Contents lists available at ScienceDirect

Journal of Statistical Planning and Inference

journal homepage: www.elsevier.com/locate/jspi

A graph decomposition-based approach for the graph-fused lasso ☆

Feng Yu [a,*], Archer Yi Yang [b], Teng Zhang [c]

[a] University of Texas at El Paso, 500 W University Ave, El paso, TX 79968, USA
[b] McGill University, 805 Sherbrooke Street West, Montreal, QC H3A 0B9, Canada
[c] University of Central Florida, 4000 Central Florida Blvd, Orlando, FL 32816, USA

ARTICLE INFO

Keywords:
ADMM
Graph-fused lasso
Nonsmooth convex optimization
Total variation minimization
Graph decomposition

ABSTRACT

We propose a new algorithm for solving the graph-fused lasso (GFL), a regularized model that operates under the assumption that the signal tends to be locally constant over a predefined graph structure. The proposed method applies a novel decomposition of the objective function for the alternating direction method of multipliers (ADMM) algorithm. While ADMM has been widely used in fused lasso problems, existing works such as the network lasso decompose the objective function into the loss function component and the total variation penalty component. In contrast, based on the graph matching technique in graph theory, we propose a new method of decomposition that separates the objective function into two components, where one component is the loss function plus part of the total variation penalty, and the other component is the remaining total variation penalty. We develop an exact convergence rate of the proposed algorithm by developing a general theory on the local convergence of ADMM. Compared with the network lasso algorithm, our algorithm has a faster exact linear convergence rate (although in the same order as for the network lasso). It also enjoys a smaller computational cost per iteration, thus converges overall faster in most numerical examples.

1. Introduction

The graph-fused lasso (GFL) has a wide range of applications in machine learning and pattern recognition, including image denoising and segmentation (Chopra and Lian, 2010), texture classification (Nelson, 2013), feature selection (Zhang et al., 2017; Cui et al., 2021), feature learning (Yang et al., 2021), signal processing and computer vision (Mu and Liu, 2020), etc. It offers a wide range of variants based on different graph structures, for example, the chain graph (Nelson, 2013; Zhang et al., 2017), the grid graph (Chopra and Lian, 2010; Mu and Liu, 2020), the complete graph (Yang et al., 2021), or a general graph (Cui et al., 2021). Due to the popularity of GFL, there have been extensive studies on its computation. For the one-dimensional chain graph (Tibshirani et al., 2005), there are taut-string method (Davies and Kovac, 2001), duality-based method (Condat, 2013), dynamic programming-based approach (Johnson, 2013), and modular proximal optimization method (Barbero and Sra, 2018). When the given graph is a two-dimensional grid, the corresponding fused lasso model is often as the total-variation denoising (Rudin et al., 1992), which has important applications in image denoising and processing. A parametric max-flow algorithm proposed in Chambolle and Darbon (2009) can be used to efficiently solve this variant. When the given graph is a tree, one can apply a dynamic programming approach proposed in Kolmogorov et al. (2016) to solve the corresponding fused lasso problem.

While all the aforementioned algorithms can find the exact solutions of different fused lasso problems in finite steps, they are restricted to some specific graph structure and cannot work for general graphs. In addition, they cannot be naturally generalized to

☆ This document is the results of the research project funded by the National Science Foundation, CNS-1818500.
* Corresponding author.
E-mail addresses: fyu@utep.edu (F. Yu), archer.yang@mcgill.ca (A.Y. Yang), teng.zhang@ucf.edu (T. Zhang).

https://doi.org/10.1016/j.jspi.2024.106221
Received 14 April 2023; Accepted 2 August 2024

the setting with multi-dimensional signals, which is sometimes called group fused lasso (Vert and Bleakley, 2010). In this paper, we consider a general graph-fused lasso problem based on the assumption that the signal tends to be locally constant over a predefined graph structure. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ is the set of edges, we let $n = |\mathcal{V}|$ be the cardinality of $\mathcal{V}$ and let $d = |\mathcal{E}|$ be the cardinality of $\mathcal{E}$. We let $\mathbf{x}_i \in \mathbb{R}^p$ be the signal that is associated with the ith vertex of the graph, then the GFL is defined as the solution to the following optimization problem:

$$\min_{\{\mathbf{x}_i\}_{i\in\mathcal{V}}\subset\mathbb{R}^p} \sum_{i\in\mathcal{V}} f_i(\mathbf{x}_i) + \lambda \sum_{(s,t)\in\mathcal{E}} \|\mathbf{x}_t - \mathbf{x}_s\|, \tag{1}$$

where $\|\cdot\|$ is the ℓ_2 norm, the first component is a loss function for the observation $\mathbf{x}_i$, and the second component uses the total variation norm to penalize the difference between the two signals on the edges in the graph.

Numerous iterative algorithms have been proposed to solve (1), including the projected gradient descent (Liu et al., 2010), the smoothing proximal gradient (SPG) (Chen et al., 2012), the alternating linearization method (Lin et al., 2014), the majorization–minimization method (Yu et al., 2015), and many other types (Friedman et al., 2007; Tibshirani et al., 2011). Besides the above methods, the alternating direction method of multipliers (ADMM) has become increasingly popular for solving the fused lasso problems, due to its simplicity and competitive empirical performance (Ye and Xie, 2011; Wahlberg et al., 2012). However, in ADMM, there is a necessary step of solving a linear system for the $n \times n$ matrix $\mathbf{I} + \rho\mathbf{D}^T\mathbf{D}$, where $\mathbf{D}$ represents the edge incidence operator of size $d \times n$ such that all the nonzero elements are given as follows: for the kth edge (i_k, j_k), $\mathbf{D}_{k,i_k} = 1$ and $\mathbf{D}_{k,j_k} = -1$. Solving this linear system costs $O(n^2)$. The authors of Batson et al. (2013) improved it to nearly $O(n)$ using a graph-sparsification based method. The authors of Zhu (2017) proposed a modified ADMM algorithm that has a smaller computational cost of $O(n)$ in an update step, but this modification generally converges slower. A special ADMM algorithm was proposed in Ramdas and Tibshirani (2015) that used dynamic programming in one of the update steps, which can be used in the trend filtering problem, or when $\mathbf{D}$ has a diagonal structure. A method based on the Douglas–Rachford decomposition for the two-dimensional grid graph was proposed in Barbero and Sra (2018), which can be considered as the dual algorithm of ADMM (Eckstein and Bertsekas, 1992). The authors of Tansey and Scott (2015) leveraged fast one-dimensional fused lasso solvers in an ADMM method based on graph decomposition, but it can only be applied when $p = 1$.

Among all ADMM type algorithms, the network lasso (Hallac et al., 2015) is particularly interesting since it is scalable to any generic graphs and can be applied to any dimension $p \geq 1$. Our paper follows the same direction and can be considered as an improvement of the network lasso algorithm. The main contributions of our paper are as follows: we propose a novel ADMM method which decomposes the objective function into two parts based on the graph structures, such that one resulting subgraph structure does not contain any two adjacent edges simultaneously. This method can be applied to any graph and can be generalized to some other problems such as trend filtering as well; We characterize the exact convergence rate for the proposed algorithm by developing a general theory on the local convergence of ADMM, which can also be used for analyzing the convergence rate of the network lasso (Hallac et al., 2015). Compared with the competitive network lasso algorithm, our algorithm has a faster exact linear convergence rate (although both are in the same order); We also study the computational cost of the proposed method and find that it enjoys a smaller computational cost per iteration compared to the network lasso, resulting in faster convergence in most numerical examples.

The rest of this paper is organized as follows. In Section 2 we introduce our proposed method. In Section 3 we analyze its computational complexity per iteration as well as convergence rates and establish the advantage of the proposed algorithm theoretically. Then we compare our algorithm with the alternative algorithm (Hallac et al., 2015) in Section 4 for solving the network lasso, both in simulated data sets and a real-life data set, which verifies the advantage of the proposed method.

2. Methodology

In this section, we first review the network lasso algorithm in Section 2.1, which can be considered as the motivation of this work. Based on it, we propose our method in Section 2.2 and discuss its implementation in Section 2.3. A step-by-step implementation is described in Algorithm 1.

2.1. The network lasso algorithm

The authors of Hallac et al. (2015) introduced the following "Network Lasso" algorithm for solving (1): for any edge $(s,t) \in \mathcal{E}$, a pair of auxiliary variables $\mathbf{z}_{st}, \mathbf{z}_{ts} \in \mathbb{R}^p$ is introduced, which are the copies of $\mathbf{x}_t$ and $\mathbf{x}_s$ respectively. The problem (1) can be rewritten as follows:

$$\begin{aligned}\{\hat{\mathbf{x}}_i\}_{i\in\mathcal{V}} = \underset{\substack{\{\mathbf{x}_i\}_{i\in\mathcal{V}},\\ \{\mathbf{z}_{st},\mathbf{z}_{ts}\}_{(s,t)\in\mathcal{E}}}}{\operatorname{argmin}} & \sum_{i\in\mathcal{V}} f_i(\mathbf{x}_i) + \lambda \sum_{(s,t)\in\mathcal{E}} \|\mathbf{z}_{st} - \mathbf{z}_{ts}\| \\ \text{s.t. } \mathbf{x}_s = \mathbf{z}_{st} \text{ and } & \mathbf{x}_t = \mathbf{z}_{ts} \text{ for all } (s,t) \in \mathcal{E}.\end{aligned} \tag{2}$$

For each edge $(s,t) \in \mathcal{E}$, let $\mathbf{u}_{st}, \mathbf{u}_{ts} \in \mathbb{R}^p$ be the dual variables for $\mathbf{x}_s - \mathbf{z}_{st}$ and $\mathbf{x}_t - \mathbf{z}_{ts}$ respectively, then the augmented Lagrangian is (here x, z and u represent $\{\mathbf{x}_i\}_{i\in\mathcal{V}}$, $\{\mathbf{z}_i\}_{i\in\mathcal{V}}$, and $\{\mathbf{u}_{s,t}\}_{(s,t)\in\mathcal{E}}$ respectively):

$$\begin{aligned}L_\rho(x,z,u) = \sum_{i\in\mathcal{V}} f_i(\mathbf{x}_i) + \sum_{(s,t)\in\mathcal{E}} \Big(& \lambda\|\mathbf{z}_{st} - \mathbf{z}_{ts}\| + \mathbf{u}_{st}^T(\mathbf{x}_s - \mathbf{z}_{st}) \\ & + \mathbf{u}_{ts}^T(\mathbf{x}_t - \mathbf{z}_{ts}) + \frac{\rho}{2}\|\mathbf{x}_s - \mathbf{z}_{st}\|^2 + \frac{\rho}{2}\|\mathbf{x}_t - \mathbf{z}_{ts}\|^2 \Big),\end{aligned} \tag{3}$$

where $\rho > 0$ is a parameter. Then the standard ADMM routine would apply:

$$x^{(k+1)} = \underset{x}{\operatorname{argmin}}\, L_\rho(x, z^{(k)}, u^{(k)}) \tag{4}$$

$$z^{(k+1)} = \underset{z}{\operatorname{argmin}}\, L_\rho(x^{(k+1)}, z, u^{(k)}) \tag{5}$$

$$\mathbf{u}_{st}^{(k+1)} = \mathbf{u}_{st}^{(k)} + \rho(\mathbf{x}_s^{(k+1)} - \mathbf{z}_{st}^{(k+1)}) \tag{6}$$

$$\mathbf{u}_{ts}^{(k+1)} = \mathbf{u}_{ts}^{(k)} + \rho(\mathbf{x}_t^{(k+1)} - \mathbf{z}_{ts}^{(k+1)}). \tag{7}$$

The advantage of this algorithm is that, in each iteration, the optimization problem can be decomposed into smaller subproblems with explicit solutions: the updates of x requires solving problems, $\min_{\mathbf{x}_i} f_i(\mathbf{x}_i) + \|\mathbf{x}_i - \mathbf{t}\|^2$, which has explicit solutions for a large range of f_i; and the updates of z requires solving $\min_{\mathbf{z}_{st},\mathbf{z}_{ts}} \|\mathbf{z}_{ts} - \mathbf{t}_1\|^2 + \|\mathbf{z}_{st} - \mathbf{t}_2\|^2 + \lambda\|\mathbf{z}_{ts} - \mathbf{z}_{st}\|$, which also has explicit solutions.

2.2. Our proposed approach

Alternatively, in this paper we propose another ADMM algorithm for solving (1), based on the reformulation as follows: we divide the set of edges $\mathcal{E}$ into $\mathcal{E}_0$ and $\mathcal{E}_1$, such that the set $\mathcal{E}_0$ does not contain two neighboring edges. We then solve the following optimization problem:

$$\underset{\substack{\{\mathbf{x}_i\}_{i\in\mathcal{V}},\\ \{\mathbf{z}_{st}\}_{(s,t)\in\mathcal{E}_1}}}{\operatorname{argmin}} \left(\sum_{i\in\mathcal{V}} f_i(\mathbf{x}_i) + \lambda\sum_{(s,t)\in\mathcal{E}_0}\|\mathbf{x}_s - \mathbf{x}_t\|\right) + \lambda\sum_{(s,t)\in\mathcal{E}_1}\|\mathbf{z}_{st} - \mathbf{z}_{ts}\|$$

$$\text{s.t. } \mathbf{x}_s = \mathbf{z}_{st} \text{ and } \mathbf{x}_t = \mathbf{z}_{ts} \text{ for all } (s,t) \in \mathcal{E}_1. \tag{8}$$

Let $\mathbf{u}_{st}, \mathbf{u}_{ts}$ be the dual variables for $\mathbf{x}_s - \mathbf{z}_{st}$ and $\mathbf{x}_t - \mathbf{z}_{ts}$ respectively, then the augmented Lagrangian is

$$\begin{aligned}\hat{L}_\rho(x, z, u) = &\sum_{i\in\mathcal{V}} f_i(\mathbf{x}_i) + \lambda \sum_{(s,t)\in\mathcal{E}_0}\|\mathbf{x}_s - \mathbf{x}_t\| \\ &+ \sum_{(s,t)\in\mathcal{E}_1}\Big(\lambda\|\mathbf{z}_{st} - \mathbf{z}_{ts}\| + \mathbf{u}_{st}^T(\mathbf{x}_s - \mathbf{z}_{st}) \\ &+ \mathbf{u}_{ts}^T(\mathbf{x}_t - \mathbf{z}_{ts}) + \frac{\rho}{2}\|\mathbf{x}_s - \mathbf{z}_{st}\|^2 + \frac{\rho}{2}\|\mathbf{x}_t - \mathbf{z}_{ts}\|^2\Big),\end{aligned} \tag{9}$$

where $\rho > 0$ is a parameter, and ADMM updates are

$$x^{(k+1)} = \underset{x}{\operatorname{argmin}}\, \hat{L}_\rho(x, z^{(k)}, u^{(k)}) \tag{10}$$

$$z^{(k+1)} = \underset{z}{\operatorname{argmin}}\, \hat{L}_\rho(x^{(k+1)}, z, u^{(k)}) \tag{11}$$

$$\mathbf{u}_{st}^{(k+1)} = \mathbf{u}_{st}^{(k)} + \rho(\mathbf{x}_s^{(k+1)} - \mathbf{z}_{st}^{(k+1)}) \tag{12}$$

$$\mathbf{u}_{ts}^{(k+1)} = \mathbf{u}_{ts}^{(k)} + \rho(\mathbf{x}_t^{(k+1)} - \mathbf{z}_{ts}^{(k+1)}). \tag{13}$$

