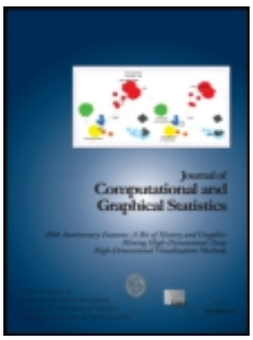

Journal of Computational and Graphical Statistics

ISSN: 1061-8600 (Print) 1537-2715 (Online) Journal homepage: http://www.tandfonline.com/loi/ucgs20

A Path Algorithm for the Fused Lasso Signal Approximator

Holger Hoefling

Supplementary materials for this article are available online.
Please click the JCGS link at http://pubs.amstat.org.

A Path Algorithm for the Fused Lasso Signal Approximator

Holger HOEFLING

The Lasso is a very well-known penalized regression model, which adds an L_1 penalty with parameter λ_1 on the coefficients to the squared error loss function. The Fused Lasso extends this model by also putting an L_1 penalty with parameter λ_2 on the difference of neighboring coefficients, assuming there is a natural ordering. In this article, we develop a path algorithm for solving the Fused Lasso Signal Approximator that computes the solutions for all values of λ_1 and λ_2. We also present an approximate algorithm that has considerable speed advantages for a moderate trade-off in accuracy. In the Online Supplement for this article, we provide proofs and further details for the methods developed in the article.

Key Words: Convex optimization; Lasso; Penalized regression.

1. INTRODUCTION

In recent years, many regression procedures have been proposed that use penalties on the regression coefficients in order to achieve sparseness or shrink them toward zero. One of the most widely known procedures of this type is the Lasso (see Tibshirani 1996), which minimizes the loss function

$$\frac{1}{2}(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})^T(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})+\lambda_1\sum_{i=1}^{p}|\beta_i|.$$

Here, $\mathbf{y}\in\mathbb{R}^n$ is the response vector, $\mathbf{X}\in\mathbb{R}^{n\times p}$ is the matrix of predictors, and $\boldsymbol{\beta}\in\mathbb{R}^p$ is the coefficient vector. Several years after the original Lasso article was published, an algorithm was developed which gives the whole solution path of the Lasso for the penalty parameter λ_1 with the computational complexity of an ordinary least squares problem (see Osborne, Presnell, and Turlach 2000; Efron et al. 2004). Subsequently, path algorithms for several other regression methods were developed as well, for example for generalized

Holger Hoefling is Scientific Assistant, Institute of Medical Biometry and Medical Informatics, Department of Medical Biometry and Statistics, University Medical Center Freiburg, Freiburg 70194, Germany (E-mail: *hhoeflin@gmail.com*).

Journal of Computational and Graphical Statistics, Volume 19, Number 4, Pages 984–1006
DOI: 10.1198/jcgs.2010.09208

linear models (see Park and Hastie 2007) or the SVM (see Hastie et al. 2004) among others. A more general treatment of conditions under which the solution paths are piecewise linear can be found in the article by Rosset and Zhu (2007).

An example of an extension of the Lasso is the Fused Lasso introduced by Tibshirani et al. (2005). For the Fused Lasso, it is assumed that there is some natural ordering of the coefficients (e.g., each coefficient corresponds to a position on a straight line). If coefficients in the true model are closely related to their neighbors, we can exploit this by placing an additional penalty on the differences of neighboring coefficients. Several different choices for these penalties are possible and in the case of the Fused Lasso, an L_1 penalty is being used. The resulting loss function is then

$$\frac{1}{2}(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})^T(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})+\lambda_1\sum_{i=1}^{n}|\beta_i|+\lambda_2\sum_{i=1}^{n-1}|\beta_i-\beta_{i+1}|,$$

where the second penalty with parameter λ_2 shrinks neighboring coefficients toward each other. Just as the L_1 penalty on the absolute values $|\beta_i|$ for the Lasso encourages sparseness, the penalty on $|\beta_i - \beta_{i+1}|$ tends to set neighboring penalties exactly equal to each other. As such, the method is especially suitable for coefficients that are constant for an interval and change in jumps.

In this article, we want to concentrate on the most widely used case for this method, the Fused Lasso Signal Approximator (FLSA). In the FLSA, we assume that we have $\mathbf{X} = \mathbf{I}$ as the predictor matrix. One example for this would be comparative genomic hybridization (CGH) or chromosomal microarray analysis (CMA) data. CGH is a method that identifies DNA copy number gains and losses on chromosomes by making two color fluorescence in situ hybridization at various points of the chromosomes. In this technique, normal and tumor DNA are labeled with fluorescent dyes (e.g., red and green) and using a microarray analysis, regions of increased or decreased fluorescence of one color compared to the other can be identified, indicating gains or losses of DNA at this place of the chromosome. As usual with this type of data, it is very noisy. Therefore, we seek to exploit that gains or losses typically appear for whole regions in the genome and that these changes usually occur in jumps. We can do this by penalizing differences of neighboring coefficients and therefore decrease the noise in the data and improve estimation. In this case, we use the one-dimensional Fused Lasso Signal Approximator (FLSA), for which the loss function is

$$L(\mathbf{y},\boldsymbol{\beta})=\frac{1}{2}\sum_{i=1}^{n}(y_i-\beta_i)^2+\lambda_1\sum_{i=1}^{n}|\beta_i|+\lambda_2\sum_{i=1}^{n-1}|\beta_i-\beta_{i+1}|.$$

Every coefficient β_i is an estimate of the measurement y_i taken at position i (which we assume to be ordered along the chromosome). Apart from the Lasso penalty $\lambda_1 \sum_{i=1}^{n} |\beta_i|$, the additional penalty placed on the difference between neighboring coefficients is $\lambda_2 \sum_{i=1}^{n-1} |\beta_i - \beta_{i+1}|$. An example of CGH measurements in lung cancer can be seen in Figure 1. The solid line is the estimates for penalty parameters $\lambda_1 = 0$ and $\lambda_2 = 2$. We can see that starting around measurement 150, the CGH results are on average below 0, indicating a loss of DNA in this region.

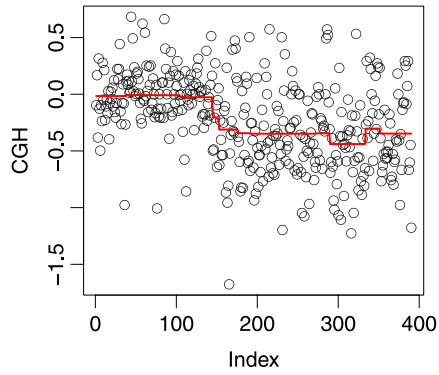

Figure 1. Example using the one-dimensional Fused Lasso Signal Approximator on lung cancer CGH data. The online version of this figure is in color.

Another example where the Fused Lasso model can be used is in image reconstruction. As a toy example, look at Figure 2. On the left side, we can see the true image and a noisy version in the middle. On the right side is the de-noised version using the Fused Lasso. As the coefficients are not located on a straight line but instead on a two-dimensional (2-D) grid, we have to use a different version of the penalty that penalizes all differences of neighboring coefficients in two dimensions.

In its more general form, we assume that each coefficient corresponds to a node in a graph $\mathcal{G} = (V, E)$. Then we penalize every difference of coefficients if the corresponding nodes have an edge between them. Specifically, the loss function becomes in this case

$$\frac{1}{2}\sum_{i=1}^{n}(y_i - \beta_i)^2 + \lambda_1 \sum_{s=i}^{n} |\beta_i| + \lambda_2 \sum_{(i,j)\in E; i<j} |\beta_i - \beta_j|,$$

which we will refer to as the Fused Lasso Signal Approximator (FLSA) with general graphs. In the example above, the graph is a 2-D grid. Previous work published by Pollak et al. (2005) provides an exact algorithm for the one-dimensional case and an approximate version for the two-dimensional image reconstruction. In our work, we solve a more general problem exactly and get the exact one-dimensional algorithm as a special case.

