A fast and flexible algorithm for the graph-fused lasso

1 The Graph-Fused Lasso

We seek to find a scalable, efficient solution to the following optimization problem over a general graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and for a general smooth convex loss function ℓ:

$$\underset{\boldsymbol{\beta} \in \mathbb{R}^{|\mathcal{V}|}}{\text{minimize}} \quad \sum_{s=1}^{n} \ell(\beta_s) + \lambda \sum_{(r,s) \in \mathcal{E}} |\beta_r - \beta_s| \,. \tag{1}$$

We can alternatively rewrite (1) in a more general form:

$$\underset{\boldsymbol{\beta} \in \mathbb{R}^{|\mathcal{V}|}}{\text{minimize}} \quad \sum_{s \in \mathcal{V}} \ell(\beta_s) + \lambda \left|\left|D\boldsymbol{\beta}\right|\right|_1 \,, \tag{2}$$

where D is the $|\mathcal{E}| \times |\mathcal{V}|$ oriented edge matrix with the i^{th} row corresponding to the i^{th} edge in $\mathcal{E}$. Specifically, for a given edge $\mathcal{E}_i = (s, r)$, the row D_i takes the values -1 and 1 in the s and r columns, respectively, and 0 everywhere else.

We note that this problem is convex (though not smooth), and thus capable of being solved via classical methods such as projected subgradient descent. For massive graphs, with potentially millions of nodes, such approaches take prohibitively long to converge. Many efficient, specialized algorithms have been developed for the case when ℓ is a Gaussian loss and the graph has a specific constrained form. For example, when $\mathcal{G}$ is a one-dimensional chain graph, (1) is the ordinary (1D) fused lasso (Tibshirani et al., 2005), solvable in linear time via dynamic programming (Johnson, 2013). When $\mathcal{G}$ is a d-dimensional grid graph, (1) is often referred to as total variation denoising

(Rudin et al., 1992), for which several efficient solutions have been proposed (Chambolle and Darbon, 2009a; Barbero and Sra, 2011, 2014). In the case of a general graph with a Gaussian loss, the dual formulation can be solved efficiently along a solution path of decreasing choices of λ, with efficient warm-starts from the previous solution (Tibshirani and Taylor, 2011). For fixed λ, the most efficient methods to-date have either relied on leveraging the structure of D in combination with efficient methods for solving linear systems (Kim et al., 2009), or by decomposing the problem into a series of subproblems (one for every edge) and using proximal algorithms (Hallac et al., 2015). While the latter approach is capable of scaling by distributing the work over a compute cluster, splitting into m subproblems, where $m = |\mathcal{E}|$, makes the overall algorithm somewhat slow to converge.

The core idea of our algorithm is to decompose a graph into a much smaller set of subproblems, each of which can be solved in linear time, thereby substantially improving the convergence rate. Since the algorithm relies on a fundamental result in graph theory, it is useful to first define some precise terminology. A *walk* is a sequence of vertices, where there exists an edge between the preceding and following vertices in the sequence. A *trail* is a walk in which all the edges are distinct. An *Eulerian trail* is a trail which visits every edge in a graph exactly once. A *tour* (also called a *circuit*) is a trail that begins and ends at the same vertex. An *Eulerian tour* is a circuit where all edges in the graph are visited exactly once.

To begin decomposing the graph, first note that that if $\mathcal{G}$ is not connected, then the objective function in (1) is separable across the connected components of $\mathcal{G}$, each of which can be solved independently. Therefore, for the rest of this note we assume (without loss of generality) that the edge set $\mathcal{E}$ forms a connected graph. Next, note that every connected graph has an even number of odd-degree vertices (West, 2001). Following upon this property, we finally note that theorem 1.2.33 from West (2001) states that any connected graph can be decomposed into a set of trails $\mathcal{T}$. We reproduce the theorem below:

Theorem 1. *(West, 2001, Thm 1.2.33). The edges of a connected graph with exactly* $2K$ *odd-*

degree vertices can be partitioned into K trails if $K > 0$. If $K = 0$, there is an Eulerian tour. Furthermore, the minimum number of trails that can partition the graph is $max(1, K)$.