While the update formula for x in (10) is similar to that in (4), it requires solving a slightly different problem due to the additional component $\lambda\sum_{(s,t)\in\mathcal{E}_0}\|\mathbf{x}_s - \mathbf{x}_t\|$. In particular, (10) can be divided into subproblems as follows

$$\begin{aligned}\underset{\mathbf{x}_s,\mathbf{x}_t\in\mathbb{R}^p}{\operatorname{argmin}}\; &f_s(\mathbf{x}_s) + f_t(\mathbf{x}_t) + \frac{\rho}{2}d_s\|\mathbf{x}_s\|^2 - \alpha_s^T\mathbf{x}_s \\ &+ \frac{\rho}{2}d_t\|\mathbf{x}_t\|^2 - \alpha_t^T\mathbf{x}_t + \lambda\|\mathbf{x}_s - \mathbf{x}_t\|,\end{aligned} \tag{14}$$

for some $\alpha_s, \alpha_t \in \mathbb{R}^p$ and $d_s, d_t \in \mathbb{R}$ with explicit expressions: d_s is the degree of the vertex s in the graph $(\mathcal{V}, \mathcal{E}_1)$ and $\alpha_s = \sum_{(s,t')\in\mathcal{E}_1}(\mathbf{u}_{st'} - \rho\mathbf{z}_{st'})$. For many choices of f_s and f_t, this problem has an explicit solution. For example, if f_s are squared functions in the form of $f_s(\mathbf{x}) = c_s\|\mathbf{x} - \mathbf{a}_s\|^2$, then the problem (14) has closed-form solutions by Lemma 2.

Intuitively, we expect our proposed algorithm would achieve a faster convergence rate than (2) because (a) (8) has fewer "dummy variables" in the form of $\mathbf{z}_{st}$ ($2|\mathcal{E}_1|$ instead of $2|\mathcal{E}|$); (b) (9) has fewer dual parameters than (8). This intuition is later verified by the computational analysis discussed in Section 3 and the numerical examples in Section 4.

2.3. Implementation

To reduce the computational cost, we provide an equivalent form of (8) in terms of ADMM as follows,

$$\underset{\substack{\{\mathbf{x}_i\}_{i\in\mathcal{V}},\\ \{\mathbf{z}_{st}\}_{(s,t)\in\mathcal{E}_1}}}{\operatorname{argmin}} \left(\sum_{i\in\mathcal{V}} f_i(\mathbf{x}_i) + \lambda\sum_{(s,t)\in\mathcal{E}_0}\|\mathbf{x}_s - \mathbf{x}_t\|\right) + \lambda\sum_{(s,t)\in\mathcal{E}_1}\|\mathbf{z}_{st}\|$$

$$\text{s.t. } \mathbf{z}_{st} = \mathbf{x}_s - \mathbf{x}_t. \tag{15}$$

and its Lagrangian reads

$$\tilde{L}_\rho(x,z,u)=\sum_{i\in\mathcal{V}} f_i(\mathbf{x}_i)+\lambda\sum_{(s,t)\in\mathcal{E}_0}\|\mathbf{x}_s-\mathbf{x}_t\|+\sum_{(s,t)\in\mathcal{E}_1}\Big(\lambda\|\mathbf{z}_{st}\|+\mathbf{u}_{st}^T(\mathbf{z}_{st}-\mathbf{x}_s+\mathbf{x}_t)+\frac{\rho}{2}\|\mathbf{z}_{st}-\mathbf{x}_s+\mathbf{x}_t\|^2\Big). \tag{16}$$

Then the preconditioned ADMM algorithm (Benning et al., 2016) can be applied,

$$x^{(k+1)}=\operatorname*{argmin}_x \tilde{L}_\rho(x,z^{(k)},u^{(k)})+\frac{\rho}{2}\sum_{(s,t)\in\mathcal{E}_1}\|\mathbf{x}_s+\mathbf{x}_t-\mathbf{x}_s^{(k)}-\mathbf{x}_t^{(k)}\|^2 \tag{17}$$

$$z^{(k+1)}=\operatorname*{argmin}_z \tilde{L}_\rho(x^{(k+1)},z,u^{(k)}) \tag{18}$$

$$\mathbf{u}_{st}^{(k+1)}=\mathbf{u}_{st}^{(k)}+\rho(\mathbf{z}_{st}^{(k+1)}-\mathbf{x}_s^{(k+1)}+\mathbf{x}_t^{(k+1)}). \tag{19}$$

The following lemma shows that the algorithm based on (17)–(19) is equivalent to that based on (10)–(13). The proof of Lemma 1 is included in Appendix.

Lemma 1. *For any $\rho_0>0$, the update formulas in (17)–(19) with $\rho=\rho_0$ are equivalent to those in (10)–(13) with $\rho=2\rho_0$.*

Compared with the updating formulas in (10)–(13), the computational costs of (17)–(19) are smaller since there are no "dummy variables" $\mathbf{z}_{ts}$ and its dual $\mathbf{u}_{ts}$. Specifically, problem (17) can be solved as follows:

$$(\mathbf{x}_s^{(k+1)},\mathbf{x}_t^{(k+1)})=\operatorname*{argmin}_{\mathbf{x}_s,\mathbf{x}_t} f_s(\mathbf{x}_s)+f_t(\mathbf{x}_t)+\lambda\|\mathbf{x}_s-\mathbf{x}_t\|+\rho(d_s\|\mathbf{x}_s\|^2+d_t\|\mathbf{x}_t\|^2)+\mathbf{x}_s^T\mathbf{t}_s^{(k)}+\mathbf{x}_t^T\mathbf{t}_t^{(k)}, \tag{20}$$

$$\mathbf{x}_i^{(k+1)}=\operatorname*{argmin}_{\mathbf{x}_i} f_i(\mathbf{x}_i)+\rho d_i\|\mathbf{x}_i\|^2+\mathbf{x}_i^T\mathbf{t}_i^{(k)}. \tag{21}$$

where the updates occur for any $(s,t)\in\mathcal{E}_0$ and any $i\in\mathcal{V}$ and does not belong to any edges in $\mathcal{E}_0$. The vectors $\mathbf{t}_i^{(k)}$ are defined as $\mathbf{t}_i^{(k)}=\sum_{j:(i,j)\in\mathcal{E}_1}[-(\mathbf{u}_{ij}^{(k)}+\rho\mathbf{z}_{ij}^{(k)})-\rho(\mathbf{x}_i^{(k)}+\mathbf{x}_j^{(k)})]+\sum_{j:(j,i)\in\mathcal{E}_1}[(\mathbf{u}_{ji}^{(k)}+\rho\mathbf{z}_{ji}^{(k)})-\rho(\mathbf{x}_i^{(k)}+\mathbf{x}_j^{(k)})]$. The implementation details based on (17)–(19) are described in Algorithm 1.

Algorithm 1 Proposed algorithm based on (17)–(19).

1: **Input:** Graph $(\mathcal{V},\mathcal{E})$ and its partition $\mathcal{E}=\mathcal{E}_0\cup\mathcal{E}_1$ ($\mathcal{E}_0$ has not neighboring edges); loss functions $\{f_i\}_{i\in\mathcal{V}}$; parameters ρ and λ.
2: **for** $k=1,2,\cdots$ **do**
3: Update $(\mathbf{x}_s^{(k+1)},\mathbf{x}_t^{(k+1)})$ using (20) and (21).
4: For any $(s,t)\in\mathcal{E}_1$, update

$$\mathbf{z}_{st}^{(k+1)}=\text{threshold}\Big(\mathbf{x}_s^{(k+1)}-\mathbf{x}_t^{(k+1)}-\frac{\mathbf{u}_{st}^{(k)}}{\rho},\frac{\lambda}{\rho}\Big)$$

$$\mathbf{u}_{st}^{(k+1)}=\mathbf{u}_{st}^{(k)}+\rho(\mathbf{z}_{st}^{(k+1)}-\mathbf{x}_s^{(k+1)}+\mathbf{x}_t^{(k+1)}).$$

5: stop until $\|\mathbf{x}_i^{(k+1)}-\mathbf{x}_i^{(k)}\|<\varepsilon$
6: **end for**
7: **Output:** $\hat{\mathbf{x}}_i=\mathbf{x}_i^{(k+1)}$.

As the performance of Algorithm 1 depends on how $\mathcal{E}$ is decomposed, $\mathcal{E}_0$ is called matching in graph theory and there are numerous algorithms for finding a matching within a graph. It is related to the classic problem of graph matching in graph theory. In practice, we choose greedy algorithm to find $\mathcal{E}_0$. First, label all edges in some arbitrary order and $\mathcal{E}_0$ be an empty set. Second, cycle once through each edge and add it to $\mathcal{E}_0$ if it is not neighboring any existing edge in $\mathcal{E}_0$.

3. Computational analysis

While the ADMM algorithm and its convergence rate has been well studied, many existing works require very strong conditions on the objective which are not necessarily satisfied in our problem. For example, Sections 4–5 in Goldstein et al. (2014) proved that the standard ADMM algorithm for $f(\mathbf{x})+g(\mathbf{z})$ subject to $\mathbf{Ax}+\mathbf{Bz}=\mathbf{c}$ converges with rate $O(1/k)$, and an accelerated version converges with rate $O(1/k^2)$ in terms of the residue of the algorithm, when both f and g are strongly convex and ρ is appropriately chosen. However, it cannot be directly applied to our case since g in (8) is not strongly convex. In addition, Theorem 7 in Nishihara et al. (2015) proved that in minimizing $f(\mathbf{x})+g(\mathbf{z})$ subject to $\mathbf{Ax}+\mathbf{Bz}=\mathbf{c}$, if g is convex, f is strongly convex with convex parameter m and ∇f is Lipschitz continuous with parameter L, then the algorithm has linear convergence rate when ρ is appropriately chosen. But again this result cannot be directly applied to our case, since in (8), ∇f is not Lipschitz continuous. The authors of Deng and Yin (2016) proved the global convergence for ADMM under the assumptions of strong convexity and Lipschitz gradient on one of the two functions, along with certain rank assumptions on $\mathbf{A}$ and $\mathbf{B}$, but in (8), neither ∇f or ∇g is Lipschitz continuous. Similarly, a tight linear convergence rate bound was proved in Giselsson (2017) under the assumption that one of the two operators is strongly convex and the other is Lipschitz continuous, which again does not apply to (8). On the other hand, some previous works can handle the settings of our problem, but the established convergence rate is not optimal. For example, a sub-linear convergence rate of $O(1/\sqrt{k})$ is obtained in Guo et al. (2017) for the Douglas–Rachford Splitting Method for solving $\min_x f(x)+g(x)$ when f is strongly convex and g is weakly convex.

The only reference that proves the global linear convergence of our algorithm might be (Yang and Han, 2016), which proves a global linear convergence when both ∂f and ∂g are piecewise linear multifunctions. However, they only find a bound on the linear convergence rate, and this rate is implicitly defined. In this paper, we provide an exact convergence rate of the proposed algorithm by developing a general theory on the local convergence of ADMM. Notably, this result can be applied to analyze both the convergence rate of our algorithm and that of the network lasso algorithm. The proof of the following theorem is deferred to Appendix.

Theorem 1 (*Local Convergence Rate of ADMM*). *Considering the problem of minimizing* $f_1(\mathbf{x}_1) + f_2(\mathbf{x}_2)$ *subject to* $\mathbf{A}_1\mathbf{x}_1 + \mathbf{A}_2\mathbf{x}_2 = \mathbf{b}$. *Assume that around the solution* $(\mathbf{x}^*, \mathbf{y}^*)$, $\partial f_1(\mathbf{x}) = \mathbf{C}_1\mathbf{x} + \mathbf{c}_1$ *and* $\partial f_2(\mathbf{x}) = \mathbf{C}_2\mathbf{x} + \mathbf{c}_2$, *then the local convergence rate of the ADMM algorithm is* $O(c(\rho)^k)$, *where* k *is the number of iterations and* $c(\rho)$ *is the largest real components among all eigenvalue of*

$$\frac{1}{2}[(\mathbf{I} - 2(\mathbf{I} + \rho\mathbf{A}_2\mathbf{C}_2^{-1}\mathbf{A}_2^T)^{-1})(\mathbf{I} - 2(\mathbf{I} + \rho\mathbf{A}_1\mathbf{C}_1^{-1}\mathbf{A}_1^T)^{-1}) + \mathbf{I}].$$

Remark. We note that Theorem 1 is similar to França and Bento (2017) in the sense that both investigate the exact local convergence rate. However, the objective function in França and Bento (2017) is quadratic and has no ℓ_2 component as in (8).

