objectives with $(\rho/4)\|a-b\|_2^2$ (the value when $z_{ij}^\star=z_{ji}^\star$) and then choosing the θ which yields the best of the three objectives to obtain $z_{ij}^\star$, $z_{ji}^\star$. If the quadratic term has no real roots, which happens more frequently as λ increases, we set $\theta=\frac{1}{2}$, meaning the edge is in consensus. It is worth reiterating that this method is not guaranteed to reach the global optimum. Instead, it is an easy-to-implement algorithm that parallels ADMM from the convex case.

5. EXPERIMENTS

We now apply our approach on three examples to illustrate the diverse set of problems that fall under the network lasso framework, and to provide a simple and unified view of these seemingly different applications. First, we look at a synthetic example in which we gather statistical power from the network to improve classification accuracy. Next, we see how our approach can apply to a geographic network, allowing us to gain insights on residential neighborhoods by predicting housing prices. Finally, we look at a time series dataset for the purpose of detecting outliers, or anomalous events, in the temporal data. To run these experiments, we built a module combining Snap.py [17] and CVXPY [10]. The network is stored as a Snap.py structure, and the x-updates of ADMM are run in parallel using CVXPY. Even though this algorithm is capable of being distributed across many machines, we instead distribute it across multiple cores of a single machine for our prototype. Our network-based convex optimization solver is available at `http://snap.stanford.edu/snapvx`, and the code for this paper can be found on the SnapVX website.

5.1 Network-Enhanced Classification

We first analyze a synthetic network in which each node has a support vector machine (SVM) classifier [9], but does not have enough training data to accurately estimate it. The clustering of the nodes in the network occurs because some of the nodes have common underlying SVMs. The hope is that nodes can, in essence, "borrow" training examples from their relevant neighbors to improve their own results. Of course, neighbors with different underlying models will provide misleading information to each other.

These are the edges whose lasso penalties should be non-zero, yielding different solutions at the two connected nodes.

Dataset. We randomly generate a dataset containing 1000 nodes, each with its own classifier, a support vector machine in $\mathbf{R}^{50}$. Given an input $w \in \mathbf{R}^{50}$, each node tries to predict $y \in \{-1, 1\}$, where

$$y = \text{sgn}(a_i^T w + a_{i,0} + v),$$

and $v \sim \mathcal{N}(0, 1)$, the noise, is independent for each data point. An SVM involves solving a convex optimization problem from a set of training examples to obtain $x_i = \begin{bmatrix} a_i^T & a_{i,0} \end{bmatrix}^T \in \mathbf{R}^{51}$. This defines a separating hyperplane to determine how to classify new inputs. There is no way to counter the noise v, but an accurate x_i can help us predict y from w reasonably accurately. Each node determines its own optimal classifier from a training set consisting of 25 (w, y)-pairs per node, which are used to solve for x. All elements in w, a, and v are drawn independently from a normal distribution, with the y values dependent on the other variables.

Network. The 1000 nodes are split into 20 equally-sized groups. Each group has a common underlying classifier, $\begin{bmatrix} a^T & a_0 \end{bmatrix}^T$, while different groups have independent a's. If i and j are in the same group, they have an edge with probability 0.5, and if they are in different groups, there is an edge with probability 0.01. Overall, this leads to a total of 17079 edges, with 28.12% of the edges connecting nodes in different underlying groups. Even though this is a synthetic example, there are a large number of misleading edges, and each node has only 25 examples to train an SVM in $\mathbf{R}^{50}$, so solving this problem is far from an easy task.

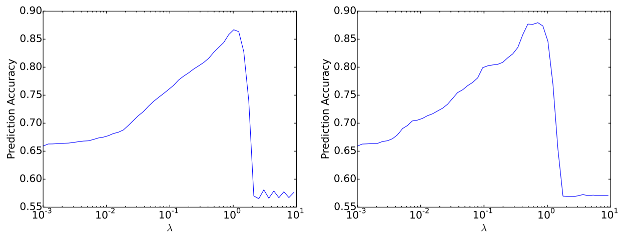
Optimization Parameter and Objective Function. At node i, the optimization parameter $x_i = \begin{bmatrix} x_{i,a}^T & x_{i,0} \end{bmatrix}^T = \begin{bmatrix} a_i^T & a_{i,0} \end{bmatrix}^T$ defines our estimate for the separating hyperplane for the SVM [12]. The node then solves its own optimization problem, using its 25 training examples. At each node, f_i is defined as