In the following sections, we will present path algorithms for the Fused Lasso Signal Approximator in its special one-dimensional and its general form. In Section 2 we will

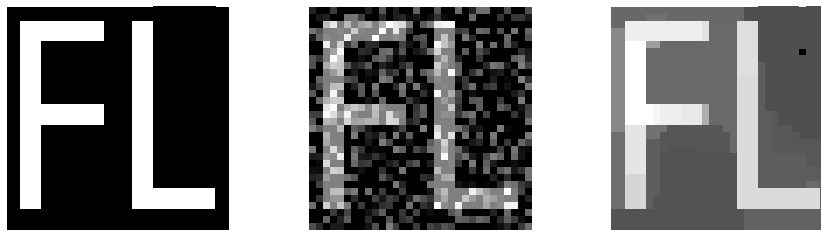

Figure 2. Example of image reconstruction using the Fused Lasso. On the left side is the true image. The noisy version is in the middle. The reconstructed version using the Fused Lasso is on the right.

first treat the general FLSA case for an arbitrary penalty graph $\mathcal{G}$ and will then specialize it in Section 3 to the one-dimensional case. We prove that our algorithm yields the exact solution and present simulation studies that compare the new algorithms to existing methods. Finally, in Section 4 we will discuss the results and give possible extensions of these algorithms.

2. FUSED LASSO SIGNAL APPROXIMATOR WITH GENERAL GRAPHS

In the Introduction we have already seen an example where a more general penalty structure than in the one-dimensional FLSA can be very useful for reconstructing a noisy image. However, we do not need to restrict our attention to a two-dimensional grid. In this section, we will present an algorithm that finds the solution for the FLSA problem with an arbitrary graph $\mathcal{G} = (V, E)$ (with set of vertices $V = \{1, \ldots, n\}$ and edges E) specifying the structure of the penalty parameter on differences. The loss function in this case is

$$L(\mathbf{y}, \boldsymbol{\beta}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \beta_i)^2 + \lambda_1 \sum_{i=1}^{p}|\beta_i| + \lambda_2 \sum_{(i,j)\in E, i<j} |\beta_i - \beta_j| \tag{2.1}$$

so that we penalize $|\beta_i - \beta_j|$ for every edge $(i, j) \in E$. The condition $i < j$ makes sure that we penalize $|\beta_i - \beta_j|$ only once as the edges in the graph are assumed to be undirected.

As a first step, we can see that by using the following theorem we can set $\lambda_1 = 0$ and thereby simplify our problem. The solution for any $\lambda_1 > 0$ can then easily be obtained by soft-thresholding the solution for $\lambda_1 = 0$.

Theorem 1. *Assume that the solution for* $\lambda_1 = 0$ *and* $\lambda_2 > 0$ *is known and denote it by* $\boldsymbol{\beta}(0, \lambda_2)$. *Then the solution for* $\lambda_1 > 0$ *is*

$$\beta_i(\lambda_1, \lambda_2) = \operatorname{sign}(\beta_i(0, \lambda_2))\big(|\beta_i(0, \lambda_2)| - \lambda_1\big)^+ \quad \text{for } i = 1, \ldots, p.$$

The proof of this theorem is presented in the article by Friedman et al. (2007). For the rest of this section, we assume that $\lambda_1 = 0$.

In the following subsections we will first describe the problem in a little more detail, especially wrt *fused* sets of variables. Then we will explain how to *merge* and *split* sets of fused variables. While it will be relatively simple to determine when to merge sets, it is much more complicated when to split them and we will give some motivation as to why the problem of splitting sets of fused variables can be tackled by maximum flow algorithms from graph theory. After incorporating this into the final algorithm, we present an approximate version of our method that is faster on large datasets by sacrificing some precision. Finally, we use our new algorithm on simulated data and compare its speed and accuracy to the other methods.

2.1 SETS OF FUSED VARIABLES

In our loss function

$$L(\mathbf{y}, \boldsymbol{\beta}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \beta_i)^2 + \lambda_2 \sum_{(i,j)\in E, i<j} |\beta_i - \beta_j| \tag{2.2}$$

the penalty term shrinks coefficients toward each other and for sufficiently large values of λ_2 they are even exactly equal. Therefore, we can have sets of variables whose nodes are connected in the underlying graph $\mathcal{G}$ and which are equal to each other. Loosely speaking, we will refer to these sets as being *fused*, which refers to the fact that for each such set F and for all coefficients $\beta_i(\lambda_2)$ for $i \in F$ we have $\beta_i(\lambda_2) = \beta_{F_i}(\lambda_2)$ and therefore they all behave exactly the same for increasing λ_2 (at least until we have to split F).

When defining the sets of fused variables for a penalty parameter λ_2^0, we want to make sure that every variable belongs to some set (even if it is of size 1), that sets are mutually exclusive, and each set is connected in $\mathcal{G}$ as well as that they remain together for at least another increase in λ_2 of ε, that is, in $[\lambda_2^0, \lambda_2^0 + \varepsilon]$. As a formal definition, we have:

Definition 1: Let $n_F(\lambda_2^0)$ be the number of sets of fused variables for penalty parameter λ_2^0. Then the sets $F_i(\lambda_2^0)$, $i = 1, \ldots, n_F(\lambda_2^0)$, are called valid, if the following conditions hold:

1. $\bigcup_{i=1}^{n_F(\lambda_2^0)} F_i(\lambda_2^0) = \{1, \ldots, n\}$,
2. $F_i(\lambda_2^0) \cap F_j(\lambda_2^0) = \emptyset, i \neq j$,
3. If $k, l \in F_i(\lambda_2^0)$, then k and l are connected in $\mathcal{G}$ by only going over nodes in $F_i(\lambda_2^0)$, that is, k, l are connected in $\mathcal{G}|_{F_i(\lambda_2^0)}$, the subgraph of $\mathcal{G}$ induced by $F_i(\lambda_2^0)$.
4. If $k, l \in F_i(\lambda_2^0)$, then $\hat{\beta}_k(\lambda_2^0) = \hat{\beta}_l(\lambda_2^0)$ and if $k \in F_{i,}(\lambda_2^0), l \in F_j(\lambda_2^0), i \neq j$ and $F_i(\lambda_2^0)$ and $F_j(\lambda_2^0)$ have a connecting edge, then $\hat{\beta}_k(\lambda_2) \neq \hat{\beta}_l(\lambda_2)$ for all penalty parameters $\lambda_2 \in (\lambda_2^0, \lambda_2^0 + \varepsilon)$ for some $\varepsilon > 0$.

2.2 SLOPE OF $\hat{\boldsymbol{\beta}}(\lambda_2)$

Now, assuming that the sets of variables $F_i(\lambda_2)$ that are fused are given at λ_2, we will determine the slope of the optimal $\hat{\beta}_{F_i}(\lambda_2)$ with respect to λ_2. In order to do this, we will incorporate the sets of fused coefficients into the loss function in (2.1). Setting $\beta_k = \beta_{F_i}$ for all $k \in F_i(\lambda_2)$, the loss function becomes (β_k here is just the variable, not the solution and therefore does not depend on λ_2):

$$\begin{aligned} L_{F,\lambda_2}(\mathbf{y}, \boldsymbol{\beta}) = &\sum_{i=1}^{n_F(\lambda_2)} \Bigg(\sum_{j\in F_i(\lambda_2)} \big(y_j - \beta_{F_i}\big)^2 \Bigg) \\ &+ \lambda_2 \sum_{i<j} \big|\{(k,l) \in E : k \in F_i(\lambda_2), l \in F_j(\lambda_2)\}\big| \big|\beta_{F_i} - \beta_{F_j}\big|. \end{aligned} \tag{2.3}$$

Here, note that by definition of the sets $F_i(\lambda_2)$, they remain constant for at least some small interval $[\lambda_2, \lambda_2 + \varepsilon]$. Also, we know that for the solution of the loss function $\hat{\boldsymbol{\beta}}(\lambda_2)$ we have