The convergence rate of Algorithm 1 follows from Theorem 1 with

$$f_1(\{\mathbf{x}_i\}_{i\in\mathcal{V}}) = \sum_{i\in\mathcal{V}} f_i(\mathbf{x}_i) + \lambda \sum_{(s,t)\in\mathcal{E}_0} \|\mathbf{x}_s - \mathbf{x}_t\|$$
$$f_2(\{\mathbf{z}_{st}\}_{(s,t)\in\mathcal{E}_1}) = \lambda \sum_{(s,t)\in\mathcal{E}_1} \|\mathbf{z}_{st} - \mathbf{z}_{ts}\|.$$

That is, $\mathbf{x}_1$ in Theorem 1 is replaced by $\{\mathbf{x}_i\}_{i\in\mathcal{V}}$, $\mathbf{x}_2$ in Theorem 1 is replaced by $\{\mathbf{z}_{st}\}_{(s,t)\in\mathcal{E}_1}$, and $\mathbf{A}_1\mathbf{x}_1 + \mathbf{A}_2\mathbf{x}_2 = \mathbf{b}$ is replaced by $\mathbf{x}_s = \mathbf{z}_{st}$ and $\mathbf{x}_t = \mathbf{z}_{ts}$ for all $(s,t) \in \mathcal{E}_1$. Therefore, we have $\mathbf{A}_1 \in \mathbb{R}^{np\times 2p|\mathcal{E}_1|}$, defined such that $\mathbf{A}_1(2i-1, s_i) = \mathbf{I}_{p\times p}$ and $\mathbf{A}_1(2i, t_i) = \mathbf{I}_{p\times p}$ if (s_i, t_i) is the ith edge in $\mathcal{E}_1$, and $\mathbf{A}_2 = -\mathbf{I}_{2p|\mathcal{E}_1|\times 2p|\mathcal{E}_1|}$. The matrix $\mathbf{C}_1 \in \mathbb{R}^{pn\times pn}$ can be generated by the following three steps. First, the (i,i)th $p\times p$ block is given by

$$\mathbf{C}_1(i,i) = \text{Hessian} f_i(\mathbf{x}_i^*)$$

Second, for $(i,j) \in \mathcal{E}_0$ we let $\mathbf{T}(i,j) = \frac{1}{\|\mathbf{x}_i^* - \mathbf{x}_j^*\|}\mathbf{I} - \frac{1}{\|\mathbf{x}_i^* - \mathbf{x}_j^*\|^3}(\mathbf{x}_i^* - \mathbf{x}_j^*)(\mathbf{x}_i^* - \mathbf{x}_j^*)^T$ if $\mathbf{x}_i^* \neq \mathbf{x}_j^*$, and $\mathbf{T}(i,j) = \infty\mathbf{I}$ if $\mathbf{x}_i^* = \mathbf{x}_j^*$. Third, we update the $(i,i),(i,j),(j,i),(j,j)$th $p\times p$ blocks of $\mathbf{C}_1$ by

$$\mathbf{C}_1(i,i) \leftarrow \mathbf{C}_1(i,i) + \mathbf{T}(i,j),$$
$$\mathbf{C}_1(j,j) \leftarrow \mathbf{C}_1(j,j) + \mathbf{T}(i,j),$$
$$\mathbf{C}_1(i,j) \leftarrow \mathbf{C}_1(i,j) - \mathbf{T}(i,j),$$
$$\mathbf{C}_1(j,i) \leftarrow \mathbf{C}_1(j,i) - \mathbf{T}(i,j).$$

The matrix $\mathbf{C}_2 \in \mathbb{R}^{2p|\mathcal{E}_1|\times 2p|\mathcal{E}_1|}$ is generated as follows: for the ith edge in $\mathcal{E}_1$, (s_i, t_i), the (i,i)th $2p\times 2p$ block of $\mathbf{C}_2$ is given by $[\mathbf{T}(s_i,t_i), -\mathbf{T}(s_i,t_i); -\mathbf{T}(s_i,t_i), \mathbf{T}(s_i,t_i)]$. The remaining $2p\times 2p$ blocks are all zero matrices.

Note that the network lasso algorithm is equivalent to Algorithm 1 with $\mathcal{E}_0 = \emptyset$ and $\mathcal{E}_1 = \mathcal{E}$, this result can also be used to analyze the convergence rate of the network lasso algorithm. While it is difficult to compare their convergence rates in general due to the complexities of $\mathbf{A}_i$ and $\mathbf{C}_i$, we can calculate the convergence rate numerically for some specific examples. Here we assume two cases as follows:

1. Let $(\mathcal{V}, \mathcal{E})$ be the one-dimensional chain graph defined by $\mathcal{V} = \{1, 2, \dots, 100\}$ and $\mathcal{E} = \{(k, k+1) \mid 1 \le k \le 99\}$, and the partition such that $\mathcal{E}_0 = \{(1,2),(3,4),\dots\}$ and $\mathcal{E}_1 = \{(2,3),(4,5),\dots\}$. In addition, $\hat{\mathbf{x}}_i \neq \hat{\mathbf{x}}_{i+1}$ when $i = 9, 18\cdots, 99$, $\lambda = 1$.
2. Let $(\mathcal{V}, \mathcal{E})$ be the two-dimensional grid graph of size 10×10 and its partition $\mathcal{E} = \mathcal{E}_0 \cup \mathcal{E}_1$ as visualized in Fig. 3. In addition, $\hat{\mathbf{x}}$ has the sparsity pattern such that if we index the 100 vertices by $(i,j)_{1\le i,j\le 10}$, then $\hat{\mathbf{x}}(i,j)$ has the same value if $(i-5)^2 + (j-5)^2 \le 10$ and has another value otherwise, and $\lambda = 1$.

The comparison of corresponding $c(\rho)$ defined in Theorem 1 for Algorithm 1 and the network lasso are visualized in Fig. 1, where the y-axis represents $-\log(c(\rho))$, in which larger values means faster convergence. From Fig. 1, Algorithm 1 consistently has a larger $-\log(c(\rho))$, which implies that it has a faster convergence rate than the network lasso algorithm.

3.1. Computational cost

In this section, we compare the computational cost per iteration of Algorithm 1 and the network lasso algorithm (Hallac et al., 2015) in a common case that $f_i(\mathbf{x}_i) = \|\mathbf{x}_i - \mathbf{y}_i\|^2$. Under such setting, the solutions of (20) in Algorithm 1 can be characterized by Lemma 2.

Lemma 2. *For any* $\mathbf{a}, \mathbf{b} \in \mathbb{R}^p$,

$$\operatorname*{argmin}_{\mathbf{x},\mathbf{y}\in\mathbb{R}^p} c_1\|\mathbf{x} - \mathbf{a}\|^2 + c_2\|\mathbf{y} - \mathbf{b}\|^2 + \lambda\|\mathbf{x} - \mathbf{y}\|$$

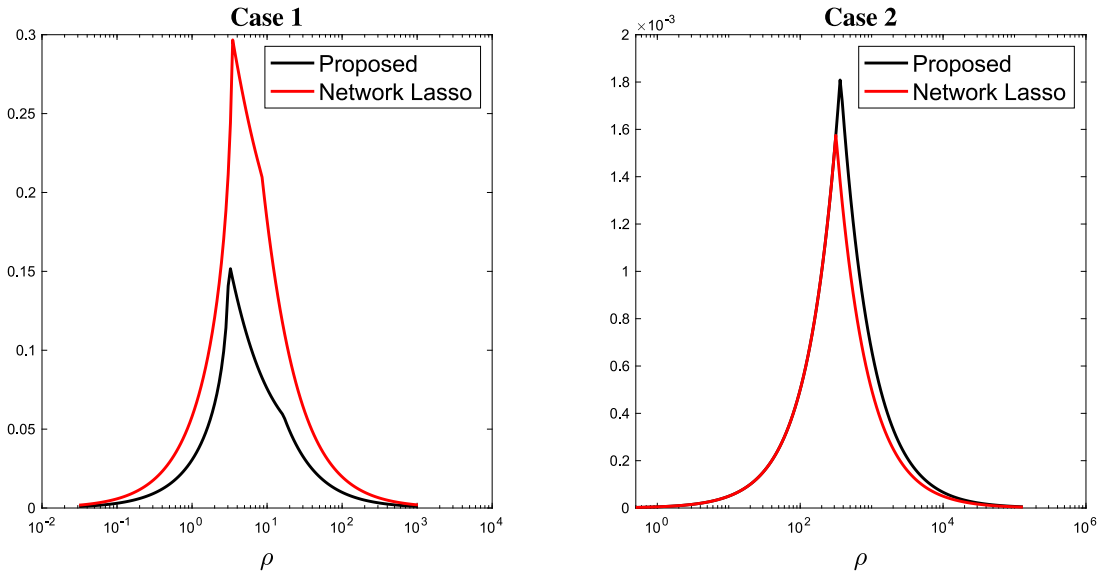

Fig. 1. Comparison of the theoretical local convergence rates between Algorithm 1 and the network lasso.

$$= \begin{cases} \left(\frac{c_1\mathbf{a}+c_2\mathbf{b}}{c_1+c_2}, \frac{c_1\mathbf{a}+c_2\mathbf{b}}{c_1+c_2}\right), & \text{if } 2c_1c_2\|\mathbf{a}-\mathbf{b}\| \le (c_1+c_2)\lambda \\ \left(\mathbf{a} - \frac{\lambda}{2c_1}\frac{\mathbf{a}-\mathbf{b}}{\|\mathbf{a}-\mathbf{b}\|}, \mathbf{b} - \frac{\lambda}{2c_2}\frac{\mathbf{b}-\mathbf{a}}{\|\mathbf{b}-\mathbf{a}\|}\right), & \text{otherwise.} \end{cases}$$

Now let us investigate the computational complexity per iteration of Algorithm 1, by keeping track of the multiplications of a scalar and a vector of $\mathbb{R}^p$ (denoted as multiplications) and the additions of two vectors of $\mathbb{R}^p$ (denoted as additions). In particular, the calculation of $\mathbf{t}_s/(1+\rho_s)$ in step 3 requires $2|\mathcal{E}_1|+n$ multiplications and $2|\mathcal{E}_1|+n$ additions, and solving (20) requires an additional cost of at most $2|\mathcal{E}_0|$ multiplications and $2|\mathcal{E}_0|$ additions, and $|\mathcal{E}_0|$ operations of finding the norm of a vector of length p and $|\mathcal{E}_0|$ operations of comparing two scalars. Solving (21) requires $3|\mathcal{E}_1|$ additions, $|\mathcal{E}_1|$ multiplications and $|\mathcal{E}_1|$ comparisons. Step 4 requires $3|\mathcal{E}_1|$ additions and $|\mathcal{E}_1|$ multiplications. Therefore the total computational cost is $p(13|\mathcal{E}_1|+7|\mathcal{E}_0|+2n) = p(6|\mathcal{E}_1|+9n) \le 15np$. On the other hand, note that the network lasso algorithm is equivalent to the case where $\mathcal{E}_0 = \emptyset$ and $\mathcal{E}_1 = \mathcal{E}$, we may also compute its computational cost per iteration, which would be $15np$. It is clear that whenever $|\mathcal{E}_1| < |\mathcal{E}|$, Algorithm 1 will always have a smaller computational cost per iteration compared to the network lasso.

3.2. Comparison with other works

Decomposing a graph into edges or paths has been used in existing literature. However, we remark that our approach is different from previous works. For instance, Tansey and Scott (2015) decomposes the graph into a set of trails (this idea is also explored in Barbero and Sra (2018) for the two-dimensional grid graph), and then apply existing algorithms to solve each problem. In particular, it decomposes $\mathcal{E}$ into K sets $\bigcup_{k=1}^K \mathcal{E}_k$ such that for each $1 \le k \le K$, $\mathcal{E}_k$ is a trail, which is a walk in which all the edges are distinct. By writing the optimization problem as

$$\min_{\mathbf{x},\mathbf{z}} \sum_{i\in\mathcal{V}} f_i(\mathbf{x}_i) + \lambda \sum_{1\le k\le K} \sum_{(s,t)\in\mathcal{E}_k} \|\mathbf{z}_{\mathcal{E}_k,s} - \mathbf{z}_{\mathcal{E}_k,t}\|,$$
$$\text{s.t. } \mathbf{z}_{\mathcal{E}_k,s} = \mathbf{x}_s, \ \forall 1 \le k \le K \text{ and } s \text{ in some edge of } \mathcal{E}_k,$$

then the ADMM algorithm can be used to update $\mathbf{x}$ and $\mathbf{z}$ alternatively. For the case $p = 1$, the update for $\mathbf{z}$ in each trail can be solved efficiently using the dynamic programming approach in Johnson (2013). We remark that there are two main differences between their method and ours: First, while ADMM algorithm requires decomposing the objective function into two components, they follow the natural decomposition by the loss function f_i and the total variation penalty term; while our partition is different and based on graph decomposition. A comparison is summarized in Table 1. In this sense, Barbero and Sra (2018) and Tansey and Scott (2015) are verysimilar to network lasso, but with the subproblem for the second component solved by graph decomposition and dynamic programming. Second, these methods apply the dynamic programming approach (Johnson, 2013), which only works for the case where $\mathbf{x}_i$ are scalars (i.e., $p = 1$). In comparison, our method can handle the multivariate observations, i.e., $\mathbf{x}_i \in \mathbb{R}^p$ with $p > 1$.

In fact, when $p = 1$, the idea in this work can be combined with the idea in Barbero and Sra (2018) and Tansey and Scott (2015) as follows: first decompose $\mathcal{E}$ into sets $(\bigcup_{k=1}^{K_1} \mathcal{E}_k^1) \cup (\bigcup_{\ell=1}^{K_0} \mathcal{E}_\ell^0)$, such that all $\mathcal{E}_\ell^0$ and $\mathcal{E}_k^1$ are disjoint trails. Then we may write the

Table 1
Comparison of decomposition of the objective function (1) in various ADMM algorithms. The second row refers Networks Lasso (Hallac et al., 2015), and Barbero and Sra (2018), Tansey and Scott (2015), Zhu (2017) and Ramdas and Tibshirani (2015).

	First component	Second component
Proposed	$\sum_{i\in\mathcal{V}} f_i(\mathbf{x}_i) \quad +\lambda \sum_{(s,t)\in\mathcal{E}_1} \|\mathbf{x}_t - \mathbf{x}_s\|$	$\lambda \sum_{(s,t)\in\mathcal{E}_1} \|\mathbf{x}_t - \mathbf{x}_s\|$
Others	$\sum_{i\in\mathcal{V}} f_i(\mathbf{x}_i)$	$\lambda \sum_{(s,t)\in\mathcal{E}} \|\mathbf{x}_t - \mathbf{x}_s\|$

optimization problem as

$$\min_{\mathbf{x},\mathbf{z}} \Big(\sum_{i\in\mathcal{V}} f_i(\mathbf{x}_i) + \lambda \sum_{1\le k\le K_0} \sum_{(s,t)\in\mathcal{E}_k^0} \|\mathbf{x}_s - \mathbf{x}_t\| \Big) + \lambda \sum_{1\le k\le K_1} \sum_{(s,t)\in\mathcal{E}_k^1} \|\mathbf{z}_{\mathcal{E}_k^1,s} - \mathbf{z}_{\mathcal{E}_k^1,t}\|,$$

$$\text{s.t. } \mathbf{z}_{\mathcal{E}_k^1,s} = \mathbf{x}_s,$$

for all $1 \le k \le K-1$ and s in some edge of $\mathcal{E}_k^1$, which would lead to another ADMM algorithm for solving (1) when $p = 1$ (i.e., the problem considered in Tansey and Scott (2015)) and the dynamic programming method (Johnson, 2013) can be applied. However, we suspect that this algorithm is faster than Algorithm 1 when $p = 1$, but we will leave it as a possible future investigation.

We remark that there are other methods for fused lasso and its related problems, including Zhu (2017) and Ramdas and Tibshirani (2015). However, both of them are still based on the standard decomposition of loss function and regularizer, and only apply to the setting $p = 1$.