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\|x_{i,a}\|_2^2 + \sum_{i=1}^{25} c\|\varepsilon_i\|_1 \\ \text{subject to} & y^{(i)}(x_{i,a}^T w^{(i)} + x_{i,0}) \geq 1 - \varepsilon_i, \quad i = 1, \ldots, 25. \end{array}$$

The ε_i's are (local) slack variables. They allow points to be misclassified in the training set of a soft margin SVM [9]. We set c, the threshold parameter, to a constant which was empirically found to perform well on a common model. We solve for 51 + 25 = 76 variables at each node, so the total problem has 76,000 unknowns.

Results. To evaluate performance, we find prediction accuracy on a separate test set of 10,000 examples (10 per node). In Figure 2, we plot percentage of correct predictions vs. λ, where λ is displayed in log-scale, over the regularization path. Note that the two extremes of the path represent important baselines.

At $\lambda = 0$, each node only uses its own training examples, ignoring all the information provided by its neighbors. This is just a local SVM, with only 25 training examples to estimate a 51-dimensional vector. This leads to a prediction accuracy of 65.9% on the test set. When $\lambda \geq \lambda_{\text{critical}}$, the problem finds a common x, which is equivalent to solving a global SVM over the entire network. This assumes the entire graph is coupled together and does not allow for any edges to break. This common hyperplane at every node yields an accuracy of 57.1%, which is barely an improvement over random guessing. In contrast, both the convex and non-convex cases perform much better for λ's in the middle. From Figure 2, we see a distinct shape in the regularization paths. As λ increases, the accuracy steadily improves, until a peak near $\lambda = 1$. Intuitively, this represents the point where the algorithm has approximately split the



(a) Convex (b) Non-Convex

Figure 2: SVM regularization path.

Method	Maximum Prediction Accuracy
Local SVM ($\lambda = 0$)	65.90%
Global SVM ($\lambda \geq \lambda_{\text{critical}}$)	57.10%
Convex Network Lasso	86.68%
Non-Convex Network Lasso	87.94%

Table 1: SVM test set prediction accuracy.

nodes into their correct clusters, each with its own classifier. As λ continues to increase, there is a rapid drop off in performance, due to the different clusters "pulling" each other together. The maximum prediction accuracies on the test sets are 86.68% (convex) and 87.94% (non-convex). These prediction results are summarized in Table 1.

Timing Results. We compare our network lasso algorithm to a standard centralized method on a single 40-core CPU where the entire problem fits into memory. For the centralized case, we used the same solver (CVXPY) as in the x-updates for ADMM. While wrapped in a Python layer, CVXPY's underlying solver uses ECOS [11], an open-source software package specifically designed for high performance numerical optimization, so the Python overhead is negligible when it comes to the cost of scaling to large problems. We show the results on the synthetic SVM example to scale the problem size over several orders of magnitude. We solve the problem at 12 geometrically spaced values of λ to span the entire regularization path. We use $\frac{n}{20}$ underlying SVM clusters, where n is the number of nodes. The entire regularization path is one large problem (consisting of 12 smaller ones), and we measure its total runtime. Note that each node in this case is solving its own SVM, with additional coupling constraints due to the network lasso on the edges. We vary the total number of nodes, and the results are shown in Figure 3. We see that, in this example, the centralized method scales on the order of problem size cubed, whereas ADMM takes closer to linear time, until other concerns such as memory limitations begin to factor in. By the time there are 20,000 unknowns, ADMM is already 100 times faster, and this discrepancy in convergence time only grows as the problem gets larger.

To further test our algorithm, we also solve a larger yet simpler problem. We build a random 3-regular graph (every node has a degree of 3) with 2000 nodes. The objective function at each node is $f_i(x_i) = \|x_i - a_i\|_2^2$, where a_i is a random vector in $\mathbf{R}^q$. We can modify the value of q to vary the total number of unknowns. We pick a single (constant) λ in the middle of the regularization path and see how long it takes to solve the problem using ADMM. The results are shown in Table 2. We can compute a solution for 1 million unknowns in seconds, and for 100 million in under 15 minutes. It is worth reiterating that at each step, at each node, we use CVXPY rather than a more specialized solver for the x-update subproblem. This allows the same solver to work on any convex node objective, rather than being constrained to specific classes of func-