$\hat{\beta}_{F_i}(\lambda_2) \neq \hat{\beta}_{F_j}(\lambda_2)$ for $i \neq j$ and therefore the loss function is differentiable (except for a finite number of λ_2-values where we merge and split sets) with respect to β_{F_i} at the solution $\hat{\beta}_{F_i}(\lambda_2)$. Thus, at $\hat{\beta}_{F_i}(\lambda_2)$, the derivative of $L_{F,\lambda_2}(\mathbf{y}, \boldsymbol{\beta})$ with respect to β_{F_i} is 0, that is,

$$\begin{aligned}
&\frac{\partial L_{F,\lambda_2}(\mathbf{y}, \boldsymbol{\beta})}{\partial \beta_{F_i}}(\lambda_2) \\
&\quad = |F_i(\lambda_2)|\hat{\beta}_{F_i}(\lambda_2) - \sum_{j \in F_i(\lambda_2)} y_j \\
&\qquad + \lambda_2 \sum_{j \neq i} \big|\{(k,l) \in E : k \in F_i(\lambda_2), l \in F_j(\lambda_2)\}\big| \operatorname{sign}\big(\hat{\beta}_{F_i}(\lambda_2) - \hat{\beta}_{F_j}(\lambda_2)\big) \\
&\quad = 0
\end{aligned}$$

for the solutions $\hat{\beta}_{F_i}(\lambda_2)$. By taking the derivative wrt λ_2 and noting that for small changes of λ_2, the sign of $\beta_{F_i} - \beta_{F_j}$ does not change, it is possible to determine $\partial \hat{\beta}_{F_i}(\lambda_2)/\partial \lambda_2$ as

$$\frac{\partial \hat{\beta}_{F_i}(\lambda_2)}{\partial \lambda_2} = -\frac{\sum_{j \neq i} |\{(k,l) \in E : k \in F_i(\lambda_2), l \in F_j(\lambda_2)\}| \operatorname{sign}(\hat{\beta}_{F_i(\lambda_2)} - \hat{\beta}_{F_j(\lambda_2)})}{|F_i(\lambda_2)|}, \quad (2.4)$$

which is constant as long as the $F_i(\lambda_2)$ do not change. Therefore, the solution $\hat{\beta}_{F_i}(\lambda_2)$ is a piecewise linear function. At the breakpoints of the solution path, the sets of fused variables change. As we will see in more detail later, there are two things that can happen:

- $\hat{\beta}_{F_i}(\lambda_2) = \hat{\beta}_{F_j}(\lambda_2)$ for some $i \neq j$ with $F_i(\lambda_2)$ and $F_j(\lambda_2)$ connected, which violates condition 4 of Definition 1. In this case, fuse sets $F_i(\lambda_2)$ and $F_j(\lambda_2)$.
- A set $F_i(\lambda_2)$ has to be broken up into two smaller subsets.

The criterion for fusing sets of variables is very simple. Just look if two sets $F_i(\lambda_2)$ and $F_j(\lambda_2)$ are connected in $\mathcal{G}$ and have $\hat{\beta}_{F_i}(\lambda_2) = \hat{\beta}_{F_j}(\lambda_2)$, then merge the two sets. The situation for splitting sets is a little more complicated and will be treated below.

2.2.1 The Hitting Time h

Based on the slopes of $\hat{\beta}_i(\lambda_2)$, we can also calculate when the next two sets will have to be fused. We assume that the algorithm is currently at penalty parameter λ_2^0. We will refer to the value of λ_2 at which sets $F_i(\lambda_2^0)$ and $F_j(\lambda_2^0)$ would have to be fused given the current trajectory of $\hat{\beta}_{F_i}$ as $h_{ij}(\lambda_2^0)$, which is possibly infinite.

For penalty parameter λ_2^0 we have a valid grouping $F_1(\lambda_2^0), \ldots, F_{n_F}(\lambda_2^0)$ and solutions $\hat{\beta}_k(\lambda_2^0)$ as well as $\hat{\tau}_{kl}(\lambda_2^0)$. Given this, it is easy to calculate when two sets that are connected by an edge hit. For this, let the hitting time of groups i and j at λ_2^0 be

$$h_{ij}(\lambda_2^0) = \begin{cases} (\hat{\beta}_{F_i}(\lambda_2^0) - \hat{\beta}_{F_j}(\lambda_2^0))/\big(\frac{\partial \hat{\beta}_{F_j}}{\partial \lambda_2}(\lambda_2^0) - \frac{\partial \hat{\beta}_{F_i}}{\partial \lambda_2}(\lambda_2^0)\big) + \lambda_2^0, \\ \qquad \text{if } \exists k \in F_i(\lambda_2^0), l \in F_j(\lambda_2^0) \text{ with } (k,l) \in E \\ \infty, \quad \text{otherwise.} \end{cases}$$

If $h_{ij}(\lambda_2^0) < \lambda_2^0$, then given the current slopes, these groups are moving apart. If $h_{ij}(\lambda_2^0) = \lambda_2^0$, then $\hat{\beta}_{F_i}(\lambda_2^0) = \hat{\beta}_{F_j}(\lambda_2^0)$. However, as we assumed that this is a valid grouping, from

Definition 1 we get that for some $\varepsilon > 0$, for any $\lambda_2 \in (\lambda_2^0, \lambda_2^0 + \varepsilon)$ we have $\hat{\beta}_{F_i}(\lambda_2) \neq \hat{\beta}_{F_j}(\lambda_2)$ and as the trajectories are piecewise affine, the groups $F_i(\lambda_2^0)$ and $F_j(\lambda_2^0)$ move apart. Therefore, defining

$$h_i(\lambda_2^0) = \min_{j:h_{ij}>\lambda_2^0} h_{ij}(\lambda_2^0)$$

and

$$h(\lambda_2^0) = \min_i h_i(\lambda_2^0)$$

we have that group i will hit another group at $h_i(\lambda_2^0)$ and any two groups will hit at $h(\lambda_2^0)$, but not before. However, it is also possible that it is necessary to split a set before this, which is what we are looking at in the next section.

2.3 Splitting Sets of Variables

2.3.1 Motivation for the Maximum-Flow Algorithm

Here we want to determine when a set of fused variables $F_i(\lambda_2)$ has to be split up. We will solve this problem by applying algorithms that solve so-called *maximum-flow* problems in graph theory. In general, any algorithm that solves a maximum-flow problem can be used (e.g., the augmenting path algorithm by Ford–Fulkerson, or the preflow-push algorithm by Goldberg and Tarjan, among others) and a good overview can be found in the book by Jungnickel (2007).

In order to motivate using the max-flow algorithm, we assume that we have a set of nodes F for which $\hat{\beta}_i = \hat{\beta}_j$ for all $i, j \in F$, that is, those coefficients are fused. Now we look at the problem as the subgraph $\mathcal{G}_F = \mathcal{G}|_F$ induced by F. Each of the nodes in $\mathcal{G}_F$ corresponds to a coefficient in F and each of the edges (i, j) to a penalized difference $\lambda_2|\beta_i - \beta_j|$. Now when we are increasing λ_2, the nodes in $\mathcal{G}_F$ are being pulled into different directions (i.e., "up" or "down") by the edges that connect them to nodes outside of F in the full graph $\mathcal{G}$. This pull puts "strain" on the edges between the nodes in F (as those edges are keeping the $\hat{\beta}_i$ for $i \in F$ equal to each other) and the strain "flows" from edges that are being pulled upward to edges that are being pulled downward. The strain that an edge can take before "breaking" is equal to λ_2. So, for increasing values of λ_2, each edge is being put under additional strain by the pull on the nodes; however, the strain it can take also increases. In order to determine if or when we have to break up a set F, we have to determine how the strain flows from nodes being pulled upward to nodes being pulled downward under the condition that none of the edges break. So, intuitively this is a flow-problem and we will see later that we can formulate this problem rigorously in terms of maximum-flows. The condition for breaking a set apart is then that we cannot find a flow anymore that ensures that none of the edges break under the strain.

2.3.2 Subgradient Equations

In order to make this intuition more rigorous, we have to look at the derivative of the unconstrained loss function $L_{\lambda_2}(\mathbf{y}, \boldsymbol{\beta})$ in (2.1). Of course, the unconstrained loss function

is not differentiable in general, so that we will look at subgradients instead. An overview of subgradients can be found in the book by Bertsekas (1999) and a brief introduction with the relevant calculations is also provided in Section 7 of the Online Supplement.