4. Numerical experiments

In this section, Algorithm 1 will be compared with the network lasso for solving the graph-fused lasso models under various scenarios. We measure the convergence by the difference between the objective value at iteration k and the optimal objective value. We remark that all ADMM algorithms require an augmented Lagrangian parameter ρ, and the algorithms would converge slowly when ρ is too large or too small. While there have been many works on the choice of ρ (for example, a simple varying penalty strategy based on residual balancing is suggested in Section 3.4.1 of Boyd et al. (2011), and another choice based on the Barzilai–Borwein spectral method for gradient descent is proposed in Xu et al. (2017)), there is no consensus on the optimal strategy of the choice of ρ. As a result, we will test the performance of the algorithms on a range of ρ and the optimal ρ is picked inside the chosen range.

4.1. Synthetic simulations

We first test our algorithm on the one-dimensional chain graph. Following Zhu (2017), we use the model that

$$\mathbf{y}_i^* = \begin{cases} [1,1], \text{if } 1 \le i \le 11 \\ [-1,1], \text{if } 12 \le i \le 22 \\ [2,2], \text{if } 1 \le 23 \le 33 \\ [-1,-1], \text{if } 34 \le i \le 44 \\ [0,0], \text{if } i \ge 45. \end{cases}$$

We generate $\mathbf{y}_i = \mathbf{y}_i^* + 0.5 * \mathcal{N}(\mathbf{0}, \mathbf{I}_{2\times 2})$ and $\lambda \in \{0.1, 1\}$. The loss function $f_i(\mathbf{x}_i) = \|\mathbf{x}_i - \mathbf{y}_i\|^2$ is used, so we can apply Lemma 2 to obtain the closed-form solutions in Algorithm 1. Comparison between Algorithm 1 and the network lasso in terms of convergence speed under various choices of ρ are shown in Fig. 2. The figures indicate that for both choices of λ, Algorithm 1 always performs better with a good choice of ρ. In fact, if ρ is chosen to be the optimal values for both algorithms, Algorithm 1 converges twice as fast as the network lasso.

For the second simulation, we generate a set of data on a 10 by 10 grid graph; the true values at the center vertices with radius 2 are taken as $(0.5, 0.5, 0.5) \in \mathbb{R}^3$ and others are taken as $\mathbf{0}$, and we add noise of $0.2 * \mathcal{N}(\mathbf{0}, \mathbf{I}_{3\times 3})$. We present a visualization of this grid and a natural choice of $\mathcal{E}_0$ in Fig. 3. Since both of Algorithm 1 and network lass work for a general convex loss function f_i, we use the package `cvx` when updating x in Algorithm 1 (network lasso as well has this feature). The convergence times are shown in Fig. 4, which shows that Algorithm 1 takes a comparable or shorter time to reach the convergence than the network lasso algorithm. Combining it with the fact that Algorithm 1 has a smaller computational complexity per iteration, this implies the numerical superiority of Algorithm 1.

4.2. Real data example

A natural application of the graph fused lasso estimator is 2D image denoising (Chopra and Lian, 2010; Rudin et al., 1992) as the estimator can be used to penalize the differences between neighboring pixels in an image. Suppose that $f_i(\mathbf{x}_i) = (\mathbf{y}_i - \mathbf{x}_i)^2$ where $\{\mathbf{y}_i\}_{i\in\mathcal{V}}$ represents the pixels in a noisy image and $\mathcal{E}$ is a grid graph as shown in Fig. 3 such that $\sum_{(s,t)\in\mathcal{E}} \|\mathbf{x}_t - \mathbf{x}_s\|$ measures the sum

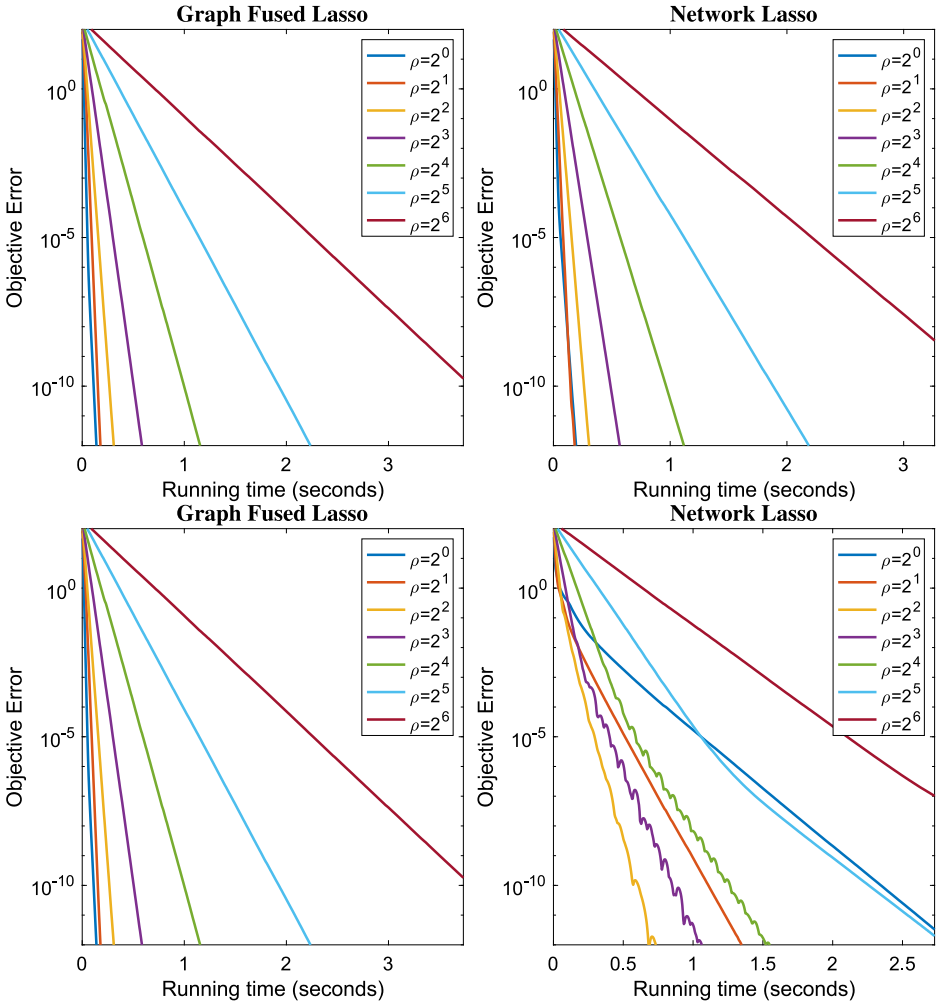

Fig. 2. Comparison of the convergence rates of Algorithm 1 and Network-Lasso Algorithm for 1D chain graph with $\lambda = 0.1$ (first row) and $\lambda = 1$ (second row).

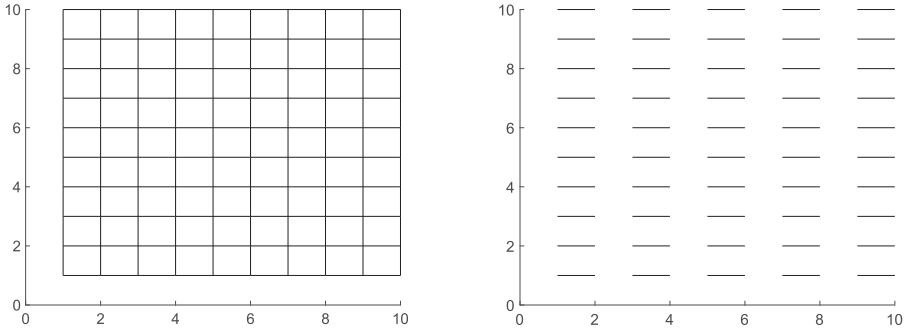

Fig. 3. Visualization of a 10 by 10 grid graph (left) and an example of $\mathcal{E}_0$ (right).

of differences between neighboring pixels. Then the graph fused lasso estimator (1) can be applied to denoise images as a natural image should be locally smooth. In this section, we apply our Algorithm 1 to two image-denoising examples, and compare it with the Network Lasso Algorithm (Hallac et al., 2015) and the Trail Decomposition Algorithm (TrailDecomp) (Tansey and Scott, 2015). While all three algorithms solve the graph fused lasso problem (1) based on ADMM, the Network Lasso algorithm is not based on graph decomposition and the TrailDecomp is based on a different graph decomposition in the sense that the graph $\mathcal{E}$ is decomposed

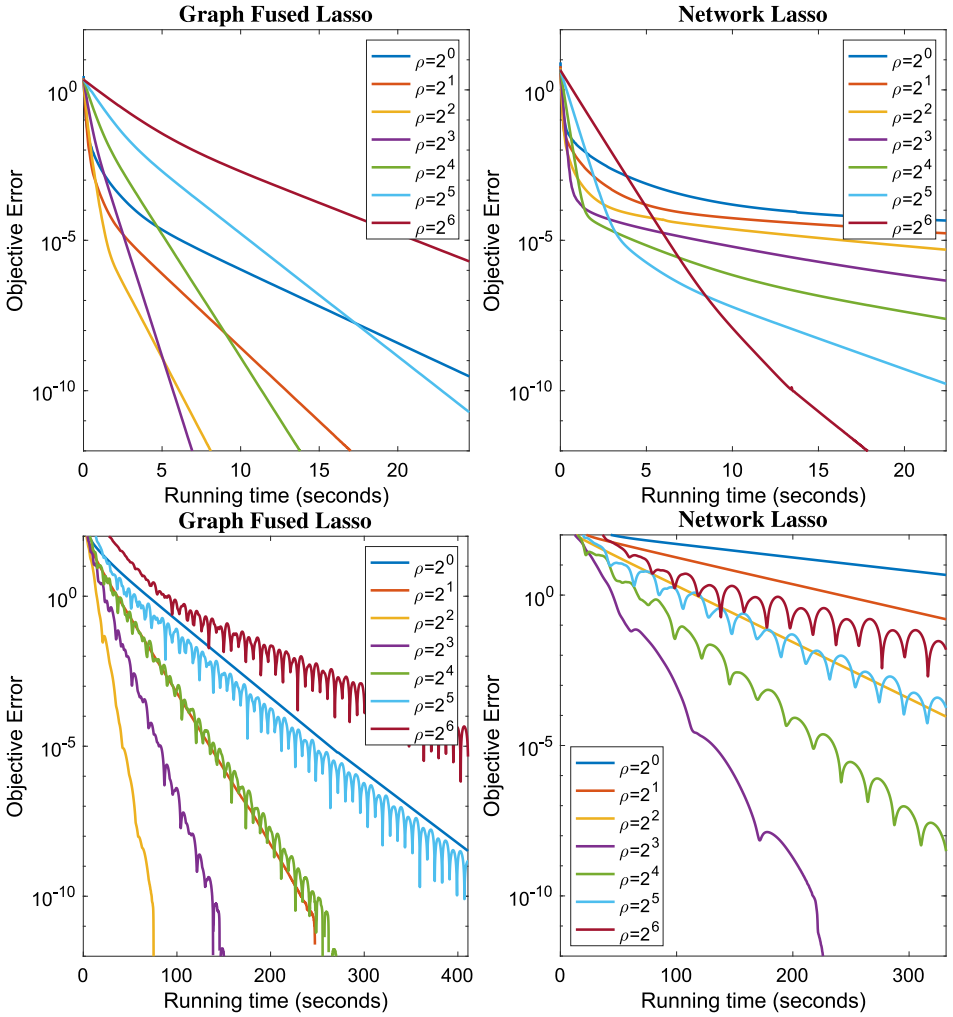

Fig. 4. Comparison of the convergence rates of Algorithm 1 and Network-Lasso Algorithm under the 2D grid graph setting with $\lambda = 0.5$ (first row) and $\lambda = 10$ (second row).

Table 2
MSEs of Algorithm 1, Network Lasso and TrailDecomp shown in Figs. 5 and 6.

	Proposed	Network Lasso	TrailDecomp
Baboon	0.0544	0.0544	0.0567
Butterfly	0.1035	0.1035	N/A

into a set of "trails", a sequence of vertices that are connected by edges in $\mathcal{E}$. We note that TrailDecomp does not apply to the case $p = 1$.

The two examples are based on the baboon image and the butterfly images as shown in Figs. 5 and 6. We add i.i.d. Gaussian noises of size $N(0, \sigma^2)$ to each pixel and then apply algorithms to denoise the two images. The size of the baboon image is 64×64 and each pixel is represented as a scalar based on the grayscale, and the size of the butterfly image is 128×128 and each pixel is represented as a vector of length 3 in the RGB color model. The comparisons of Algorithm 1, Network Lasso Algorithm, Trail Decomposition Algorithm are displayed in Figs. 5 and 6, and we do not include TrailDecomp in the butterfly example as it does not apply to the case $p = 3$. In addition, we measure the quality of the denoising by $\text{Average}_{i\in\mathcal{G}}\|\hat{\mathbf{x}}_i - \mathbf{x}_i\|^2$, the mean square error (MSE), and report the mean MSEs over 100 runs in Table 2. All algorithms stop at 50 iterations. It shows that our algorithm achieves the same error as Network Lasso and both of them have smaller MSEs than TrailDecomp. Both of our algorithm and Network Lasso Estimator converge to the global minimizer to the graph fused lasso problem (1) while TrailDecomp does not converge within 50 iterations.

Lastly we model and test a real world problem in a reasonably large, geographically-defined underlying graph. The data comes from the police reports made publicly available by the city of Chicago, from 2001 until the present (Chicago Police Department

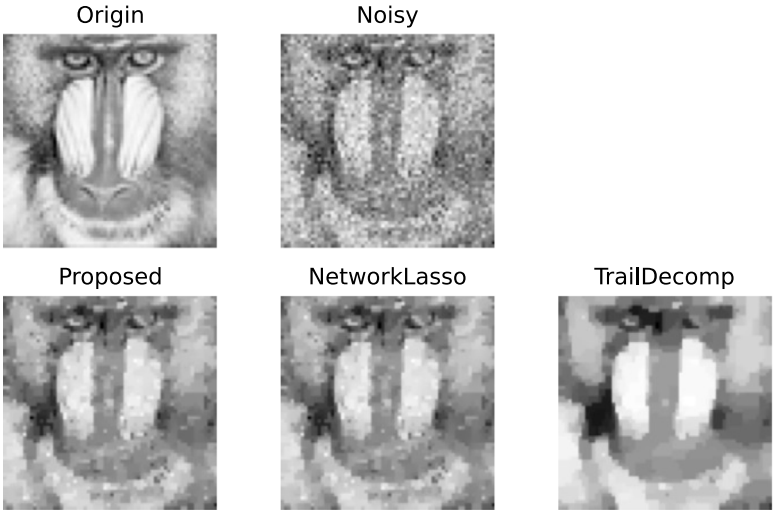

Fig. 5. Comparison of Algorithm 1, Network Lasso Algorithm, TrailDecomp for image denoising. The noise level is $\sigma = 0.1$ and λ is set to be 0.08.