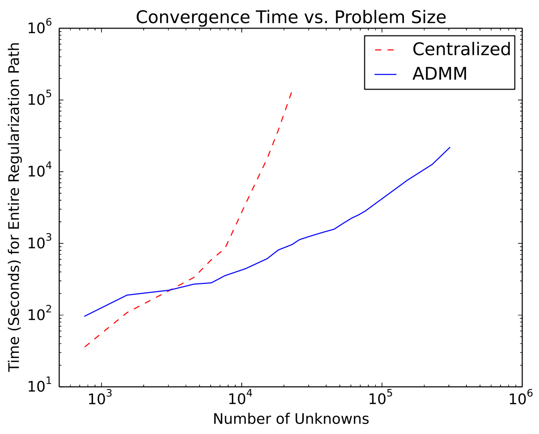


Figure 3: Convergence comparison between centralized and ADMM methods for SVM problem.

Number of Unknowns	ADMM Solution Time (seconds)
100,000	12.20
1 million	18.16
10 million	128.98
100 million	822.62

Table 2: Convergence time for large-scale 3-regular graph solved at a single (constant) value of λ.

tions, and yet it is still able to scale to tens of millions of unknown variables.

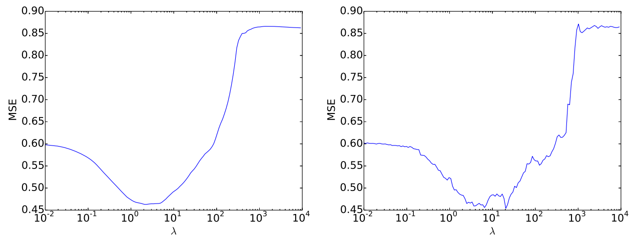
5.2 Spatial Clustering with Regressors

In this example, as described in the introduction, we attempt to estimate the price of homes based on latitude/longitude data and a set of features. Home prices often cluster together along neighborhood lines. In this case, the clustering occurs when nearby houses have similar pricing models, while edges that have non-zero edge differences will be between those in different neighborhoods. As houses are grouped together, each cluster builds its own local linear regression model to predict prices in its region. Then, when there is a new house, we can infer its regression model from the local neighborhood to estimate the sales price.

Dataset. We look at a list of real estate transactions over a one-week period in May 2008 in the Greater Sacramento area[1]. This dataset contains information on 985 sales, including latitude, longitude, number of bedrooms, number of bathrooms, square feet, and sales price. However, as often happens with real data, we are missing some of the values. 17% of the home sales are missing at least one of the features; *i.e.*, some of the bedroom/bathroom/size data is not provided. The price and all attributes are standardized to zero mean and unit variance, so any missing features are ignored by setting the value to zero, the average. To verify our results, we use a random subset of 200 houses as our test set.

Network. We build the graph by using the latitude/longitude coordinates of each house. After removing the test set, we connect every remaining house to the five nearest homes with an edge weight inversely proportional to the distance between the houses. If house j is in the set of nearest neighbors of i, there is an undirected edge regardless of whether or not house i is one of j's nearest neighbors. The resulting graph leaves 785 nodes, 2447 edges, and has a diameter of 61.

[1]Data available at `http://support.spatialkey.com/spatialkey-sample-csv-data/`.



(a) Convex (b) Non-Convex

Figure 4: Regularization path for housing data.

Method	Mean Squared Error (MSE)
Geographic ($\lambda = 0$)	0.6013
Regularized Linear Regression ($\lambda \geq \lambda_{\text{critical}}$)	0.8611
Naive Prediction (Global Mean)	1.0245
Convex Network Lasso	0.4630
Non-Convex Network Lasso	0.4539

Table 3: MSE for housing price predictions on test set.

Optimization Parameter and Objective Function. At each node, we solve for

$$x_i = \begin{bmatrix} a_i & b_i & c_i & d_i \end{bmatrix}^T,$$

which gives us the weights of the regressors. The price estimate is given by

$$\overline{\text{price}_i} = a_i \cdot \text{Bedrooms} + b_i \cdot \text{Bathrooms} + c_i \cdot \text{SQFT} + d_i,$$

where the constant offset d_i is the "baseline". To prevent overfitting, we regularize the a_i, b_i, and c_i terms, everything besides the offset. The objective function at each node then becomes

$$f_i = \|\overline{\text{price}_i} - \text{price}_i\|_2^2 + \mu \, \|\tilde{x_i}\|_2^2$$

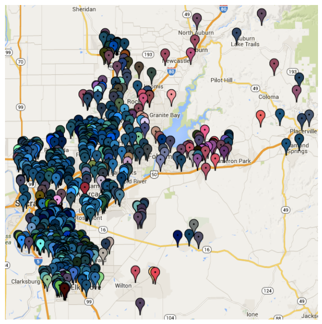
where $\tilde{x_i} = \begin{bmatrix} a_i & b_i & c_i \end{bmatrix}^T$, price_i is the actual sales price, and μ is a constant regularization parameter.