For the subgradients, a necessary and sufficient condition for β_k to be optimal is that

$$\frac{\partial L_{\lambda_2}(\mathbf{y}, \hat{\boldsymbol{\beta}})(\lambda_2)}{\partial \hat{\beta}_k} = \hat{\beta}_k(\lambda_2) - y_k + \lambda_2 \sum_{(k,l)\in E} \hat{t}_{kl}(\lambda_2) = 0 \quad \text{for } k = 1, \ldots, n, \tag{2.5}$$

where $\hat{t}_{kl}(\lambda_2) = \operatorname{sign}(\hat{\beta}_k(\lambda_2) - \hat{\beta}_l(\lambda_2))$ for $\hat{\beta}_k(\lambda_2) \neq \hat{\beta}_l(\lambda_2)$, $\hat{t}_{kl}(\lambda_2) \in [-1, 1]$ for $\hat{\beta}_k(\lambda_2) = \hat{\beta}_l(\lambda_2)$ as well as for notational convenience $\hat{t}_{kl}(\lambda_2) = -\hat{t}_{lk}(\lambda_2)$. We will refer to a vector of all such $\hat{t}_{kl}(\lambda_2)$ as $\hat{\mathbf{t}}(\lambda_2)$. We now want to identify an interval $[\lambda_2^0, \lambda_2^1]$ such that they are also satisfied by affine functions of $\hat{\boldsymbol{\beta}}(\lambda_2)$ and $\hat{\boldsymbol{\tau}}(\lambda_2) = \lambda_2 \hat{\mathbf{t}}(\lambda_2)$ and we denote their slope by $(\partial \hat{\beta}_k / \partial \lambda_2)(\lambda_2^0)$ as well as $(\partial \hat{\tau}_{kl} / \partial \lambda_2)(\lambda_2^0)$. Inserting this into the equation above we get

$$\begin{aligned}\frac{\partial L_{\lambda_2}(\mathbf{y}, \hat{\boldsymbol{\beta}})(\lambda_2)}{\partial \hat{\beta}_k} &= \hat{\beta}_k(\lambda_2^0) + (\lambda_2 - \lambda_2^0)\frac{\partial \hat{\beta}_k}{\partial \lambda_2}(\lambda_2) - y_k \\ &\quad + \sum_{(k,l)\in E} \hat{\tau}_{kl}(\lambda_2^0) + (\lambda_2 - \lambda_2^0)\frac{\partial \hat{\tau}_{kl}}{\partial \lambda_2}(\lambda_2^0) = 0 \quad \text{for } k = 1, \ldots, n, \end{aligned} \tag{2.6}$$

which when we require that it holds for an interval $\lambda_2 \in [\lambda_2^0, \lambda_2^1]$ immediately yields that

$$\frac{\partial \hat{\beta}_k}{\partial \lambda_2}(\lambda_2^0) + \sum_{(k,l)\in E} \frac{\partial \hat{\tau}_{kl}}{\partial \lambda_2}(\lambda_2^0) = 0 \quad \text{for } k = 1, \ldots, n.$$

Given a grouping $F_i(\lambda_2)$, we see that $\partial \hat{\tau}_{kl}(\lambda_2^0)/\partial \lambda_2 = \hat{t}_{kl}(\lambda_2^0) = \pm 1$ for $k \in F_i(\lambda_2^0)$ and $l \in F_j(\lambda_2^0)$ with $i \neq j$. With $k \in F_i(\lambda_2)$, we can write

$$\begin{aligned}&\frac{\partial \hat{\beta}_k}{\partial \lambda_2}(\lambda_2^0) + \sum_{i\neq j} \sum_{(k,l)\in E: l\in F_j(\lambda_2^0)} \hat{t}_{kl}(\lambda_2^0) + \sum_{(k,l)\in E: l\in F_i(\lambda_2^0)} \frac{\partial \hat{\tau}_{kl}}{\partial \lambda_2}(\lambda_2^0) \\ &\quad = 0 \quad \text{for } k = 1, \ldots, n. \end{aligned} \tag{2.7}$$

Therefore, for any solution of (2.7), we can construct an affine solution to (2.5) that is valid for an interval. However, the side conditions on the $\hat{t}_{kl}$ have to be satisfied as well during the whole interval $[\lambda_2^0, \lambda_2^1]$. There are two ways in which this can break down for some $\lambda_2 > \lambda_2^0$:

1. There are two groups $F_i(\lambda_2)$ and $F_j(\lambda_2)$ for which there exist $k \in F_i(\lambda_2)$ and $l \in F_j(\lambda_2)$ with $(k, l) \in E$, that is, $F_i(\lambda_2)$ and $F_j(\lambda_2)$ have at least one edge connecting them. For these two groups, we have that $\hat{t}_{kl}(\lambda_2) = \pm 1$ does not hold anymore for all $k \in F_i(\lambda_2)$ and $l \in F_j(\lambda_2)$. In this case, the groups $F_i(\lambda_2)$ and $F_j(\lambda_2)$ have to be fused.

2. There is a group $F_i(\lambda_2)$ for which for increasing λ_2, the condition $-1 \leq \hat{t}_{kl}(\lambda_2) \leq 1$ or equivalently $-\lambda_2 \leq \hat{\tau}_{kl}(\lambda_2) \leq \lambda_2$ for all $k, l \in F_i(\lambda_2)$ cannot be satisfied anymore. In this case, the group has to be split into two smaller subgroups. How to decide when this is the case and how to identify the two new subgroups will be treated below.

It is very easy to detect when the first case occurs, and as we already mentioned above we then just fuse the two groups. How to detect the second case will be discussed below. However, first, we will discuss the identifiability problems of $\hat{\mathbf{t}}(\lambda_2)$.

2.3.3 Identifiability of $\hat{\mathbf{t}}(\lambda_2)$

We can quickly see that although we only have n equations in (2.5) as well as (2.7), for general graphs we have considerably more variables $\hat{t}_{kl}(\lambda_2)$, thus making them unidentifiable. In order to see that this is in fact not a problem, we first note that in (2.5), $\boldsymbol{\beta}$ is optimal if there exist any $\hat{\mathbf{t}}(\lambda_2)$ that satisfy the equations. Therefore, although there may be infinitely many $\hat{\mathbf{t}}(\lambda_2)$ that are solutions, knowing a single one is sufficient. Therefore the identifiability issue is not a problem in this case. As we have seen in the derivation of (2.7) above, any solution for it yields a linear function that solves (2.5) in an interval. It remains to show that the identifiability issue does not negatively affect our ability to correctly detect when a set has to be split.

When we do not have a solution for (2.7) that satisfies the conditions on $\hat{\mathbf{t}}(\lambda_2)$, we want to conclude that then it is necessary to split the set for which the conditions fail. Here, the currently used value of $\hat{\mathbf{t}}(\lambda_2)$ may be important because it could be possible that while there is no possible solution for (2.7) that satisfies the conditions for one value of $\hat{\mathbf{t}}(\lambda_2)$, it may be possible for another. In the following, we will give some arguments as to why this is not the case and a strict proof is provided in the Online Supplement in Section 8.2.

In order to see this, we first note the continuity of the solution $\boldsymbol{\beta}(\lambda_2)$ (which is unique) wrt λ_2, which follows from the strict convexity of the problem. The central question is then whether we can identify the exact value of λ_2 at which a set has to be split despite the identifiability issue. Consider a group $F_i(\lambda_2)$ that exists for $\lambda_2 \in [\lambda_2^0, \lambda_2^1]$ and at λ_2^1 breaks into the sets $R_i(\lambda_2^1)$ and $S_i(\lambda_2^1)$ (and that all other groups do not change in that interval). Then it is easy to see that the set of all solutions

$$\mathcal{S} = \{(\boldsymbol{\beta}, \boldsymbol{\tau}, \lambda_2) : (\boldsymbol{\beta}, \boldsymbol{\tau}, \lambda_2) \text{ satisfies Equation (2.5)}\}$$

is convex. But then no matter which $(\boldsymbol{\beta}^0, \boldsymbol{\tau}^0, \lambda_2^0) \in \mathcal{S}$ we are using, there is always an affine function wrt λ_2 linking to the breakpoint $(\boldsymbol{\beta}^1, \boldsymbol{\tau}^1, \lambda_2^1) \in \mathcal{S}$ because of the convexity. Therefore, when we are currently at $(\boldsymbol{\beta}^0, \boldsymbol{\tau}^0, \lambda_2^0)$ in our path with $\lambda_2^0 < \lambda_2^1$, we will be able to find a solution to (2.7) that satisfies the constraints. This is equivalent to stating that if we do not find a solution to (2.7) that satisfies the constraints, no solution $(\boldsymbol{\beta}^1, \boldsymbol{\tau}^1, \lambda_2^1)$ exists for any $\lambda_2^1 > \lambda_2^0$, thus implying that we have to split a set F_i in this case. Therefore, despite possibly having infinitely many correct solutions for $\hat{t}(\lambda_2^0)$ (or equivalently $\hat{\tau}(\lambda_2^0)$), we can always find a piecewise linear path through the solution space.