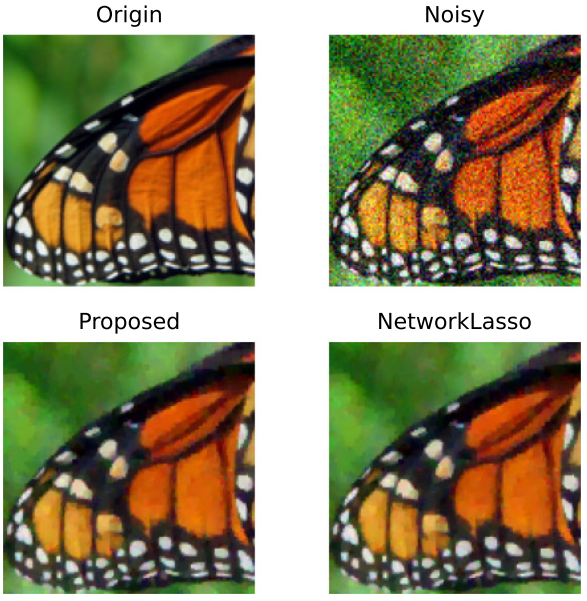

Fig. 6. Comparison of Algorithm 1 and Network Lasso Algorithm for image denoising. The noise level is $\sigma = 0.15$ and λ is set to be 0.25. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

2014) and is available as the supplementary files of Arnold and Tibshirani (2016). In this dataset, the vertices represent the census blocks and the edges represent the neighboring blocks, and $x_i \in [0, 1]$ denotes the burglaries occurring rate in block i. The original graph has 2162 vertices, 6995 edges and by running the greedy algorithm of the graph matching in the end of Section 2, we obtain $\mathcal{E}_0$ that contains 1025 edges with no sharing vertices. The convergence rates are shown in Fig. 7, which shows that Algorithm 1 converges faster than the network lasso algorithm. More results and discussions are in Appendix.

5. Conclusions

This paper proposes a new ADMM algorithm for solving graphic fused lasso, based on a novel method of dividing the objective function and into two components based on graph decomposition. According to the theoretical analysis and numerical verification, the proposed algorithm enjoys a smaller complexity per iteration and converges faster comparing with the standard ADMM algorithm of the graph-fused lasso (GFL). Because of the universality of GFL, our method can be applied to a wide range of optimization problems, and simulations show that our advantage is significant for the chain graph. However, finding a good graph decomposition

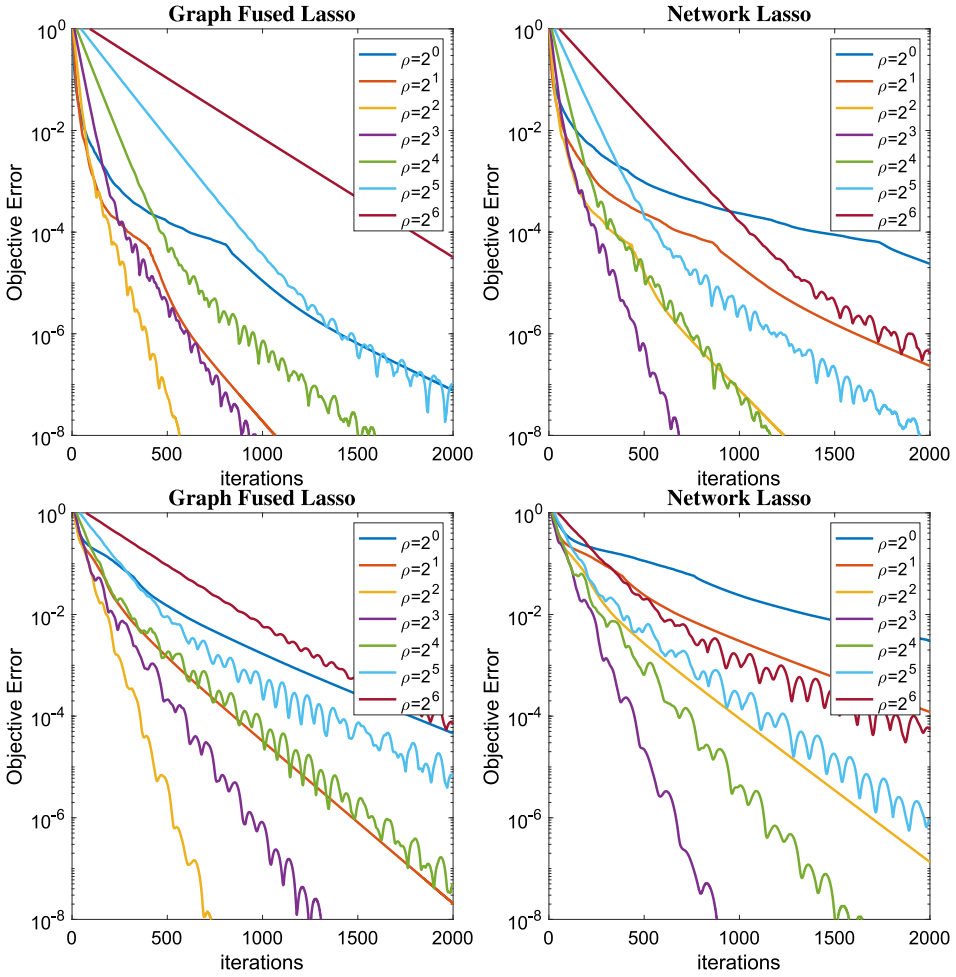

Fig. 7. Comparison of the convergence rates for the Chicago crime dataset with $\lambda = 0.05$ (first row) and $\lambda = 0.25$ (second row).

is important to the success of the proposed algorithm. While there are existing approaches for decomposing the chain graph and the grid graph, it remains a future direction to find the optimal graph decomposition for general graphs. Moreover, our idea can be applied to broader settings that are not covered by (1). For example, the trending filtering (Ramdas and Tibshirani, 2015) can be considered as a generalization of (1) to hypergraphs, and we may divide the regularization term into two components $\sum_{i=1}^{n} f_i(\mathbf{x}_i) + \|\mathbf{x}_1 - 2\mathbf{x}_2 + \mathbf{x}_3\| + \|\mathbf{x}_4 - 2\mathbf{x}_5 + \mathbf{x}_6\| + \cdots$ and $\|\mathbf{x}_2 - 2\mathbf{x}_3 + \mathbf{x}_4\| + \|\mathbf{x}_3 - 2\mathbf{x}_4 + \mathbf{x}_5\| + \|\mathbf{x}_5 - 2\mathbf{x}_6 + \mathbf{x}_7\| + \|\mathbf{x}_6 - 2\mathbf{x}_7 + \mathbf{x}_8\| + \cdots$. The analysis of its performance and the comparison with standard algorithms would be another possible future direction.

CRediT authorship contribution statement

Feng Yu: Writing – original draft, Visualization, Software. **Archer Yi Yang:** Methodology, Writing – review & editing. **Teng Zhang:** Conceptualization, Methodology, Funding acquisition, Writing – original draft.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported by the National Science Foundation, ATD-2318926.

Appendix A. Technical proofs

A.1. Proof of Lemma 1

Let us consider the problem

$$\underset{\substack{\{\mathbf{x}_i\}_{i\in\mathcal{E}},\\ \{\mathbf{z}_{st}\}_{(s,t)\in\mathcal{E}_1}}}{\operatorname{argmin}} \left(\sum_{i\in\mathcal{E}} f_i(\mathbf{x}_i) + \lambda \sum_{(s,t)\in\mathcal{E}_0} \|\mathbf{x}_s - \mathbf{x}_t\|\right) + \lambda \sum_{(s,t)\in\mathcal{E}_1} \|\mathbf{z}_{st}\| \tag{A.1}$$
$$\text{s.t. } \mathbf{z}_{st} = \mathbf{x}_s - \mathbf{x}_t,\ \mathbf{z}_{ts} = \mathbf{x}_s + \mathbf{x}_t.$$

and its associated Lagrangian

$$\begin{aligned}\bar{L}_\rho(x,z,u) = &\sum_{i\in\mathcal{V}} f_i(\mathbf{x}_i) + \lambda \sum_{(s,t)\in\mathcal{E}_0} \|\mathbf{x}_s - \mathbf{x}_t\| + \\ &\sum_{(s,t)\in\mathcal{E}_1} \Big(\lambda\|\mathbf{z}_{st}\| + \mathbf{u}_{st}^T(\mathbf{z}_{st} - \mathbf{x}_s + \mathbf{x}_t) + \mathbf{u}_{ts}^T(\mathbf{z}_{ts} - \mathbf{x}_s - \mathbf{x}_t) \\ &+ \frac{\rho}{2}\|\mathbf{z}_{st} - \mathbf{x}_s + \mathbf{x}_t\|^2 + \frac{\rho}{2}\|\mathbf{z}_{ts} - \mathbf{x}_s - \mathbf{x}_t\|^2\Big),\end{aligned} \tag{A.2}$$

as well as the ADMM algorithm of

$$x^{(k+1)} = \underset{x}{\operatorname{argmin}}\ \bar{L}_\rho(x, z^{(k)}, u^{(k)}) \tag{A.3}$$
$$z^{(k+1)} = \underset{z}{\operatorname{argmin}}\ \bar{L}_\rho(x^{(k+1)}, z, u^{(k)}) \tag{A.4}$$
$$\mathbf{u}_{st}^{(k+1)} = \mathbf{u}_{st}^{(k)} + \rho(\mathbf{z}_{st}^{(k+1)} - \mathbf{x}_s^{(k+1)} + \mathbf{x}_t^{(k+1)}) \tag{A.5}$$
$$\mathbf{u}_{ts}^{(k+1)} = \mathbf{u}_{ts}^{(k)} + \rho(\mathbf{z}_{ts}^{(k+1)} - \mathbf{x}_s^{(k+1)} - \mathbf{x}_t^{(k+1)}). \tag{A.6}$$

It can be verified that the update of (A.3)–(A.6) with $\bar{L}_\rho$ is in fact identical to the update of (10)–(13) with $\hat{L}_{2\rho}$: if we replace $\mathbf{z}_{st}$, $\mathbf{z}_{ts}$, ρ in (8) and (9) using $\mathbf{z}_{st} = (\mathbf{z}'_{st} + \mathbf{z}'_{ts})/2$, $\mathbf{z}_{ts} = (\mathbf{z}'_{st} - \mathbf{z}'_{ts})/2$, and $\rho = \rho'/2$, then we obtain the formulations in (A.1) and (A.2), thus the equivalency between (8)/(9) and (A.1)/(A.2). As a result, their ADMM algorithms (A.3)–(A.6) and (10)–(13) are equivalent as well. Then Lemma 1 is proved by combining the previous analysis with Lemma 3, which shows that (A.3)–(A.6) are equivalent to (17)–(19).

Lemma 3. *The pre-conditioned ADMM procedure for solving* $\min_{\mathbf{x},\mathbf{y}} f(\mathbf{x}) + g(\mathbf{y})$ *subject to* $\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} = \mathbf{c}$

$$\mathbf{x}^{(k+1)} = \underset{\mathbf{x}}{\operatorname{argmin}}\ L(\mathbf{x}, \mathbf{y}^{(k)}, \mathbf{v}^{(k)}) + \frac{\rho}{2}(\mathbf{x} - \mathbf{x}^{(k)})^T\mathbf{C}_1^T\mathbf{C}_1(\mathbf{x} - \mathbf{x}^{(k)}), \tag{A.7}$$
$$\mathbf{y}^{(k+1)} = \underset{\mathbf{y}}{\operatorname{argmin}}\ L(\mathbf{x}^{(k+1)}, \mathbf{y}, \mathbf{v}^{(k)}) + \frac{\rho}{2}(\mathbf{y} - \mathbf{y}^{(k)})^T\mathbf{C}_2^T\mathbf{C}_2(\mathbf{y} - \mathbf{y}^{(k)}), \tag{A.8}$$
$$\mathbf{v}^{(k+1)} = \mathbf{v}^{(k)} + \rho(\mathbf{A}\mathbf{x}^{(k+1)} + \mathbf{B}\mathbf{y}^{(k+1)} - \mathbf{c}), \tag{A.9}$$

where $L(\mathbf{x},\mathbf{y},\mathbf{v}) = f(\mathbf{x}) + g(\mathbf{y}) + \mathbf{v}^T(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} - \mathbf{c}) + \frac{\rho}{2}\|\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} - \mathbf{c}\|^2$*, is equivalent to the standard ADMM procedure applied to the augmented problem*

$$\min_{\mathbf{x},\mathbf{y}} f(\mathbf{x}) + g(\mathbf{y}),\ \ s.t.\ [\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y}, \mathbf{C}_1\mathbf{x}, \mathbf{C}_2\mathbf{y}] = [\mathbf{c}, \mathbf{z}, \mathbf{w}].$$

Proof of Lemma 3. Applying the standard ADMM routine to optimize $(\mathbf{x}, \mathbf{w})$ and $(\mathbf{y}, \mathbf{z})$ alternatively, the update formula for the augmented ADMM is

$$\mathbf{x}^{(k+1)} = \underset{\mathbf{x}}{\operatorname{argmin}}\ L(\mathbf{x}, \mathbf{y}^{(k)}, \mathbf{v}^{(k)}) + \frac{\rho}{2}\|\mathbf{C}_1^{0.5}\mathbf{x} - \mathbf{z}^{(k)}\|^2 + \mathbf{v}_1^{(k)T}(\mathbf{C}_1^{0.5}\mathbf{x} - \mathbf{z}^{(k)}), \tag{A.10}$$
$$\mathbf{w}^{(k+1)} = C_2^{0.5}\mathbf{y}^{(k)} + \frac{1}{\rho}\mathbf{v}_2^{(k)}, \tag{A.11}$$
$$\begin{aligned}\mathbf{y}^{(k+1)} = &\underset{\mathbf{y}}{\operatorname{argmin}}\ L(\mathbf{x}^{(k+1)}, \mathbf{y}, \mathbf{v}^{(k)}) + \\ &\frac{\rho}{2}\|\mathbf{C}_2^{0.5}\mathbf{y} - \mathbf{w}^{(k+1)}\|^2 + \mathbf{v}_2^{(k)T}(\mathbf{C}_2^{0.5}\mathbf{y} - \mathbf{w}^{(k+1)}),\end{aligned} \tag{A.12}$$
$$\mathbf{v}^{(k+1)} = \mathbf{v}^{(k)} + \rho(\mathbf{A}\mathbf{x}^{(k+1)} + \mathbf{B}\mathbf{y}^{(k+1)} - \mathbf{c}), \tag{A.13}$$
$$\mathbf{z}^{(k+1)} = C_1^{0.5}\mathbf{x}^{(k+1)} + \frac{1}{\rho}\mathbf{v}_1^{(k)} \tag{A.14}$$
$$\mathbf{v}_1^{(k+1)} = \mathbf{v}_1^{(k)} + \rho(\mathbf{C}_1^{0.5}\mathbf{x}^{(k+1)} - \mathbf{z}^{(k+1)}) \tag{A.15}$$
$$\mathbf{v}_2^{(k+1)} = \mathbf{v}_2^{(k)} + \rho(\mathbf{C}_2^{0.5}\mathbf{y}^{(k+1)} - \mathbf{w}^{(k+1)}). \tag{A.16}$$

Note that by plugging the definition of $\mathbf{z}^{(k+1)}$ in (A.13) to the definition of $\mathbf{v}^{(k+1)}$ in (A.16), we have $\mathbf{v}_1^{(k+1)} = 0$ for all k. So (A.13) implies that $\mathbf{z}^{(k+1)} = C_1^{0.5}\mathbf{x}^{(k+1)}$ and (A.10) is equivalent to (A.7). Plugging in the definition of $\mathbf{w}^{(k+1)}$ to (A.12), we obtain the equivalence between (A.12) and (A.8). □

A.2. Proof of Lemma 2

Let $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ be the solution. If $\hat{x} = \hat{y}$, then the minimizer is $\hat{x} = \hat{y} = \frac{c_1 a + c_2 b}{c_1 + c_2}$.