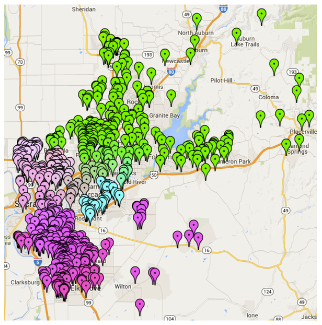
To predict the prices on the test set, we connect each new house to the 5 nearest homes, weighted by inverse distance, just like before. We then infer the value of x_j at node j by solving problem (5), and we use this value to estimate the sales price.

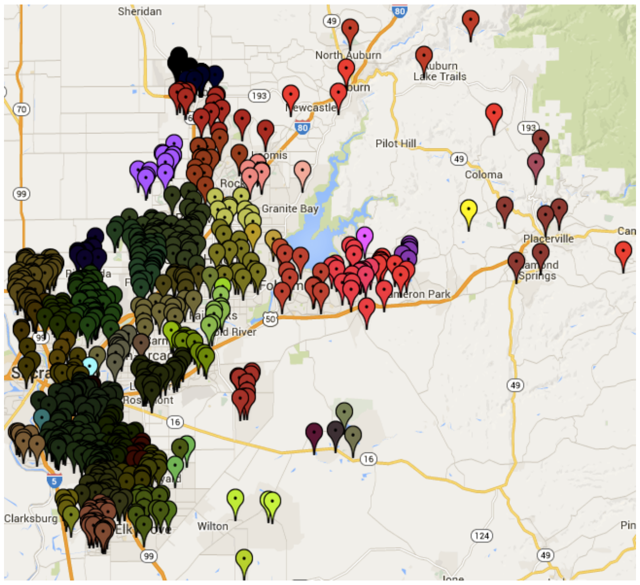
Results. We plot the mean squared error (MSE) vs. λ in Figure 4 for both the convex and non-convex formulations of the problem. Once again, the two extremes of the regularization path are relevant baselines.

At $\lambda = 0$, the regularization term in $f_i(x_i)$ insures that the only non-zero element of x_i is d_i. This ignores the regressors and is a prediction based solely on spatial data. Our estimate for each new house is simply the weighted median price of the 5 nearest homes, which leads to an MSE of 0.6013 on the test set. For large λ's, we are fitting a common model for all the houses. This is just regularized linear regression on the entire dataset and is the canonical method of estimating housing prices from a series of features. Note that this approach completely ignores the geographic network. As expected, it performs rather poorly, with an MSE of 0.8611. Since the prices are standardized with unit variance, a naive guess (with no information about the house) would just be the global average of the training set, which has an MSE of 1.0245. The convex and non-convex methods are both maximized around $\lambda = 5$, with minimum MSE's of 0.4630 and 0.4539, respectively.

We can visualize the clustering pattern by overlaying the network on a map of Sacramento. We plot each sale with a marker, colored according to its corresponding x_i (so houses with similar colors have similar models, and those with the same color are in consensus). With this, we see how the clustering pattern emerges.


(a) $\lambda = 0.1$


(b) $\lambda = 1000$



(c) $\lambda = 10$

Figure 5: Regularization path clustering pattern.

In Figure 5, we look at this plot for three values of λ. In 5(a), λ is too small, so the neighborhoods have not yet formed. On the other hand, in 5(b), λ is too large. The clustering is clear, but it performs poorly because it forces together neighborhoods which are very different. Figure 5(c) is a viable choice of λ, leading to low MSE while showing a clear partitioning of the network into neighborhoods of different sizes.