In the next section, we will now treat the problem of how to find a solution (among possibly infinitely many) for (2.7) that satisfies the constraints on $\hat{\mathbf{t}}(\lambda_2)$ and how to detect when such a solution does not exist.

2.3.4 The Maximum-Flow Problem

In order to decide when to split sets, it is necessary to find solutions for $\hat{t}_{kl}(\lambda_2)$ (or equivalently $\hat{\tau}_{kl}(\lambda_2)$) in (2.7). For k and l that are not in the same group, this will be easy

as then $\hat{t}_{kl}(\lambda_2) = \text{sign}(\hat{\beta}_{F_i}(\lambda_2) - \hat{\beta}_{F_j}(\lambda_2))$ for $k \in F_i(\lambda_2)$ and $l \in F_j(\lambda_2)$. For k and l in the same group, that is, $k, l \in F_i(\lambda_2)$, we will see that $\hat{\tau}_{kl}(\lambda_2)$ is an affine function of λ_2 and the slope can be calculated by solving a maximum-flow problem in graph theory. The same calculations can also be used to identify when a group has to be split (more on this later). As maximum-flow problems are a well-studied area, fast algorithms for this problem such as the push-relabel algorithm, among others, exist and can be used here (see Cormen et al. 2001; Jungnickel 2007). Before going into more details, assume that for penalty parameter λ_2, we know $\hat{t}_{kl}(\lambda_2)$ for $k, l \in F_i(\lambda_2)$, $i = 1, \ldots, n_F(\lambda_2)$, that solve (2.5). Now, for notational convenience, define

$$\hat{p}_k(\lambda_2) = -\sum_{i \neq j} \sum_{(k,l) \in E : l \in F_j(\lambda_2)} \hat{t}_{kl} - \frac{\partial \hat{\beta}_{F_i}(\lambda_2)}{\partial \lambda_2}$$

for $k \in F_i(\lambda_2)$ and call it the *pull* on node k as it measures the influence other variables in neighboring groups that are connected to k have on node k. Using these definitions, (2.7) can be written as

$$\sum_{(k,l) \in E : l \in F_i(\lambda_2)} \frac{\partial \hat{\tau}_{kl}(\lambda_2)}{\partial \lambda_2} = \hat{p}_k(\lambda_2) \quad \text{for } k = 1, \ldots, n. \tag{2.8}$$

Additionally, we also have to ensure that $\hat{\tau}_{kl}(\lambda_2) \in [-\lambda_2, \lambda_2]$ by imposing restrictions on $\partial \hat{\tau}_{kl}(\lambda_2)/\partial \lambda_2$. In order to motivate the definition of the max-flow graph below, we restrict our view to the graph $\mathcal{G}|_{F_i(\lambda_2)}$. If $\hat{p}_k(\lambda_2) > 0$, connect node k to the source, otherwise to the sink. Then (2.8) just states that the flow going into node k from the source is also going out to other nodes—considering $\partial \hat{\tau}_{kl}(\lambda_2)/\partial \lambda_2$ as the flow from node k to node l. The capacities in the graph then have to be chosen in order to enforce the interval condition on $\hat{\tau}_{kl}(\lambda_2)$ and that the flows coming from the source or going to the sink are exactly $\hat{p}_k(\lambda_2)$ when at maximum capacity.

For each of these equations, we can see that they only involve variables that belong to the same groups, that is, if $k \in F_i(\lambda_2)$, then all l used for the variables $\hat{\tau}_{kl}(\lambda_2)$ are also in $F_i(\lambda_2)$ and thus these n equations are separable according to the groups $F_i(\lambda_2)$. Therefore, for each of the groups $F_i(\lambda_2)$, we will solve a separate maximum-flow problem to find $\partial \hat{\tau}_{kl}(\lambda_2)/\partial \lambda_2$ and determine if it is necessary to split the group.

In order to specify the maximum-flow problem, we need to define the underlying graph which consists of vertices, edges, and capacities on each edge in both directions, which we do at penalty parameter λ_2^0. First, in our notation, the graph $\mathcal{G}_i(\lambda_2^0) = \mathcal{G}|_{F_i(\lambda_2^0)}$ is the graph $\mathcal{G}$ restricted to the nodes in the set $F_i(\lambda_2^0)$. Now let $\tilde{\mathcal{G}}_i(\lambda_2^0, \delta) = (\tilde{V}_i, \tilde{E}_i, \tilde{C}_i)$ be the vertices, edges, and capacities of the ith problem. For these, we define:

Vertices: To each of the subgraphs $\mathcal{G}_i(\lambda_2^0)$, we add an artificial source node r and sink node s, such that $\tilde{V}_i(\lambda_2) = V_i(\lambda_2^0) \cup \{r, s\}$.

Edges: For the edges $\tilde{E}_i(\lambda_2^0)$ we use all the edges in $E_i(\lambda_2^0)$ and will add additional edges connecting each of the nodes in $V_i(\lambda_2^0)$ to either the source or the sink. In order to motivate which nodes will be connected to which, note that at the end, we will set

$\partial\hat{\tau}_{kl}(\lambda_2^0)/\partial\lambda_2 = \hat{f}_{kl}(\lambda_2^0)$ for the maximal flow $\hat{f}_{kl}(\lambda_2^0)$ from node k to node l. As the excess flow through every node (except for the source and sink) has to be 0, a node has an edge with the source if the right side of (2.8) is greater than 0 and an edge with the sink if it is less than 0. Thus

$$\tilde{E}_i(\lambda_2^0) = E_i(\lambda_2^0) \cup \{(r,l) : \hat{p}_l(\lambda_2^0) > 0\} \cup \{(k,s) : \hat{p}_k(\lambda_2^0) < 0\}.$$

As usual, all the edges are undirected.

Capacities: Of course, the capacities on the edges have to be defined as well and we will do so for each direction separately. We want to define the capacities on the source and the sink such that whenever all nodes coming from the source are at maximum capacity, then (2.8) holds. As the sum of all flows going into a node is the same as the sum of all flows leaving the node, we can achieve this by choosing the capacities as suggested by the right side of (2.8). Then for the edges coming from the source r we therefore set for all l for which $(r,l) \in \tilde{E}_i(\lambda_2^0)$,

$$(c_{rl}(\lambda_2^0), c_{lr}(\lambda_2^0)) = (\hat{p}_l(\lambda_2^0), 0)$$

and correspondingly for the edges to the sink set for all k with $(k,s) \in \tilde{E}_i(\lambda_2^0)$,

$$(c_{ks}(\lambda_2^0), c_{sk}(\lambda_2^0)) = (-\hat{p}_k(\lambda_2^0), 0).$$

Using this and noting that we will define $\partial\hat{\tau}_{kl}(\lambda_2^0)/\partial\lambda_2 = \hat{f}_{kl}(\lambda_2^0)$, then (2.8) is satisfied iff all flows from the source and all to the sink are at maximum capacity (using that the net flow in a node is 0).

In addition to this, we also want to ensure that for $k,l \in F_i(\lambda_2^0)$, the flows are constrained such that $-\lambda_2 \le \hat{\tau}_{kl}(\lambda_2) \le \lambda_2$ in the interval $\lambda_2 \in [\lambda_2^0, \lambda_2^0 + \delta]$ for some δ to be defined later. In order to ensure this, we need the restrictions

$$\begin{aligned} -c_{lk}(\delta, \lambda_2^0) = -1 - \frac{\hat{\tau}_{kl}(\lambda_2^0) - (-\lambda_2^0)}{\delta} &\le \frac{\partial\hat{\tau}_{kl}}{\partial\lambda_2}(\lambda_2^0) \\ &\le 1 + \frac{\lambda_2^0 - \hat{\tau}_{kl}(\lambda_2^0)}{\delta} = c_{kl}(\delta, \lambda_2^0) \end{aligned}$$

for $k,l \in F_i(\lambda_2^0)$ and therefore we also have the capacities $c_{kl}(\delta, \lambda_2^0)$ for all edges not going to the source or the sink. Using all this set $\tilde{C}_i = \{c_{kl}(\delta, \lambda_2^0) : k,l \in \tilde{V}_i(\lambda_2^0)\}$.