If $\hat{x} \neq \hat{y}$, then the minimizer satisfies $2c_1(\hat{x} - a) + \lambda = 0$ and $\hat{x} = \frac{2c_1 a - \lambda}{2c_1}$ and similarly, $\hat{y} = \frac{2c_2 b + \lambda}{2c_2}$. So if $2c_2(2c_1 a - \lambda) > 2c_1(2c_2 b + \lambda)$, i.e., $2c_1 c_2(a - b) > (c_1 + c_2)\lambda$, this is the solution.

If $-2c_1 c_2(a - b) > (c_1 + c_2)\lambda$, then $\hat{x} = \frac{2c_1 a + \lambda}{2c_1}$ and similarly, $\hat{y} = \frac{2c_2 b - \lambda}{2c_2}$.

A.3. Proof of Theorem 1

By calculation, the ADMM algorithm is equivalent to the Douglas–Rachford splitting method applied to

$$\max_{\mathbf{z}} -\mathbf{b}^T\mathbf{z} - f_1^*(-\mathbf{A}_1^T\mathbf{z}) - f_2^*(-\mathbf{A}_2^T\mathbf{z})$$

with two parts given by $f = f_2^*(-\mathbf{A}_2^T\mathbf{z})$ and $g = \mathbf{b}^T\mathbf{z} + f_1^*(-\mathbf{A}_1^T\mathbf{z})$ respectively, and the Douglas–Rachford splitting method is an iterative method that minimizes $f(\mathbf{x}) + g(\mathbf{x})$ with the updating formula

$$\mathbf{x}^{(k+1)} = \frac{1}{2}[(\mathbf{I} - 2\mathrm{prox}_{\rho f})(\mathbf{I} - 2\mathrm{prox}_{\rho g}) + \mathbf{I}](\mathbf{x}^{(k)}).$$

Note that locally around the optimal solution2 we have

$$\mathrm{prox}_{\rho f} = (\mathbf{I} + \rho\partial f)^{-1}, \quad \partial f(\mathbf{x}) = \mathbf{A}_2\mathbf{C}_2^{-1}\mathbf{A}_2^T\mathbf{x} + \mathbf{c}_1,$$
$$\mathrm{prox}_{\rho g} = (\mathbf{I} + \rho\partial g)^{-1}, \quad \partial g(\mathbf{x}) = \mathbf{A}_1\mathbf{C}_1^{-1}\mathbf{A}_1^T\mathbf{x} + \mathbf{c}_2$$

for some constants $\mathbf{c}_1$ and $\mathbf{c}_2$, each iteration of the algorithm is a linear operator in the form of

$$\begin{aligned}&\frac{1}{2}\Big[(\mathbf{I} - 2\mathrm{prox}_{\rho f})(\mathbf{I} - 2\mathrm{prox}_{\rho g}) + \mathbf{I}\Big](\mathbf{x}^{(k+1)} - \mathbf{c}_0)\\ =&\frac{1}{2}\Big[(\mathbf{I} - 2(\mathbf{I} + \rho\mathbf{A}_2\mathbf{C}_2^{-1}\mathbf{A}_2^T)^{-1})(\mathbf{I} - 2(\mathbf{I} + \rho\mathbf{A}_1\mathbf{C}_1^{-1}\mathbf{A}_1^T)^{-1}) + \mathbf{I}\Big](\mathbf{x}^{(k)} - \mathbf{c}_0),\end{aligned}$$

where $\mathbf{c}_0$ is the fixed point of the iterative algorithm that depends on $\mathbf{c}_1$ and $\mathbf{c}_2$. As a result,

$$\mathbf{x}^{(k+1)} - \mathbf{c}_0 = \left(\frac{1}{2}\Big[(\mathbf{I} - 2(\mathbf{I} + \rho\mathbf{A}_2\mathbf{C}_2^{-1}\mathbf{A}_2^T)^{-1})(\mathbf{I} - 2(\mathbf{I} + \rho\mathbf{A}_1\mathbf{C}_1^{-1}\mathbf{A}_1^T)^{-1}) + \mathbf{I}\Big]\right)^k(\mathbf{x}^{(1)} - \mathbf{c}_0),$$

which completes the proof.

Appendix B. Additional experiments

We append more results and discussions for the Chicago Crime experiment in Section 4.2. First of all, we present the fitting results of Algorithm 1 in a geographical map of Chicago in Fig. B.1 for $\lambda \in \{0.013, 0.019, 0.024, 0.041, 0.07\}$.

A popular and common approach for solving classification and regression problems involving graph is the nearest neighbors algorithm (Hastie et al., 2001). Given a set of labeled data $\{(\mathbf{x}_i, y_i)\}$ and an unlabeled data $\mathbf{x}$, it assigns $\mathbf{x}$ to a class based on its nearest neighbors. We apply the idea of k-NN algorithm and adapt it to Chicago Crime example as baseline algorithms. For any two vertices i, j, we define the distance $d(i, j)$ as the length of the shortest path from i to j. The first method, *NN-average (average over nearest neighbors)*, is to assign the value of the vertex i is taking the averaged response values of its neighbors $\hat{x}_i = \frac{1}{|\mathcal{N}_K(i)|}\sum_{j\in\mathcal{N}_K(i)} x_j$ where $\mathcal{N}_K(i) := \{j \mid d(i, j) \leq K\}$. The second method, *NN-max (maximum vote over nearest neighbors)*, treat it as a classification problem by first assigning each vertex i to one of the M classes by $L_i = \lceil M x_i / \max\{x_1, \ldots, x_n\}\rceil$, where M is total number of labels, and then predict $\hat{x}_i$ based on the majority vote of its neighbors in $\mathcal{N}_K(i)$, i.e., $\hat{x}_i = \frac{\max\{x_1,\ldots,x_n\}}{M} \times \arg\max_v \sum_{j\in\mathcal{N}_K(x_i)} \mathbf{1}_{\{v=L_j\}}$. The smoothed burglaries occurring rates generated by NN-average and NN-max are reported in Figs. B.2 and B.3 respectively, where the parameters are set by $K \leq 4$ and $M = 10$. As predicted in Arnold and Tibshirani (2016), the graph fused lasso-based algorithms have the advantage over NN-average and NN-max in the sense of local adaptivity: By only tuning the parameter λ, the size of a cluster given the measurements of a graph is automatically determined by our algorithm, while NN-average and NN-max tend to create roughly equal sized clusters.

References

Arnold, T.B., Tibshirani, R.J., 2016. Efficient implementations of the generalized Lasso dual path algorithm. J. Comput. Graph. Statist. 25 (1), 1–27. http://dx.doi.org/10.1080/10618600.2015.1008638, arXiv:https://doi.org/10.1080/10618600.2015.1008638.

Barbero, À., Sra, S., 2018. Modular proximal optimization for multidimensional total-variation regularization. J. Mach. Learn. Res. 19 (1), 2232–2313.

Batson, J., Spielman, D.A., Srivastava, N., Teng, S.-H., 2013. Spectral sparsification of graphs: Theory and algorithms. Commun. ACM 56 (8), 87–94. http://dx.doi.org/10.1145/2492007.2492029.

Benning, M., Knoll, F., Schönlieb, C.-B., Valkonen, T., 2016. Preconditioned ADMM with nonlinear operator constraint. In: Bociu, L., Désidéri, J.-A., Habbal, A. (Eds.), System Modeling and Optimization. Springer International Publishing, Cham, pp. 117–126.

Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J., 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. Found. Trends Mach. Learn. 3 (1), 1–122. http://dx.doi.org/10.1561/2200000016.

Chambolle, A., Darbon, J., 2009. On total variation minimization and surface evolution using parametric maximum flows. Int. J. Comput. Vis. 84 (3), 288.

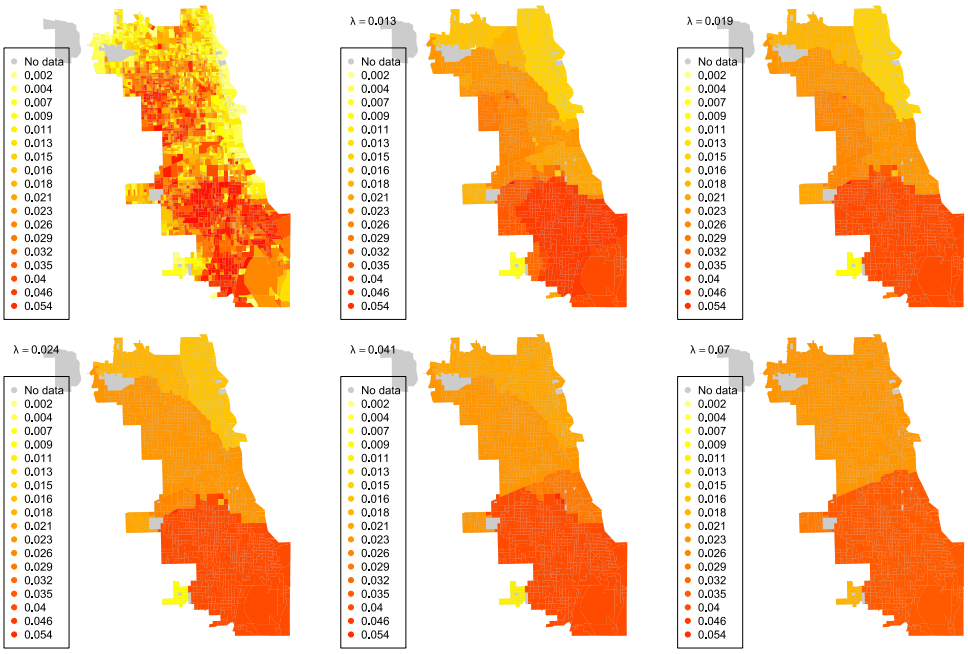

Fig. B.1. Sub-figure 1 (top left) shows observed proportions of reported burglaries per household between 2005–2009 in Chicago, IL. Sub-figure 2–6 show solutions of our algorithm, corresponding to $\lambda \in \{0.013, 0.019, 0.024, 0.041, 0.07\}$ respectively, along the fused lasso path that was fit to the observed proportions of burglaries.

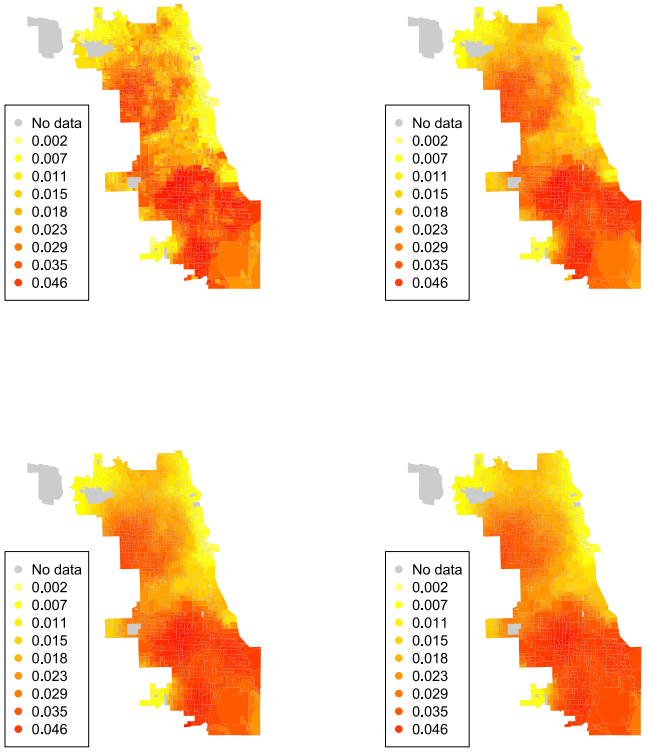

Fig. B.2. The smoothed of burglaries occurring rates by NN-average. The maximum distance of neighbors K are set as 1 to 4 for the subplots.

Chen, X., Lin, Q., Kim, S., Carbonell, J.G., Xing, E.P., 2012. Smoothing proximal gradient method for general structured sparse regression. Ann. Appl. Stat. 6 (2), 719–752. http://dx.doi.org/10.1214/11-AOAS514.

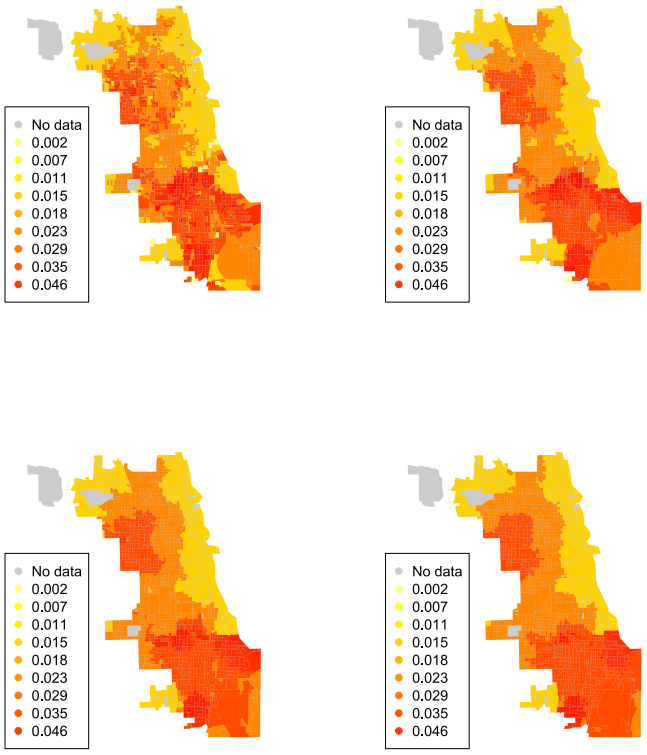

Fig. B.3. The smoothed of burglaries occurring rates by NN-max. The maximum distance of neighbors K are set as 1 to 4 for the subplots. The number of the total class is set as $M = 10$.