This theorem enables us to rewrite the penalty term in (1) as a summation over trails $\mathcal{T} = \{\tau_1, \ldots, \tau_K\}$, where each trail τ has vertex set $\mathcal{V}^{(\tau)}$ and edge set $\mathcal{E}^{(\tau)}$:

$$\underset{\boldsymbol{\beta} \in \mathbb{R}^p}{\text{minimize}} \quad \sum_{s=1}^{n} \ell(\beta_s) + \lambda \sum_{\tau \in \mathcal{T}} \sum_{(r,s) \in \mathcal{E}^{(\tau)}} |\beta_r - \beta_s| \,. \tag{3}$$

Slack variables z_r and z_s are then introduced for each β_r and β_s occurrence in the penalty term, respectively, which results in the following equivalent problem:

$$\begin{aligned} \underset{\boldsymbol{\beta} \in \mathbb{R}^n}{\text{minimize}} \quad & \sum_{s=1}^{n} \ell(\beta_s) + \lambda \sum_{\tau \in \mathcal{T}} \sum_{(r,s) \in \mathcal{E}^{(\tau)}} |z_r^{(\tau)} - z_s^{(\tau)}| \,. \\ \text{subject to} \quad & z_r^{(\tau)} = \beta_r \\ & z_s^{(\tau)} = \beta_s \,, \end{aligned} \tag{4}$$

where the constraints hold for all edges $(r, s) \in \mathcal{E}^{(\tau)}$, for all $\tau \in \mathcal{T}$. We can then solve this problem efficiently via the Alternating Direction Method of Multipliers (ADMM) (Boyd et al., 2011) with the following updates:

$$\hat{\boldsymbol{\beta}}^{(j+1)} = \underset{\boldsymbol{\beta}}{\text{argmin}} \left(\sum_{s=1}^{n} \ell(\beta_s) + \frac{\alpha}{2} \left|\left| A\boldsymbol{\beta} - \mathbf{z}^{(j)} + \mathbf{u}^{(j)} \right|\right|^2 \right) \tag{5}$$

$$\mathbf{z}^{(\tau, j+1)} = \underset{\mathbf{z}}{\text{argmin}} \left(\frac{\alpha}{2} \sum_{s \in \mathcal{V}^{(\tau)}} (\tilde{y}_s - z_s^{(\tau)})^2 + \sum_{(r,s) \in \mathcal{E}^{(\tau)}} |z_r^{(\tau)} - z_s^{(\tau)}| \right) \,, \quad \tau \in \mathcal{T} \tag{6}$$

$$\mathbf{u}^{(j+1)} = \mathbf{u}^{(j)} + A\boldsymbol{\beta}^{(j+1)} - \mathbf{z}^{(k+1)} \,, \tag{7}$$

where u is the scaled dual variable, α is the scalar ADMM penalty parameter, $\tilde{y}_s = \beta_s - u_s$, and A is a sparse binary matrix used to encode the appropriate β_s for each $z_j^{(\tau)}$. The update in (5) is separable in $\boldsymbol{\beta}$ with each β_s having a closed-form solution. In the concrete case of a Gaussian loss,

the $\hat{\beta}_s$ updates have a simple closed-form solution:

$$\hat{\beta}_s^{k+1} = \frac{2y_s + \alpha \sum_{j \in \mathcal{J}} (z_j - u_j)}{2 + \alpha |\mathcal{J}|}, \tag{8}$$

where $\mathcal{J}$ is the set of dual variable indices that map to β_j. The update in (6) corresponds to solving a weighted 1-dimensional fused lasso problem, which can be done in linear time via an efficient dynamic programming routine (Johnson, 2013; Arnold et al., 2014). The updates in (5)-(7) are iterated in order until the dual and primal residuals have sufficiently small norms.

2 Benchmarks

To evaluate the effectiveness of the algorithm in Section 1, we compare it against several other approaches to solving (3). For all benchmarks, we report timings averaged across 30 independent trials. In each trial, we randomly create four blobs of different mean values, with each blob being approximately 5% of the nodes in the graph. All algorithms are then run until 10^{-4} precision and we use a varying penalty acceleration heuristic for the ADMM algorithm (Boyd et al., 2011), though performance results are not meaningfully different for different precision levels or fixed penalty parameters.

As a baseline approach, a sparse LU decomposition of the D matrix is cached, enabling fast solving of the linear system implied by (2). A more recent method in the literature (Wahlberg et al., 2012) uses two sets of slack variables are introduced in order to reformulate the problem such that the dominant cost is in repeatedly solving the system $(I + L)x = b$, where L is the graph Laplacian and only b changes at every iteration. Modern algebraic multigrid (AMG) methods have become highly efficient at solving such systems for certain forms of L, and in the following benchmarks we compare against an implementation of this approach using a smoothed aggregation AMG approach (Bell et al., 2011) with conjugate gradient acceleration. In all reported timings, we do not include the time to calculate the multigrid hierarchy, nor the LU decomposition, though we

note this can add significant overhead and may even take longer than solving the subsequent linear system. Finally, we also compare against a trail decomposition containing only edges, equivalent to a special case of the network lasso (Hallac et al., 2015).