Here it is interesting to note that

$$\sum_{k \in F_i} \hat{p}_k(\lambda_2) = 0, \tag{2.9}$$

which is easy to see by summing up (2.7), the definition of $\hat{p}_k(\lambda_2)$, and that $\hat{t}_{kl}(\lambda_2) = -\hat{t}_{lk}(\lambda_2)$. Therefore, the sum of all capacities going out of the source is equal to the sum of all capacities going into the sink and thus a flow that is maximal for all source edges is also maximal for all sink edges.

In this maximum-flow problem, we still have to specify δ. From the previous section, we have already computed the hitting time for set $F_i(\lambda_2^0)$. For penalty parameter λ_2^0, we

therefore in a first step want to see if we have to split set $F_i(\lambda_2^0)$ before the hitting time $h_i(\lambda_2^0)$. For this, set $\delta = h_i(\lambda_2^0) - \lambda_2^0 + \varepsilon$ in the maximum-flow graph above for some small $\varepsilon > 0$ and solve it. If all edges coming from the source and going to the sink are at maximum capacity, then we know we have found trajectories $\partial \hat{\tau}_{kl}(\lambda_2^0)/\partial\lambda_2$ such that (2.8) holds as well as $\hat{\tau}_{kl}(\lambda_2) \in [-\lambda_2, \lambda_2]$ for $\lambda_2 \in [\lambda_2^0, \lambda_2^0 + \delta]$. Therefore, the set $F_i(\lambda_2^0)$ will be fused before it would be necessary to split it. This computation will be repeated every time $h_i(\lambda_2)$ gets updated for $\lambda_2 > \lambda_2^0$ (e.g., by merging).

However, it is possible that no such solution exists where the source edges are at maximum. This means that $F_i(\lambda_2^0)$ has to be split before $h_i(\lambda_2^0)$, but we still have to find the exact δ at which a split has to be made. This δ is the maximum value, for which (2.8) and $-c_{lk}(\delta, \lambda_2) \le \frac{\partial \hat{\tau}_{kl}}{\partial \lambda_2} \le c_{kl}(\delta, \lambda_2)$ hold for all $k, l \in F_i(\lambda_2^0)$. This is equivalent to solving the program

$$\begin{aligned} &\text{minimize } 1/\delta \\ &\text{subject to} \quad \sum_{(k,l)\in E: l\in F_i(\lambda_2^0)} \hat{f}_{kl}(\lambda_2^0) = \hat{p}_k(\lambda_2^0), \qquad k \in F_i(\lambda_2^0), \\ &\qquad\qquad -c_{lk}(\delta, \lambda_2^0) \le \hat{f}_{kl}(\lambda_2^0) \le c_{kl}(\delta, \lambda_2^0), \qquad k, l \in F_i(\lambda_2^0) \end{aligned} \tag{2.10}$$

which keeping λ_2^0 fixed is a linear program in $1/\delta$ and $\hat{f}_{kl}(\lambda_2^0)$. We can solve this either using standard convex optimization techniques or by iteratively using the maximum-flow problem. In the iterative application of the maximum-flow problem, we first guess a possible δ (i.e., the one used before) and then depending on how well it worked we update our guess until we reach the optimal solution. This procedure is just a simple root searching method and as it is not important for the algorithm beyond the linear program it is solving, we have moved its exact description and proof to the Online Supplement in Section 8.4. As this method is iterative, the maximum-flow algorithm has to be applied several times to find the next split point. In our simulations, six runs were needed on average and a more detailed analysis with taking account of the size of the maximum-flow graph can be found in the Online Supplement in Section 8.4. It should also be mentioned that for large graphs, finding a maximum flow can be slow and we will present an approximate method that avoids this problem below.

Using this procedure for picking the optimal δ, in the following by $\tilde{\mathcal{G}}_i(\lambda_2^0)$ we refer to the graph $\tilde{\mathcal{G}}_i(\lambda_2^0, \delta)$ with δ chosen as described above.

For the rest of the algorithm, we just need the flows $\hat{f}_{kl}(\lambda_2^0)$ from the linear program as well as the exact value of δ. In either case (exact or using the hitting time), we define

$$v_i(\lambda_2^0) = \delta + \lambda_2^0 \tag{2.11}$$

as the splitting time for set $F_i(\lambda_2^0)$. In the case where $v_i(\lambda_2^0) > h_i(\lambda_2^0)$, the exact value of δ that would be the solution of the linear program does not matter so that the single application of the maximum-flow algorithm is appropriate.

In the following, the solution to the flow problem will be referred to as $\hat{f}_{kl}(\lambda_2)$, which is the flow from node k to node l, assuming that $k, l \in \tilde{E}_i(\lambda_2)$. Using this result we will now show that the solution path is piecewise-linear. The next theorem guarantees that for an interval, the solutions for $\hat{\beta}_k(\lambda_2)$ and $\hat{\tau}_{kl}(\lambda_2)$ are affine and have the slope as stated above.

Theorem 2. *For some λ_2^0, let $F_1(\lambda_2^0), \ldots, F_{n_F}(\lambda_2^0)$ be a valid grouping of the variables. Let $\tilde{\mathcal{G}}_i(\lambda_2^0)$ be the with $F_i(\lambda_2^0)$ associated maximum-flow graph as defined above. Also let $\hat{\beta}_k(\lambda_2^0)$ and $\hat{\tau}_{kl}(\lambda_2^0)$ be a solution to the FLSA problem for penalty parameter $\lambda_2 = \lambda_2^0$. If $\tilde{\mathcal{G}}_i(\lambda_2^0)$ has a maximum flow for which all flows coming from the source are at maximum capacity (i.e., $\hat{f}_{rl}(\lambda_2^0) = \hat{c}_{rl}(\lambda_2^0)$ for all $(r,l) \in \tilde{E}_i$), and $\frac{\partial \hat{\beta}_{F_i}(\lambda_2^0)}{\partial \lambda_2}$ is as defined in (2.4), then there exists some $\Delta > 0$ such that for any $\lambda_2 \in [\lambda_2^0, \lambda_2^0 + \Delta]$, a solution to the FLSA problem is given by*

$$\hat{\beta}_k(\lambda_2) = \hat{\beta}_k(\lambda_2^0) + \frac{\partial \hat{\beta}_{F_i}}{\partial \lambda_2}(\lambda_2^0) \cdot (\lambda_2 - \lambda_2^0) \quad \text{for } k \in F_i(\lambda_2^0)$$

and

$$\hat{\tau}_{kl}(\lambda_2) = \begin{cases} \hat{\tau}_{kl}(\lambda_2^0) + \hat{f}_{kl}(\lambda_2^0)(\lambda_2 - \lambda_2^0), & \text{for } k,l \in F_i(\lambda_2^0) \text{ for some } i \\ \operatorname{sign}(\hat{\tau}_{kl}(\lambda_2^0))\lambda_2, & \text{otherwise.} \end{cases}$$

The proof of this theorem can be found in the Online Supplement in Section 8.1. Here, please note that the non-uniqueness of $\hat{\tau}_{kl}(\lambda_2)$, which is caused by non-uniqueness of the maximum flow $\hat{f}_{kl}(\lambda_2^0)$, is not a problem for the theorem as discussed in Section 2.3.3. Furthermore, note that even if λ_2^0 is a breakpoint, the derivative $\partial \hat{\beta}_{F_i} / \partial \lambda_2(\lambda_2^0)$ is still well defined. In Definition 1, for $F_1(\lambda_2^0), \ldots, F_{n_F}(\lambda_2^0)$ to be a valid grouping, point 4 requires it to depend on the behavior of $\hat{\boldsymbol{\beta}}$ for $\lambda_2 > \lambda_2^0$, thereby resolving the ambiguity at the breakpoints.