Chopra, A., Lian, H., 2010. Total variation, adaptive total variation and nonconvex smoothly clipped absolute deviation penalty for denoising blocky images. Pattern Recognit. 43 (8), 2609–2619. http://dx.doi.org/10.1016/j.patcog.2010.03.022, URL https://www.sciencedirect.com/science/article/pii/S0031320310001421.

Condat, L., 2013. A direct algorithm for 1-D total variation denoising. IEEE Signal Process. Lett. 20 (11), 1054–1057. http://dx.doi.org/10.1109/LSP.2013.2278339.

Cui, L., Bai, L., Wang, Y., Yu, P.S., Hancock, E.R., 2021. Fused Lasso for feature selection using structural information. Pattern Recognit. 119, 108058. http://dx.doi.org/10.1016/j.patcog.2021.108058, URL https://www.sciencedirect.com/science/article/pii/S0031320321002454.

Davies, P.L., Kovac, A., 2001. Local extremes, runs, strings and multiresolution. Ann. Statist. 29 (1), 1–65. http://dx.doi.org/10.1214/aos/996986501.

Deng, W., Yin, W., 2016. On the global and linear convergence of the generalized alternating direction method of multipliers. J. Sci. Comput. 66 (3), 889–916. http://dx.doi.org/10.1007/s10915-015-0048-x.

Eckstein, J., Bertsekas, D.P., 1992. On the Douglas–Rachford splitting method and the proximal point algorithm for maximal monotone operators. Math. Program. 55 (1), 293–318. http://dx.doi.org/10.1007/BF01581204.

França, G., Bento, J., 2017. How is distributed ADMM affected by network topology. Stat 1050, 2.

Friedman, J., Hastie, T., Höfling, H., Tibshirani, R., et al., 2007. Pathwise coordinate optimization. Ann. Appl. Stat. 1 (2), 302–332.

Giselsson, P., 2017. Tight global linear convergence rate bounds for Douglas–Rachford splitting. J. Fixed Point Theory Appl. 19 (4), 2241–2270.

Goldstein, T., O'Donoghue, B., Setzer, S., Baraniuk, R., 2014. Fast alternating direction optimization methods. SIAM J. Imaging Sci. 7 (3), 1588–1623. http://dx.doi.org/10.1137/120896219, arXiv:https://doi.org/10.1137/120896219.

Guo, K., Han, D., Yuan, X., 2017. Convergence analysis of Douglas–Rachford splitting method for "strongly+ weakly" convex programming. SIAM J. Numer. Anal. 55 (4), 1549–1577.

Hallac, D., Leskovec, J., Boyd, S., 2015. Network Lasso: Clustering and optimization in large graphs. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 387–396.

Hastie, T., Tibshirani, R., Friedman, J., Hastie, T., Friedman, J., Tibshirani, R., 2001. The Elements of Statistical Learning. Springer New York.

Johnson, N.A., 2013. A dynamic programming algorithm for the fused Lasso and l 0-segmentation. J. Comput. Graph. Statist. 22 (2), 246–260. http://dx.doi.org/10.1080/10618600.2012.681238, arXiv:http://dx.doi.org/10.1080/10618600.2012.681238.

Kolmogorov, V., Pock, T., Rolinek, M., 2016. Total variation on a tree. SIAM J. Imaging Sci. 9 (2), 605–636.

Lin, X., Pham, M., Ruszczyundefinedski, A., 2014. Alternating linearization for structured regularization problems. J. Mach. Learn. Res. 15 (1), 3447–3481.

Liu, J., Yuan, L., Ye, J., 2010. An efficient algorithm for a class of fused Lasso problems. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '10, ACM, New York, NY, USA, pp. 323–332. http://dx.doi.org/10.1145/1835804.1835847, URL http://doi.acm.org/10.1145/1835804.1835847.

Mu, L., Liu, H., 2020. Noninvasive electrocardiographic imaging with low-rank and non-local total variation regularization. Pattern Recognit. Lett. 138, 106–114. http://dx.doi.org/10.1016/j.patrec.2020.07.007, URL https://www.sciencedirect.com/science/article/pii/S0167865520302531.

Nelson, J., 2013. Fused Lasso and rotation invariant autoregressive models for texture classification. Pattern Recognit. Lett. 34 (16), 2166–2172. http://dx.doi.org/10.1016/j.patrec.2013.08.003, URL https://www.sciencedirect.com/science/article/pii/S0167865513002985.

Nishihara, R., Lessard, L., Recht, B., Packard, A., Jordan, M., 2015. A general analysis of the convergence of ADMM. In: International Conference on Machine Learning. pp. 343–352.

Ramdas, A., Tibshirani, R.J., 2015. Fast and flexible ADMM algorithms for trend filtering. J. Comput. Graph. Statist. 25 (3), 839–858. http://dx.doi.org/10.1080/10618600.2015.1054033, arXiv:1406.2082.

Rudin, L.I., Osher, S., Fatemi, E., 1992. Nonlinear total variation based noise removal algorithms. Physica D 60 (1), 259–268. http://dx.doi.org/10.1016/0167-2789(92)90242-F, URL http://www.sciencedirect.com/science/article/pii/016727899290242F.

Tansey, W., Scott, J.G., 2015. A fast and flexible algorithm for the graph-fused Lasso. arXiv preprint arXiv:1505.06475.

Tibshirani, R., Saunders, M., Rosset, S., Zhu, J., Knight, K., 2005. Sparsity and smoothness via the fused Lasso. J. R. Stat. Soc. Ser. B Stat. Methodol. 67 (1), 91–108.

Tibshirani, R.J., Taylor, J., et al., 2011. The solution path of the generalized Lasso. Ann. Statist. 39 (3), 1335–1371.

Vert, J.-P., Bleakley, K., 2010. Fast detection of multiple change-points shared by many signals using group LARS. Adv. Neural Inf. Process. Syst. 23, 2343–2351.

Wahlberg, B., Boyd, S., Annergren, M., Wang, Y., 2012. An ADMM algorithm for a class of total variation regularized estimation problems. IFAC Proc. Vol. 45 (16), 83–88.

Xu, Z., Figueiredo, M., Goldstein, T., 2017. Adaptive ADMM with spectral penalty parameter selection. In: Singh, A., Zhu, J. (Eds.), Proceedings of the 20th International Conference on Artificial Intelligence and Statistics. In: Proceedings of Machine Learning Research, vol. 54, PMLR, pp. 718–727, URL https://proceedings.mlr.press/v54/xu17a.html.

Yang, L., Ding, S., Zhou, F., Yang, X., Xiao, Y., 2021. Robust EEG feature learning model based on an adaptive weight and pairwise-fused Lasso. Biomed. Signal Process. Control 68, 102728. http://dx.doi.org/10.1016/j.bspc.2021.102728, URL https://www.sciencedirect.com/science/article/pii/S1746809421003256.

Yang, W.H., Han, D., 2016. Linear convergence of the alternating direction method of multipliers for a class of convex optimization problems. SIAM J. Numer. Anal. 54 (2), 625–640. http://dx.doi.org/10.1137/140974237, arXiv:https://doi.org/10.1137/140974237.

Ye, G.-B., Xie, X., 2011. Split bregman method for large scale fused Lasso. Comput. Statist. Data Anal. 55 (4), 1552–1569. http://dx.doi.org/10.1016/j.csda.2010.10.021, URL http://www.sciencedirect.com/science/article/pii/S0167947310004093.

Yu, D., Won, J.-H., Lee, T., Lim, J., Yoon, S., 2015. High-dimensional fused Lasso regression using majorization-minimization and parallel processing. J. Comput. Graph. Statist. 24 (1), 121–153.

Zhang, Z., Tian, Y., Bai, L., Xiahou, J., Hancock, E., 2017. High-order covariate interacted Lasso for feature selection. Pattern Recognit. Lett. 87, 139–146. http://dx.doi.org/10.1016/j.patrec.2016.08.005, URL https://www.sciencedirect.com/science/article/pii/S0167865516302045, Advances in Graph-based Pattern Recognition.

Zhu, Y., 2017. An augmented ADMM algorithm with application to the generalized lasso problem. J. Comput. Graph. Statist. 26 (1), 195–204. http://dx.doi.org/10.1080/10618600.2015.1114491, arXiv:https://doi.org/10.1080/10618600.2015.1114491.

Appendix A. Appendix: Technical Proofs

Appendix A.1. Proof of Lemma 1

Let us consider the problem

$$\underset{\substack{\{\mathbf{x}_i\}_{i\in\mathcal{E}},\\ \{\mathbf{z}_{st}\}_{(s,t)\in\mathcal{E}_1}}}{\operatorname{argmin}} \Big(\sum_{i\in\mathcal{E}} f_i(\mathbf{x}_i) + \lambda \sum_{(s,t)\in\mathcal{E}_0} \|\mathbf{x}_s - \mathbf{x}_t\|\Big) + \lambda \sum_{(s,t)\in\mathcal{E}_1} \|\mathbf{z}_{st}\| \tag{A.1}$$
$$\text{s.t. } \mathbf{z}_{st} = \mathbf{x}_s - \mathbf{x}_t,\, \mathbf{z}_{ts} = \mathbf{x}_s + \mathbf{x}_t.$$

and its associated Lagrangian

$$\begin{aligned}\bar{L}_\rho(x,z,u) = \sum_{i\in\mathcal{V}} f_i(\mathbf{x}_i) + \lambda \sum_{(s,t)\in\mathcal{E}_0} \|\mathbf{x}_s - \mathbf{x}_t\| + \\ \sum_{(s,t)\in\mathcal{E}_1} \Big(\lambda\|\mathbf{z}_{st}\| + \mathbf{u}_{st}^T(\mathbf{z}_{st} - \mathbf{x}_s + \mathbf{x}_t) + \mathbf{u}_{ts}^T(\mathbf{z}_{ts} - \mathbf{x}_s - \mathbf{x}_t) \\ + \frac{\rho}{2}\|\mathbf{z}_{st} - \mathbf{x}_s + \mathbf{x}_t\|^2 + \frac{\rho}{2}\|\mathbf{z}_{ts} - \mathbf{x}_s - \mathbf{x}_t\|^2\Big),\end{aligned} \tag{A.2}$$

as well as the ADMM algorithm of

$$x^{(k+1)} = \underset{x}{\operatorname{argmin}}\, \bar{L}_\rho(x, z^{(k)}, u^{(k)}) \tag{A.3}$$
$$z^{(k+1)} = \underset{z}{\operatorname{argmin}}\, \bar{L}_\rho(x^{(k+1)}, z, u^{(k)}) \tag{A.4}$$
$$\mathbf{u}_{st}^{(k+1)} = \mathbf{u}_{st}^{(k)} + \rho(\mathbf{z}_{st}^{(k+1)} - \mathbf{x}_s^{(k+1)} + \mathbf{x}_t^{(k+1)}) \tag{A.5}$$
$$\mathbf{u}_{ts}^{(k+1)} = \mathbf{u}_{ts}^{(k)} + \rho(\mathbf{z}_{ts}^{(k+1)} - \mathbf{x}_s^{(k+1)} - \mathbf{x}_t^{(k+1)}). \tag{A.6}$$

It can be verified that the update of (A.3)–(A.6) with $\bar{L}_\rho$ is in fact identical to the update of (10)–(13) with $\hat{L}_{2\rho}$: if we replace $\mathbf{z}_{st}$, $\mathbf{z}_{ts}$, ρ in (8) and (9) using $\mathbf{z}_{st} = (\mathbf{z}'_{st} + \mathbf{z}'_{ts})/2$, $\mathbf{z}_{ts} = (\mathbf{z}'_{st} - \mathbf{z}'_{ts})/2$, and $\rho = \rho'/2$, then we obtain the formulations in (A.1) and (A.2), thus the equivalency between (8)/(9) and (A.1)/(A.2). As a result, their ADMM algorithms (A.3)–(A.6) and (10)–(13) are equivalent as well. Then Lemma 1 is proved by combining the previous analysis with Lemma 3, which shows that (A.3)–(A.6) are equivalent to (17)–(19).

Lemma 3. *The pre-conditioned ADMM procedure for solving* $\min_{\mathbf{x},\mathbf{y}} f(\mathbf{x}) + g(\mathbf{y})$ *subject to* $\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} = \mathbf{c}$

$$\mathbf{x}^{(k+1)} = \underset{\mathbf{x}}{\operatorname{argmin}}\, L(\mathbf{x}, \mathbf{y}^{(k)}, \mathbf{v}^{(k)}) + \frac{\rho}{2}(\mathbf{x} - \mathbf{x}^{(k)})^T \mathbf{C}_1^T \mathbf{C}_1 (\mathbf{x} - \mathbf{x}^{(k)}), \tag{A.7}$$

$$\mathbf{y}^{(k+1)} = \underset{\mathbf{y}}{\operatorname{argmin}}\, L(\mathbf{x}^{(k+1)}, \mathbf{y}, \mathbf{v}^{(k)}) + \frac{\rho}{2}(\mathbf{y} - \mathbf{y}^{(k)})^T \mathbf{C}_2^T \mathbf{C}_2 (\mathbf{y} - \mathbf{y}^{(k)}), \tag{A.8}$$

$$\mathbf{v}^{(k+1)} = \mathbf{v}^{(k)} + \rho(\mathbf{A}\mathbf{x}^{(k+1)} + \mathbf{B}\mathbf{y}^{(k+1)} - \mathbf{c}), \tag{A.9}$$

where $L(\mathbf{x}, \mathbf{y}, \mathbf{v}) = f(\mathbf{x}) + g(\mathbf{y}) + \mathbf{v}^T(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} - \mathbf{c}) + \frac{\rho}{2}\|\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} - \mathbf{c}\|^2$, *is equivalent to the standard ADMM procedure applied to the augmented problem*

$$\min_{\mathbf{x},\mathbf{y}} f(\mathbf{x}) + g(\mathbf{y}),\ \ s.t.\ [\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y}, \mathbf{C}_1\mathbf{x}, \mathbf{C}_2\mathbf{y}] = [\mathbf{c}, \mathbf{z}, \mathbf{w}].$$