Aside from outperforming the baselines, this method is also well-suited to detect and handle anomalies. As shown in the plots, outliers are often treated as single-element clusters, for example the yellow house on the right side of 5(c). These houses are ones which do not fit in with their local model (for a variety of possible reasons), but using the network lasso, neither they nor their neighbors are adversely affected too significantly by each other. Of course, as λ approaches $\lambda_{\text{critical}}$, these clusters are forced together into consensus. However, near the optimal λ, we accurately classify these anomalies, isolate them from the rest of the graph, and build separate and relatively accurate models for both subsets.

5.3 Event Detection in Time Series Data

Lastly, we aim to predict the existence of certain "events" in a building, those which were officially listed by the building coordinator. We are given the entry and exit data from the building over a 15 week interval. For these events, we expect to see an anomalous increase in traffic. Note that this is just a partial ground truth, only containing events officially reported by the coordinator, and many unreported events likely occurred during this interval. Therefore, "false positives" are not necessarily incorrect, so the absolute results (how accurately we predict the events) are not a perfect indicator of performance. However, this provides a good benchmark, especially when compared to a common baseline.

Dataset. The data comes from the main door of the Calit2 building at UC Irvine. This count data, the number of entries and exits, is reported once every 30 minutes over the course of 15 weeks from July to November 2005, for a total of 5,040 readings[2]. Additionally, we use a list of the 30 official events which occurred inside the building during that interval.

Network. We build a linear network where node i, covering the ith interval in the time series, has only two edges. These connect it to nodes $i-1$ and $i+1$. The first and last nodes only have one edge, leaving 5,040 nodes and 5,039 edges. There are more complicated ways to model the coupling of time series data, but we opt for simplicity since our goal is to show one approach, rather than necessarily the optimal method, of solving this class of problems.

Optimization Parameter and Objective Function. Traffic is periodic on a weekly basis. That is, a relatively similar number of people enter and exit the building on, for example, Mondays from 1:00 - 1:30PM. We do not care for instance that there is more traffic at 1:00 PM than at 1:00 AM. This is not an indicator that an event occurred at 1PM. Instead, we care about the number of people relative to the periodic signal. We let

$$\overline{x}_i = \begin{bmatrix} \text{in}_i - \overline{\text{in}}(i \bmod 336) \\ \text{out}_i - \overline{\text{out}}(i \bmod 336) \end{bmatrix},$$

where $\overline{\text{in}}(i \bmod 336)$ and $\overline{\text{out}}(i \bmod 336)$ are the median value of entrances/exits for the given time and day of the week ($7 \cdot 24 \cdot 2 = 336$) over the 15 week interval. We use the median because the mean can be skewed by the increases due to actual events.

The objective function is defined as

$$f_i = \|x_i - \overline{x}_i\|_2^2 + \mu\|x_i\|_2.$$

The variable that we optimize over, x_i, is an attempt to match the non-periodic signal at that time. The regularization term on x_i is a lasso penalty, so only a select few of the x's will be non-zero. These non-zero values refer to the times of the anomalous events that we are trying to predict. It is worth noting that for any finite network lasso parameter λ, there exists a μ large enough so that every x_i is guaranteed to be $[0,0]^T$.

An event often manifests itself as a sustained period of increased activity. Therefore, we declare an event on the interval $[t, t+k]$ if

$$x_{i,\text{in}} + x_{i,\text{out}} > 0 \quad i \in [t, t+k].$$

We vary μ to change the number of events predicted. For small μ, the slightest noise can be interpreted as an event. Large μ's lead to fewer predictions, until eventually every $x(t)$ is forced to 0, as mentioned before. The parameter λ determines the average event length, as it encourages prolonged increases in activity and discourages single outliers from being picked up. However, in this example, the model is relatively robust to changes in λ (up to a certain point), so we keep it constant as we vary μ, as a slight modification of the regularization path from previous experiments.

Baseline. This type of problem is often modeled as a Poisson process, so we use that as our baseline method [15]. We consider each

[2]Data from `https://archive.ics.uci.edu/ml/datasets/CalIt2+Building+People+Counts` [18].