The only item in the previous proof that we have not specified so far is the length of the interval Δ, for which the solution will be linear as described. There are two things that can occur, that would violate the assumptions of Theorem 2. Either sets could have to be merged or a set needs to be split. What we still need to find now is the size of Δ as well as show that merging and splitting sets leads again to valid fused sets according to Definition 1 as required by Theorem 2.

2.3.5 The Hitting Time h and Violation Time v

The hitting time for each of the sets has already been defined in Section 2.2.1 and the violation time was defined in (2.11). The hitting time for all sets is then given by

$$h(\lambda_2^0) = \min_{i=1,\ldots,n_F(\lambda_2^0)} h_i(\lambda_2^0)$$

and the violation time for all sets correspondingly as

$$v(\lambda_2^0) = \min_{i=1,\ldots,n_F(\lambda_2^0)} v_i(\lambda_2^0).$$

Theorem 2 holds as long as the sets $F_i(\lambda_2^0)$ stay the same and the boundary conditions on the $\hat{\tau}_{kl}(\lambda_2)$ are satisfied. For $\lambda_2 \in [\lambda_2^0, \min\{h(\lambda_2^0), v(\lambda_2^0)\}]$, it is guaranteed that no sets are fused or split; therefore the sets do not change in this interval. Furthermore, by the construction of the maximum-flow problem, it is guaranteed that for $\lambda \in [\lambda_2^0, v(\lambda_2^0)]$

the boundary conditions are satisfied as well. Therefore, for Theorem 2 we choose $\Delta = \min\{h(\lambda_2^0), v(\lambda_2^0)\} - \lambda_2^0$ in order to satisfy its conditions.

Furthermore, we also know that if $\lambda_2 = \lambda_2^0 + \Delta$, we have reached either $h(\lambda_2^0)$ or $v(\lambda_2^0)$. We distinguish the following two cases:

Case 1 ($h(\lambda_2^0) \leq v(\lambda_2^0)$)**:** Here, the two sets that hit at $\lambda_2 = h(\lambda_2^0)$ have to be merged.

Case 2 ($h(\lambda_2^0) > v(\lambda_2^0)$)**:** Here, for one of the sets $F_i(\lambda_2^0)$ we have $v_i(\lambda_2^0) = v(\lambda_2^0) = \lambda_2$. By definition of $v_i(\lambda_2^0)$, this is the largest time for which the boundary conditions can be satisfied for set $F_i(\lambda_2)$. Therefore, set $F_i(\lambda_2^0)$ has to be split.

Therefore, we have defined the Δ of the previous theorem and identified the values of λ_2 at which the piecewise-linear solution path has breakpoints.

2.3.6 Adapting the Sets of Fused Variables

Of course, we still have to specify how to exactly split a set, for which the source edges are not at capacity, into two smaller subsets. Assume that $F_i(\lambda_2^0)$ is the set that has to be split. For this, we calculate the associated maximum-flow graph $\tilde{\mathcal{G}}_i(\lambda_2^0, 0)$ where we set $\delta = 0$ (we know that we have to split the set, we just want to find out how). Here note that for the capacities we define $0/0 = 0$ and $1/0 = \infty$. Then define the set

$$R_i(\lambda_2^0) = \{l \in F_i(\lambda_2^0) : r \text{ connected to } l \text{ by an augmenting path in } \tilde{\mathcal{G}}_i(\lambda_2^0, 0)\},$$

where the augmenting path is defined with respect to the maximal flow $\hat{f}_{kl}(\lambda_2^0)$; that is, for each node $l \in R_i(\lambda_2^0)$ there exists a path from the source r to l using only edges for which the flow is not at capacity. The complement of $R_i(\lambda_2^0)$ with respect to $F_i(\lambda_2^0)$ is defined as $S_i(\lambda_2^0) = F_i(\lambda_2^0) \backslash R_i(\lambda_2^0)$. Then we divide the set $F_i(\lambda_2^0)$ into the two subsets $R_i(\lambda_2^0)$ and $S_i(\lambda_2^0)$.

Now it remains to be shown that fusing or splitting sets as described above will yield sets of fused variables that satisfy the assumptions of Theorem 2. In particular, whenever we are at a breakpoint, that is, have to fuse or split sets or both, we propose the following procedure for adapting the sets of fused variables:

1. If there are sets $F_i(\lambda_2^0)$ and $F_j(\lambda_2^0)$ for which $\exists k \in F_i(\lambda_2^0)$ and $l \in F_j(\lambda_2^0)$ with $(k, l) \in E$ and $\hat{\beta}_{F_i}(\lambda_2^0) = \hat{\beta}_{F_j}(\lambda_2^0)$, then fuse these sets into a new set $\tilde{F}_{ij}(\lambda_2^0) = F_i(\lambda_2^0) \cup F_j(\lambda_2^0)$ if $(\partial \hat{\beta}_{F_i}/\partial \lambda_2(\lambda_2^0)) - (\partial \hat{\beta}_{F_j}/\partial \lambda_2)(\lambda_2^0) \leq 0$ and $\hat{t}_{kl}(\lambda_2^0) = 1$.
2. If there is a set $F_i(\lambda_2^0)$ for which in the associated maximal-flow graph not all edges coming from the source are at maximal capacity, then split $F_i(\lambda_2^0)$ in the two subsets $R_i(\lambda_2^0)$ and $S_i(\lambda_2^0)$ as described above.
3. Iterate steps 1 and 2 until nothing changes.

Using this procedure we can now show that adapting the sets in this way is correct.

Proposition 1. *Assume that we perform the merge and split steps as described above. Then, the algorithm stops after a finite number of merge and split steps and the resulting sets of variables are valid by Definition 1 and satisfy the assumptions of Theorem 2.*

Again, the proof can be found in the Online Supplement in Section 8.2. Putting all this together, we have shown that the solutions are piecewise-linear and how to change the sets of fused variables at the breakpoints. So overall, using this algorithm we can calculate the entire solution path by iterating Theorem 2 and Proposition 1.

2.3.7 Complexity of the Algorithm

Calculating the complexity for the general algorithm is difficult as it depends on the exact sequence of merges and splits. Therefore, here we will just look at some good-case and bad-case scenarios to get a better sense of the computational complexity. In addition to this, we later present timing results on our implementation of the method to give a better impression of real-world performance.

The computational cost of the maximum-flow problem dominates the whole algorithm so that it is sufficient to look at the cost of executing the max-flow calculations. If we assume that we are using the max-flow algorithm with the best complexity available (which is $O(|V||E|)\log(|V|)$ using the Dinitz blocking flow algorithm with dynamic trees; see Sleator and Tarjan 1983) we get a good-case complexity of $O(n^2\log(n))$ and a bad-case complexity of $O(n^3\log(n))$ (where n is the number of coefficients in our problem, assuming the maximum degree of a node in the graph is bounded and independent of n). Here in the good case we assume that the merges occur as in a balanced binary tree so that the average size of the fused sets on which the max-flow algorithm has to be run remains only about $\log(n)$. In the bad-case scenario we assume that the nodes get added sequentially to the largest set so that the average size of the fused sets is about $n/2$. Here we assume when iteratively applying the maximum-flow algorithm, the number of iterations needed is independent of $|V|$ for $|V| > 1000$ and relatively low. In our simulations below, we never needed more than 12 iterations and on average six were sufficient. Further details about the assumptions and calculations are provided in the Online Supplement in Sections 8.3 and 8.4.

2.4 Outline of the Algorithm

In the previous sections, we have seen how to derive the entire solution path of the general Fused Lasso Signal Approximator. If we put all those steps together, we get the outline of the algorithm for the general FLSA presented in Algorithm 1.