The algorithms were evaluated on two synthetic datasets: random and grid graphs. Random graphs were created by randomly connecting nodes to ensure the graph had approximately 99.8% sparsity, whereas the square grid graphs had adjacency defined by each node's immediate neighbor. For random graphs, we generate trails by connecting pairs of odd-degree nodes and forming an Eulerian tour, then removing the inserted edges, leaving K trails. In the grid graph case, it is straight-forward to hand-engineer trails along the rows and columns of the spatial lattice, equivalent to the "proximal stacking" method of (Barbero and Sra, 2014). In preliminary experiments, we found this row/column decomposition to be more efficient than the Eulerian trails. Since the grid graph case always admits such a straightforward decomposition, we use this strategy in our benchmark for the grid graph. Under this constrained spatial structure, a highly efficient parametric max-flow method is also available (Chambolle and Darbon, 2009b), and we compare against that method as well. For both graph types, we track the performance as the number of nodes in the graph increases.

Figure 1 shows the results of the synthetic benchmarks. Both methods based on directly solving linear systems vastly underperform the ADMM and max-flow solutions. On the random graphs, the network lasso approach appears to perform similarly to the GFL algorithm from Section 1, however it vastly underperforms on grid graphs. The GFL algorithm is also competitive with the specialized max-flow algorithm, despite being able to handle non-Gaussian loss functions and arbitrary graphs.

One potential criticism of the synthetic benchmarks is that randomly generated networks often produce graph forms that are unlikely to be seen in the wild. At the same time, square grids are likely too uniformly structured to be common in many scientific applications, with computer vision being a notable exception. Therefore, to better understand performance in practice, we also

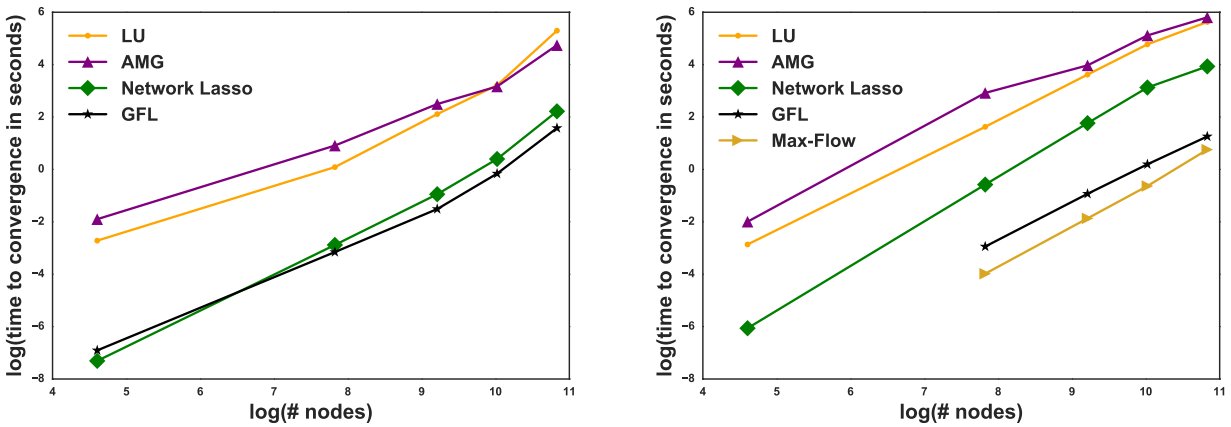

Figure 1: Results of the two synthetic benchmark on random graphs (left) and grid graphs (right). The GFL method from Section 1 outperforms all three generic graph-based methods and is competitive with the specialized spatial smoothing max-flow algorithm.

conduct benchmarks on two real-world datasets: a DNA electrophoresis model and an acoustic vibrobox simulation, both taken from The University of Florida sparse matrix collection (Davis and Hu, 2011). Figure 2 shows the adjacency structure of these matrices (left: DNA; middle: vibrobox). While both exhibit some degree of structure, the DNA example presents a much higher degree of regularity and is more similar to a grid graph than the vibrobox example. Figure 2 (right) shows the relative speedup of the GFL method compared to the other three generic graph methods, with further confirmation that the GFL method is superior.

The trail-based GFL algorithm is highly efficient and can solve (1) for millions of nodes locally on a single machine in seconds. In our experiments, we found the method to be substantially more efficient than any other generic graph method and nearly as efficient as the specialized methods mentioned earlier for multidimensional grid graphs. The key update steps in our ADMM algorithm is also embarrassingly parallel: each $\hat{\beta}_s$ in (5) and each $\mathbf{z}^{(\tau)}$ in (6) can be solved independently of the other nodes and trails, respectively. For even more massive problems, where the data may not fit in memory, our algorithm will continue to scale well due to its distributed nature. Finally, we note that our algorithm is extensible to other smooth, convex loss functions ℓ, though we do not investigate such functions here.