Number of Correct Events Detected	Predicted Events		
	Convex	Non-Convex	Poisson
30	146	201	264
29	125	135	214
28	116	121	201
27	101	116	188
26	97	114	131
24	76	78	100
18	56	64	62

Table 4: Number of required predictions to detect events.

time and day of the week as having an independent Poisson rate λ (which is unrelated to the regularization parameter with the same name in the network lasso). We set λ, the "expected" number of count data, to the maximum likelihood estimate of a Poisson process, the mean of the 15 values. λ_{in} and λ_{out} are calculated independently. We define an event from $[t, t+k]$ if

$$P(N(i), \lambda(i)) = \left(\frac{e^{-\lambda_{\text{in}}}\lambda_{\text{in}}^{N_{\text{in}}(i)}}{N_{\text{in}}(i)!}\right)\left(\frac{e^{-\lambda_{\text{out}}}\lambda_{\text{out}}^{N_{\text{out}}(i)}}{N_{\text{out}}(i)!}\right) < \epsilon \qquad i \in [t, t+k].$$

This says that the given number of entries and exits at time i occurs with probability less than ϵ. Since only large totals should trigger a predicted event (rather than abnormally low entry/exit numbers), one final requirement is that either $N_{in} > \lambda_{in}$ or $N_{out} > \lambda_{out}$ for every t in the interval. Varying the threshold ϵ, similar to μ for our approach, changes the number of predicted events.

Results. For both our model and the baseline, we compute the number of correct events vs. number of predicted events. We define a correct prediction as one in which the prediction and the true event overlap. The accuracy of all three approaches at several key points is summarized in Table 4. As shown, both the convex and non-convex methods outperform the Poisson baseline (though the convex approach does noticeably better than the non-convex). The Poisson is able to catch the "low-hanging fruit", the easy-to-detect events, with relatively good accuracy. The discrepancy arises in the less obvious ones. Again, this is just a partial ground truth and it is likely that there are many more than 30 events, but the poor performance of the Poisson method — it takes 264 predictions to find all 30 events — suggests that it may be an imperfect method of event detection. Note that more complicated models, specifically tuned for outlier detection, may beat these results. For example when an event occurs, we expect to see a large spike in inbound traffic at the beginning of the event, and a similar outbound one at the end. Our approach could easily be modified in future work to account for additional information such as this. However, as a simple model and a proof of concept, these results are very encouraging.

6. CONCLUSION AND FUTURE WORK

In this paper, we have shown that within one single framework, it is possible to better understand and improve on many common machine learning and network analysis problems. The network lasso is a useful way of representing convex optimization problems, and the magnitude of the improvements in the experiments show that this approach is worth exploring further, as there are many potential ideas to build on. The non-convex method gave comparable performance to the convex approach, and we leave for future work the analysis of different non-convex functions $\phi(u)$. It is also possible to look at the sensitivity of these results to the structure of the network. For example, we could attempt to iteratively reweigh the edge weights to attain some desired outcome. Within the ADMM algorithm, there are many ways to improve speed, performance, and robustness. This includes finding closed-form solutions for common objective functions $f_i(x_i)$, automatically determining the optimal ADMM parameter ρ, and even allowing edge objective functions $f_e(x_i, x_j)$ beyond just the weighted network lasso. As this topic develops further, there is an opportunity for easy-to-use software packages which allow programmers to solve these types of large-scale optimization problems in a distributed setting without having to specify the implementation details, which would greatly improve the practical benefit of this work.

Acknowledgments

The authors would like to thank Trevor Hastie for his advice on the network lasso, Stephen Bach and Christopher Ré for their help with graphical models, and Rok Sosič for his assistance during the large-scale implementation. This research has been supported in part by the Sequoia Capital Stanford Graduate Fellowship, NSF IIS-1016909, CNS-1010921, IIS-1149837, IIS-1159679, ARO MURI, DARPA XDATA, SMISC, SIMPLEX, Stanford Data Science Initiative, Boeing, Facebook, Volkswagen, and Yahoo.

7. REFERENCES

[1] S. H. Bach, B. Huang, B. London, and L. Getoor. Hinge-loss Markov random fields: Convex inference for structured prediction. In *UAI*, 2013.

[2] P. Bose, A. Maheshwari, and P. Morin. Fast approximations for sums of distances, clustering and the Fermat–Weber problem. *Computational Geometry*, 24(3):135–146, 2003.

[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*, 3:1–122, 2011.

[4] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[5] E. Candès, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *Information Theory, IEEE Transactions on*, 52(2):489–509, 2006.

[6] E. Candès, M. Wakin, and S. Boyd. Enhancing sparsity by reweighted ℓ_1 minimization. *Journal of Fourier analysis and applications*, 14:877–905, 2008.

[7] E. Chi and K. Lange. Splitting methods for convex clustering. *JCGS*, 2013.