Proof of Lemma 3. Applying the standard ADMM routine to optimize $(\mathbf{x}, \mathbf{w})$ and $(\mathbf{y}, \mathbf{z})$ alternatively, the update formula for the augmented ADMM is

$$\mathbf{x}^{(k+1)} = \underset{\mathbf{x}}{\operatorname{argmin}}\, L(\mathbf{x}, \mathbf{y}^{(k)}, \mathbf{v}^{(k)}) + \frac{\rho}{2}\|\mathbf{C}_1^{0.5}\mathbf{x} - \mathbf{z}^{(k)}\|^2 + \mathbf{v}_1^{(k)\,T}(\mathbf{C}_1^{0.5}\mathbf{x} - \mathbf{z}^{(k)}), \tag{A.10}$$

$$\mathbf{w}^{(k+1)} = C_2^{0.5}\mathbf{y}^{(k)} + \frac{1}{\rho}\mathbf{v}_2^{(k)}, \tag{A.11}$$

$$\mathbf{y}^{(k+1)} = \underset{\mathbf{y}}{\operatorname{argmin}}\, L(\mathbf{x}^{(k+1)}, \mathbf{y}, \mathbf{v}^{(k)}) + \frac{\rho}{2}\|\mathbf{C}_2^{0.5}\mathbf{y} - \mathbf{w}^{(k+1)}\|^2 + \mathbf{v}_2^{(k)\,T}(\mathbf{C}_2^{0.5}\mathbf{y} - \mathbf{w}^{(k+1)}), \tag{A.12}$$

$$\mathbf{v}^{(k+1)} = \mathbf{v}^{(k)} + \rho(\mathbf{A}\mathbf{x}^{(k+1)} + \mathbf{B}\mathbf{y}^{(k+1)} - \mathbf{c}), \tag{A.13}$$

$$\mathbf{z}^{(k+1)} = C_1^{0.5}\mathbf{x}^{(k+1)} + \frac{1}{\rho}\mathbf{v}_1^{(k)} \tag{A.14}$$

$$\mathbf{v}_1^{(k+1)} = \mathbf{v}_1^{(k)} + \rho(\mathbf{C}_1^{0.5}\mathbf{x}^{(k+1)} - \mathbf{z}^{(k+1)}) \tag{A.15}$$

$$\mathbf{v}_2^{(k+1)} = \mathbf{v}_2^{(k)} + \rho(\mathbf{C}_2^{0.5}\mathbf{y}^{(k+1)} - \mathbf{w}^{(k+1)}). \tag{A.16}$$

Note that by plugging the definition of $\mathbf{z}^{(k+1)}$ in (A.13) to the definition of $\mathbf{v}^{(k+1)}$ in (A.16), we have $\mathbf{v}_1^{(k+1)} = 0$ for all k. So (A.13) implies that $\mathbf{z}^{(k+1)} = C_1^{0.5}\mathbf{x}^{(k+1)}$ and (A.10) is equivalent to (A.7). Plugging in the definition of $\mathbf{w}^{(k+1)}$ to (A.12), we obtain the equivalence between (A.12) and (A.8). □

Appendix A.2. Proof of Lemma 2

Let $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ be the solution. If $\hat{x} = \hat{y}$, then the minimizer is $\hat{x} = \hat{y} = \frac{c_1 a + c_2 b}{c_1 + c_2}$.

If $\hat{x} \neq \hat{y}$, then the minimizer satisfies $2c_1(\hat{x} - a) + \lambda = 0$ and $\hat{x} = \frac{2c_1 a - \lambda}{2c_1}$ and similarly, $\hat{y} = \frac{2c_2 b + \lambda}{2c_2}$. So if $2c_2(2c_1 a - \lambda) > 2c_1(2c_2 b + \lambda)$, i.e., $2c_1 c_2(a - b) > (c_1 + c_2)\lambda$, this is the solution.

If $-2c_1 c_2(a - b) > (c_1 + c_2)\lambda$, then $\hat{x} = \frac{2c_1 a + \lambda}{2c_1}$ and similarly, $\hat{y} = \frac{2c_2 b - \lambda}{2c_2}$.

Appendix A.3. Proof of Theorem 1

By calculation, the ADMM algorithm is equivalent to the Douglas-Rachford splitting method applied to

$$\max_{\mathbf{z}} -\mathbf{b}^T\mathbf{z} - f_1^*(-\mathbf{A}_1^T\mathbf{z}) - f_2^*(-\mathbf{A}_2^T\mathbf{z})$$

with two parts given by $f = f_2^*(-\mathbf{A}_2^T\mathbf{z})$ and $g = \mathbf{b}^T\mathbf{z} + f_1^*(-\mathbf{A}_1^T\mathbf{z})$ respectively, and the Douglas-Rachford splitting method is an iterative method that minimizes $f(\mathbf{x}) + g(\mathbf{x})$ with the updating formula

$$\mathbf{x}^{(k+1)} = \frac{1}{2}[(\mathbf{I} - 2\mathrm{prox}_{\rho f})(\mathbf{I} - 2\mathrm{prox}_{\rho g}) + \mathbf{I}](\mathbf{x}^{(k)}).$$

Note that locally around the optimal solution2 we have

$$\begin{aligned}
&\mathrm{prox}_{\rho f} = (\mathbf{I} + \rho\partial f)^{-1}, \;\; \partial f(\mathbf{x}) = \mathbf{A}_2\mathbf{C}_2^{-1}\mathbf{A}_2^T\mathbf{x} + \mathbf{c}_1, \\
&\mathrm{prox}_{\rho g} = (\mathbf{I} + \rho\partial g)^{-1}, \;\; \partial g(\mathbf{x}) = \mathbf{A}_1\mathbf{C}_1^{-1}\mathbf{A}_1^T\mathbf{x} + \mathbf{c}_2
\end{aligned}$$

for some constants $\mathbf{c}_1$ and $\mathbf{c}_2$, each iteration of the algorithm is a linear operator in the form of

$$\begin{aligned}
&\frac{1}{2}\Big[(\mathbf{I} - 2\mathrm{prox}_{\rho f})(\mathbf{I} - 2\mathrm{prox}_{\rho g}) + \mathbf{I}\Big](\mathbf{x}^{(k+1)} - \mathbf{c}_0) \\
=&\frac{1}{2}\Big[(\mathbf{I} - 2(\mathbf{I} + \rho\mathbf{A}_2\mathbf{C}_2^{-1}\mathbf{A}_2^T)^{-1})(\mathbf{I} - 2(\mathbf{I} + \rho\mathbf{A}_1\mathbf{C}_1^{-1}\mathbf{A}_1^T)^{-1}) + \mathbf{I}\Big](\mathbf{x}^{(k)} - \mathbf{c}_0),
\end{aligned}$$

where $\mathbf{c}_0$ is the fixed point of the iterative algorithm that depends on $\mathbf{c}_1$ and $\mathbf{c}_2$. As a result,

$$\mathbf{x}^{(k+1)} - \mathbf{c}_0 = \Big(\frac{1}{2}\Big[(\mathbf{I} - 2(\mathbf{I} + \rho\mathbf{A}_2\mathbf{C}_2^{-1}\mathbf{A}_2^T)^{-1})(\mathbf{I} - 2(\mathbf{I} + \rho\mathbf{A}_1\mathbf{C}_1^{-1}\mathbf{A}_1^T)^{-1}) + \mathbf{I}\Big]\Big)^k(\mathbf{x}^{(1)} - \mathbf{c}_0),$$

which completes the proof.

Appendix B. Appendix: Additional Experiments

We append more results and discussions for the Chicago Crime experiment in Section 4.2. First of all, we present the fitting results of Algorithm 1 in a geographical map of Chicago in Figure Appendix B.1 for $\lambda \in \{0.013, 0.019, 0.024, 0.041, 0.07\}$.

A popular and common approach for solving classification and regression problems involving graph is the nearest neighbors algorithm [20]. Given a set of labeled data $\{(\mathbf{x}_i, y_i)\}$ and an unlabeled data $\mathbf{x}$, it assigns $\mathbf{x}$ to a class based on its nearest neighbors. We apply the idea of k-NN algorithm and adapt it to Chicago Crime example as baseline algorithms. For any two vertices i, j, we define the distance $d(i, j)$ as the length of the shortest path from i to j. The first method, *NN-average (average over nearest neighbors)*, is to assign the value of the vertex i is taking the averaged response values of its neighbors $\hat{x}_i = \frac{1}{|\mathcal{N}_K(i)|} \sum_{j \in \mathcal{N}_K(i)} x_j$ where $\mathcal{N}_K(i) := \{j \mid d(i, j) \leq K\}$. The second method, *NN-max (maximum vote over nearest neighbors)*, treat it as a classification problem by first assigning each vertex i to one of the M classes by $L_i = \lceil M x_i / \max\{x_1, \cdots, x_n\} \rceil$, where M is total number of labels, and then predict $\hat{x}_i$ based on the majority vote of its neighbors in $\mathcal{N}_K(i)$, i.e., $\hat{x}_i = \frac{\max\{x_1, \cdots, x_n\}}{M} \times \arg\max_v \sum_{j \in \mathcal{N}_K(x_i)} \mathbf{1}_{\{v = L_j\}}$. The smoothed burglaries occurring rates generated by NN-average and NN-max are reported in Figure Appendix B.2 and Appendix B.3 respectively, where the parameters are set by $K \leq 4$ and $M = 10$. As predicted in [1], the graph fused lasso-based algorithms have the advantage over NN-average and NN-max in the sense of local adaptivity: By only tuning the parameter λ, the size of a cluster given the measurements of a graph is automatically determined by our algorithm, while NN-average and NN-max tend to create roughly equal sized clusters.

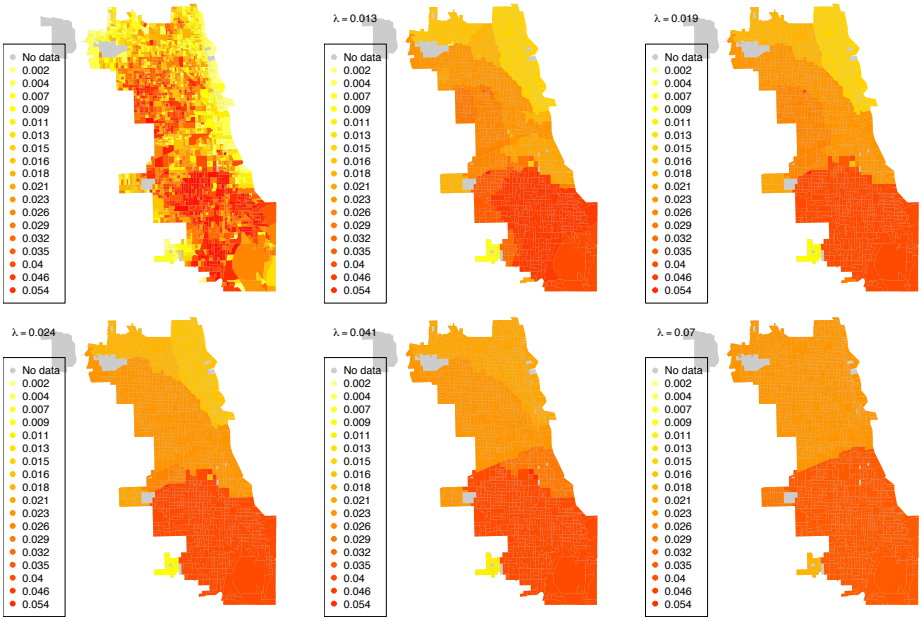

Figure Appendix B.1: Sub-figure 1 (top left) shows observed proportions of reported burglaries per household between 2005–2009 in Chicago, IL. Sub-figure 2-6 show solutions of our algorithm, corresponding to $\lambda \in \{0.013, 0.019, 0.024, 0.041, 0.07\}$ respectively, along the fused lasso path that was fit to the observed proportions of burglaries.

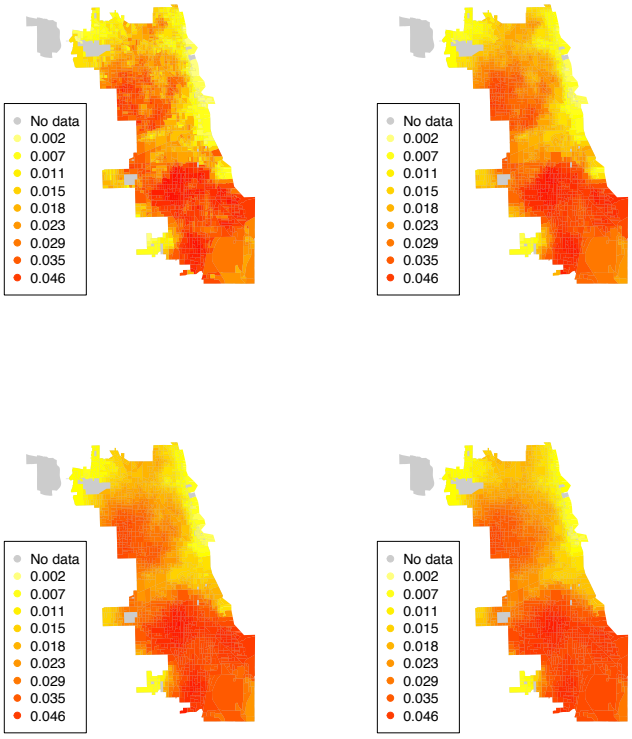

Figure Appendix B.2: The smoothed of burglaries occurring rates by NN-average. The maximum distance of neighbors K are set as 1 to 4 for the subplots.

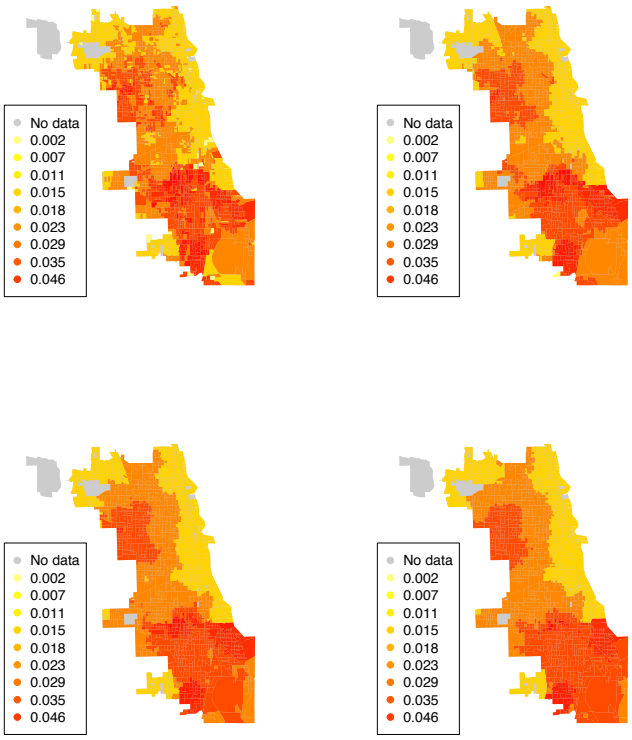

Figure Appendix B.3: The smoothed of burglaries occurring rates by NN-max. The maximum distance of neighbors K are set as 1 to 4 for the subplots. The number of the total class is set as $M = 10$.