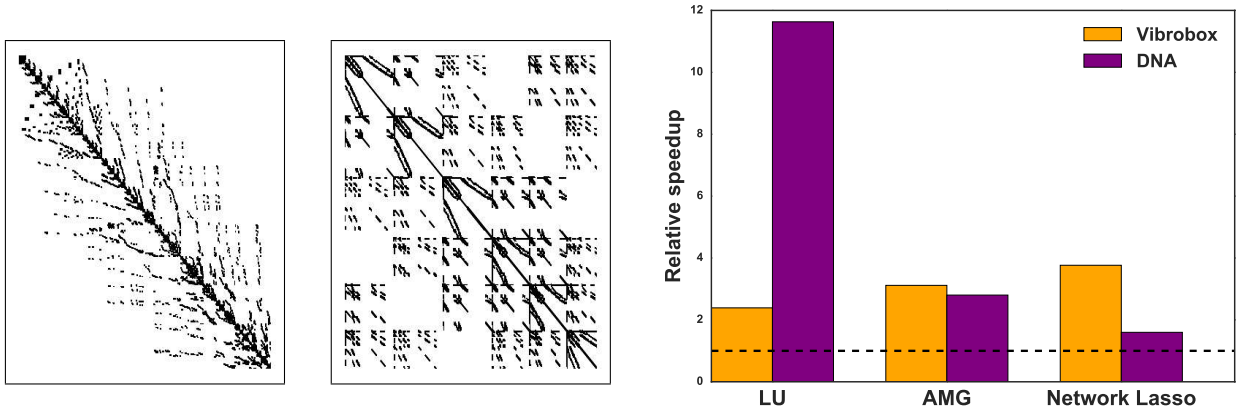

Figure 2: Two real-world graphs corresponding to a DNA electrophoresis model (left) and an acoustic vibrobox simulation (middle), and relative times compared to the GFL algorithm (right). The GFL faster than all methods on both graphs and more than twice as fast as all three other methods on at least one of the two graphs.

References

T. Arnold, V. Sadhanala, and R. J. Tibshirani. *glmgen: Fast generalized lasso solver*. https://github.com/statsmaths/glmgen, 2014. R package version 0.0.2.

A. Barbero and S. Sra. Fast newton-type methods for total variation regularization. In L. Getoor and T. Scheffer, editors, *ICML*, pages 313–320. Omnipress, 2011. URL `http://dblp.uni-trier.de/db/conf/icml/icml2011.html#JimenezS11`.

A. Barbero and S. Sra. Modular proximal optimization for multidimensional total-variation regularization. 2014. URL `http://arxiv.org/abs/1411.0589`.

W. N. Bell, L. N. Olson, and J. B. Schroder. PyAMG: Algebraic multigrid solvers in Python v2.0, 2011. URL `http://www.pyamg.org`. Release 2.0.

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): 1–122, 2011.

A. Chambolle and J. Darbon. On total variation minimization and surface evolution using parametric maximum flows. *International journal of computer vision*, 84(3):288–307, 2009a.

A. Chambolle and J. Darbon. On total variation minimization and surface evolution using parametric maximum flows. *International journal of computer vision*, 84(3):288–307, 2009b.

T. A. Davis and Y. Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1, 2011.

D. Hallac, J. Leskovec, and S. Boyd. Network lasso: Clustering and optimization in large-scale graphs. *21st ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'15)*, 2015.

N. A. Johnson. A dynamic programming algorithm for the fused lasso and l 0-segmentation. *Journal of Computational and Graphical Statistics*, 22(2):246–260, 2013.

S.-J. Kim, K. Koh, S. Boyd, and D. Gorinevsky. \ell_1 trend filtering. *SIAM review*, 51(2):339–360, 2009.

L. Rudin, S. Osher, and E. Faterni. Nonlinear total variation based noise removal algorithms. *Phys. D*, 60 (259–68), 1992.

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

R. J. Tibshirani and J. Taylor. The solution path of the generalized lasso. *Annals of Statistics*, 39:1335–71, 2011.

B. Wahlberg, S. Boyd, M. Annergren, and Y. Wang. An ADMM algorithm for a class of total variation regularized estimation problems. In *Proceedings 16th IFAC Symposium on System Identification*, volume 12, 2012.

D. B. West. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.