[8] M. Chiang, S. H. Low, A. R. Calderbank, and J. C. Doyle. Layering as optimization decomposition: A mathematical theory of network architectures. *Proceedings of the IEEE*, 95(1):255–312, 2007.

[9] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.

[10] S. Diamond, E. Chu, and S. Boyd. CVXPY. `http://cvxpy.org/`, 2014.

[11] A. Domahidi, E. Chu, and S. Boyd. ECOS: An SOCP solver for embedded systems. In *ECC*, 2013.

[12] T. Hastie, S. Rosset, R. Tibshirani, and J. Zhu. The entire regularization path for the support vector machine. *Journal of Machine Learning Research*, 5:1391–1415, 2004.

[13] M. R. Hestenes. Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, 4:302–320, 1969.

[14] T. Hocking, A. Joulin, F. Bach, and J. Vert. Clusterpath: an algorithm for clustering using convex fusion penalties. In *ICML*, 2011.

[15] A. Ihler, J. Hutchins, and P. Smyth. Adaptive event detection with time-varying Poisson processes. In *KDD*, 2006.

[16] S. Kok, P. Singla, M. Richardson, P. Domingos, M. Sumner, H. Poon, and D. Lowd. The Alchemy system for statistical relational AI. *University of Washington, Seattle*, 2005.

[17] J. Leskovec and R. Sosič. Snap.py: SNAP for Python. `http://snap.stanford.edu`, 2014.

[18] M. Lichman. UCI machine learning repository, 2013.

[19] M. Meila and M. I. Jordan. Learning with mixtures of trees. *Journal of Machine Learning Research*, 1:1–48, 2001.

[20] B. Muthen. Latent variable mixture modeling. *New developments and techniques in structural equation modeling*, pages 1–33, 2001.

[21] N. Parikh and S. Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1:123–231, 2014.

[22] K. Pelckmans, J. De Brabanter, J. Suykens, and B. De Moor. Convex clustering shrinkage. In *PASCAL Workshop on Statistics and Optimization of Clustering*, 2005.

[23] R. Tibshirani, M. Saunders, S. Rosset, J. Zhu, and K. Knight. Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society*, 67(1):91–108, 2005.

[24] B. Wahlberg, S. Boyd, M. Annergren, and Y. Wang. An ADMM algorithm for a class of total variation regularized estimation problems. *IFAC Symp. Syst. Ident*, 2012.

[25] K. Q. Weinberger, F. Sha, Q. Zhu, and L. K. Saul. Graph Laplacian regularization for large-scale semidefinite programming. In *NIPS*, 2006.

[26] S. Yang, J. Wang, W. Fan, X. Zhang, P. Wonka, and J. Ye. An efficient ADMM algorithm for multidimensional anisotropic total variation regularization problems. In *KDD*, 2013.

[27] C. Yanover, T. Meltzer, and Y. Weiss. Linear programming relaxations and belief propagation–an empirical study. *Journal of Machine Learning Research*, 7:1887–1907, 2006.

[28] M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B*, 68:49–67, 2006.

[29] C. Zhang, C. Ré, A. A. Sadeghian, Z. Shan, J. Shin, F. Wang, and S. Wu. Feature engineering for knowledge base construction. *arXiv preprint arXiv:1407.6439*, 2014.

APPENDIX

A. ANALYTICAL SOLUTION TO z-UPDATE

We will show that the solution to

$$\text{minimize} \quad \lambda w_{ij}\|z_{ij}-z_{ji}\|_2 + (\rho/2)\big(\|x_i^{k+1}-z_{ij}+u_{ij}^k\|_2^2 + \|x_j^{k+1}-z_{ji}+u_{ji}^k\|_2^2\big),$$

with variables z_{ij} and z_{ji}, is

$$z_{ij}^\star = \theta(x_i+u_{ij}) + (1-\theta)(x_j+u_{ji})$$
$$z_{ji}^\star = (1-\theta)(x_i+u_{ij}) + \theta(x_j+u_{ji}),$$

where θ is defined in equation (6).

We first note that the objective is strictly convex, so the solution is unique. As in §4, we let

$$a = x_i^{k+1}+u_{ij}^k, \quad b = x_j^{k+1}+u_{ji}^k, \quad c = \lambda w_{ij},$$

so the original problem turns into

$$\text{minimize} \quad c\|z_{ij}-z_{ji}\|_2 + (\rho/2)\left(\|a-z_{ij}\|_2^2 + \|b-z_{ji}\|_2^2\right).$$

There are two possible cases for the optimal values $z_{ij}^\star$ and $z_{ji}^\star$.

Case 1: $z_{ij}^\star = z_{ji}^\star$. If the two variables are equal, then $\|z_{ij}-z_{ji}\|_2 = 0$, so the only terms remaining are

$$(\rho/2)\left(\|a-z_{ij}\|_2^2 + \|b-z_{ji}\|_2^2\right).$$

Minimizing over the constraint that $z_{ij}=z_{ji}$ yields $z_{ij}^\star = z_{ji}^\star = (1/2)(a+b)$, with objective value $\rho/4\|a-b\|_2^2$.

Case 2: $z_{ij}^\star \neq z_{ji}^\star$. When the two variables are not equal, the objective is differentiable. In this case, the necessary and sufficient condition for optimality is $\nabla f = 0$, or

$$\nabla\left(c\|z_{ij}-z_{ji}\|_2 + (\rho/2)\|a-z_{ij}\|_2^2 + (\rho/2)\|b-z_{ji}\|_2^2\right) = 0.$$

The gradient can be written as

$$c\begin{bmatrix} \frac{z_{ij}-z_{ji}}{\|z_{ij}-z_{ji}\|_2} \\ -\frac{z_{ij}-z_{ji}}{\|z_{ij}-z_{ji}\|_2}\end{bmatrix} + \begin{bmatrix} -\rho(a-z_{ij}) \\ -\rho(b-z_{ji})\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix},$$

so the two equations that must be satisfied are

$$c\frac{z_{ij}-z_{ji}}{\|z_{ij}-z_{ji}\|_2} - \rho(a-z_{ij}) = 0, \qquad -c\frac{z_{ij}-z_{ji}}{\|z_{ij}-z_{ji}\|_2} - \rho(b-z_{ji}) = 0.$$

Letting $\mu = \|z_{ij}-z_{ji}\|_2$, we get

$$c(z_{ij}-z_{ji}) = \mu\rho(a-z_{ij}), \qquad -c(z_{ij}-z_{ji}) = \mu\rho(b-z_{ji}).$$

Adding the two equations gives

$$z_{ij}+z_{ji} = a+b,$$

and subtracting them leads to

$$z_{ij}-z_{ji} = \frac{\mu\rho(a-b)}{2c+\mu\rho}.$$

Treating μ as a constant, this yields a system of linear equations for z_{ij} and z_{ji}, which we solve to obtain

$$z_{ij} = \theta a + (1-\theta)b, \qquad z_{ji} = (1-\theta)a + \theta b,$$

where

$$\theta = \frac{1}{2} + \frac{\mu\rho}{4c+2\mu\rho}.$$

We know that $\mu = \|z_{ij}-z_{ji}\|_2$, so we plug in for z_{ij} and z_{ji},

$$\mu = \|z_{ij}-z_{ji}\|_2 = \left\|\frac{\mu\rho(a-b)}{2c+\mu\rho}\right\|_2 = \frac{\mu\rho}{2c+\mu\rho}\|a-b\|_2,$$

which reduces to

$$1 = \frac{\rho}{2c+\mu\rho}\|a-b\|_2.$$

From this, we can solve for μ,

$$\mu = \|a-b\|_2 - \frac{2c}{\rho}.$$

We plug in μ to solve for θ, which yields

$$\theta = \frac{1}{2} + \frac{\left(\|a-b\|_2 - \frac{2c}{\rho}\right)\rho}{4c+2\rho\left(\|a-b\|_2 - \frac{2c}{\rho}\right)} = \frac{1}{2} + \frac{\rho\|a-b\|_2 - 2c}{2\rho\|a-b\|_2}.$$

This is then reduced to

$$\theta = 1 - \frac{c}{\rho\|a-b\|_2}.$$

However, this only holds if $z_{ij} \neq z_{ji}$. When this condition is not satisfied, we know the solution is case 1, which is equivalent to $\theta = \frac{1}{2}$. When it is satisfied, we need to compare the resulting objective with $\rho/4\|a-b\|_2^2$, the value from case 1. Routine calculations show that this holds when $\theta > \frac{1}{2}$. Therefore, combining these equations and plugging in for a, b, and c, we arrive at our solution,

$$\theta = \max\left(1 - \frac{\lambda w_{ij}}{\rho\|x_i+u_{ij}-(x_j+u_{ji})\|_2}, 0.5\right).$$