It should be noted that this is a basic outline of the algorithm and there is room for considerable efficiency gains when implementing it. Most importantly, the hitting times $h_{i,j}$ only have to be updated if either the set F_i or F_j has changed since the last calculation. The same is true for the maximum-flow problems. The flows only have to be updated if the underlying set has changed or a violation of a constraint was triggered. Therefore, in every iteration only a small number of sets is involved in the calculation and the computations can be done quickly. Especially for larger sets, the computationally most expensive step is

Algorithm 1: General FLSA path algorithm.

initialize
  $\lambda_2 = 0$, $n_F = n$;
  $\beta_k = y_k$, $F_k = \{k\}$ for $k = 1, \ldots, n$;
  $\tau_{kl} = 0$ for $k, l = 1, \ldots, n$; $k \neq l$;
end
while $n_F > 1$ **do**
  Update β_{F_i} and τ_{kl};
  Calculate the derivatives of β_{F_i} wrt λ_2 for $i = 1, \ldots, n_F$;
  Solve the maximum flow problem for F_i for $i = 1, \ldots, n_F$;
  if *not all flows from source are at capacity for graph* $\tilde{\mathcal{G}}_i$ **then**
    Split set F_i into two smaller sets;
    $n_F := n_F + 1$;
  else
    Calculate next hitting time $h(\lambda_2)$;
    Calculate the next violation time $v(\lambda_2)$;
    if $h(\lambda_2) < v(\lambda_2)$ **then**
      Fuse the two sets that hit each other;
      $n_F := n_F - 1$;
    end
  end
  Set $\lambda_2 := \min\{h(\lambda_2), v(\lambda_2)\}$;
end

solving the maximum-flow problem. This gives us the possibility to derive an approximate version of the algorithm that is much faster, as we will see in the simulations section below.

2.5 APPROXIMATE ALGORITHM

The algorithm described above gives an exact solution to the problem. However, for large sets of fused variables, the calculation of a maximal flow is a bottleneck. Computing the maximal flow in a graph can be quite slow, especially for large values of p. In addition to this, if large sets of fused variables split, the resulting sets tend to be very unequal in size, often only splitting off a couple of nodes on the edges. Therefore, a lot of time is spent on cases that do not influence the solution very much. In order to speed up the algorithm, we propose to not check sets of fused variables for splitting up once they are larger than a certain size K. Then, for any set of size K or larger, the maximum-flow problem does not have to be solved, saving time on these especially computationally expensive sets. Also, as the values of τ_{kl} are only used to determine when a set has to be split, these also do not have to be updated any more for the large sets. As we will see in the simulations section, the trade-off in accuracy for moderate values of K is not very large but the algorithm speeds up considerably. The complexity for this approximate algorithm is then $O(nK\log(n))$ in the good case and $O(nK^2\log(n))$ in the bad case (see Online Supplement Section 8.3.3), improving the results especially in the bad-case scenario.

2.6 SIMULATIONS

In order to evaluate the performance of our exact and approximate algorithms above, we want to compare it to that of other methods that have been published before. The first alternative we also use is the component-wise algorithm presented by Friedman et al. (2007). The second is based on the general convex solver CVX, a package for specifying and solving convex problems (see Grant and Boyd 2008a, 2008b). CVX is very easy to use and flexible, which is why we chose it, despite the disadvantage that it cannot be used with a warm start. As we also want to compare the accuracy of the approximate algorithms, we will use these techniques on simulated datasets, which we describe in more detail below.

The comparisons between the algorithms will be performed on datasets of various sizes ranging from 10×10 to 200×200. On each of these datasets, the solution will be computed for 50 equally spaced values of λ_2 between 0 and 0.5, in our experience covering the relevant parameter range—which is from all variables being unfused ($\lambda_2 = 0$) to having all variables in one group. For this speed comparison, there are two things to note.

First, as noted before, CVX cannot use the solution of a similar λ_2 as a "warm start" to speed up computation. So, computing the solution for all 50 values of λ_2 takes roughly 50 times as long as computing the solution for just one value of λ_2. However, we chose to use it nonetheless as it is an easy-to-use general convex solver that can handle sparse matrices and is therefore equipped to handle large datasets.

Second, the algorithm that is presented in this article not only calculates the solution at the 50 values of λ_2, but at all breakpoints of the piecewise-linear solution. The whole path is saved in a compact format and can be used to extract the solution at other values of λ_2 later much faster. For our dataset here, the solution path has at least as many breakpoints as there are datapoints, that is, $n = m^2$ for an $m \times m$ grid. However, we still only let the other algorithm evaluate it for 50 values of λ_2 as we deemed it unrealistic that the solution for possibly thousands of λ_2 values is needed.

2.6.1 The Dataset

In our comparisons, we want to use the two-dimensional FLSA. Therefore, our data will consist of data $\mathbf{y} = \{y_{kl}\}$ with $k = 1, \ldots, m_1$ and $l = 1, \ldots, m_2$ with corresponding coefficients β_{kl}. The difference of coefficients will be penalized if they are neighbors on the two-dimensional grid (horizontal or vertical), that is, the loss function we want to minimize is

$$L(\mathbf{y}, \boldsymbol{\beta}) = \frac{1}{2} \sum_{k=1}^{m_1} \sum_{l=1}^{m_2} (y_{kl} - \beta_{kl})^2 + \sum_{k=1}^{m_1-1} \sum_{l=1}^{m_2} |\beta_{kl} - \beta_{k+1,l}| + \sum_{k=1}^{m_1} \sum_{l=1}^{m_2-1} |\beta_{kl} - \beta_{k,l+1}|,$$

where we have already set $\lambda_1 = 0$. As shown above, we can get the solution for any λ_1 by soft-thresholding the solution for $\lambda_1 = 0$. In our simulated dataset, we set $m_1 = m_2 = m$ for various values of m. The value of y_{kl} is being generated as follows:

1. Set $y_{kl} = 0$ for all $k, l = 1, \ldots, m$.
2. For some rectangles of random size, change the value of y to either 1 or 2, such that roughly 20% have value 1 and 20% have value 2.

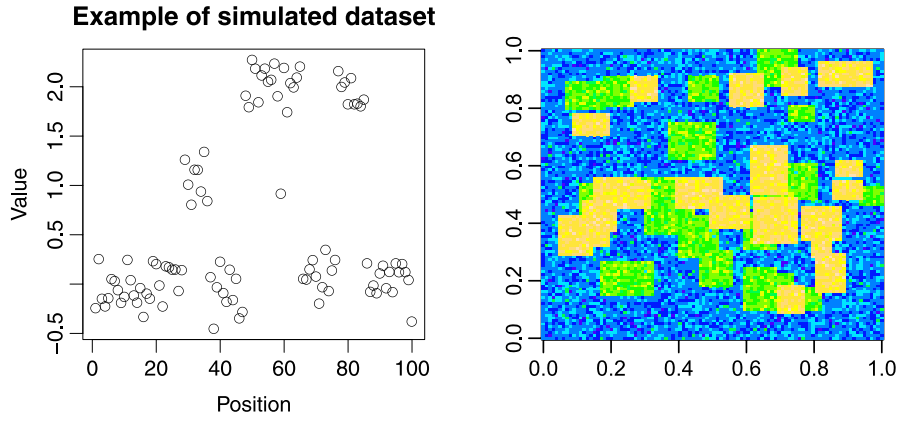

Figure 3. Sample images of a simulated one-dimensional and two-dimensional dataset. The online version of this figure is in color.

3. To every point y_{kl} add standard normal noise with standard deviation of 0.2.

A sample image of what the simulated dataset looks like can be seen in Figure 3.

2.6.2 Results

First, we compare the computation time of the three methods. The results can be seen in Table 1. The path algorithm as well as the component-wise algorithm are both much faster than the general convex solver CVX. When comparing the path algorithm to the component-wise algorithm, we see that they have roughly the same speed for low values of K, except for the small 10×10 dataset.

Table 1. Speed comparison for the two-dimensional FLSA in seconds. The solution is evaluated for 50 values of λ_2 between 0 and 0.5. The results are averaged over 10 runs for the 10×10 and 50×50 datasets and 4 simulation runs for the rest.

	Image size			
	10×10	50×50	100×100	200×200
CVX	36	140	1000	6800
Component-wise alg.	0.062	0.61	2.2	7.7
Path alg.				
$K = 1$	0.0031	0.17	1.1	10
$K = 2$	0.0032	0.18	1.1	10
$K = 5$	0.0040	0.20	1.3	10
$K = 10$	0.0046	0.23	1.4	11
$K = 50$	0.0077	0.37	2.0	14
$K = 100$	–	0.51	3.0	18
$K = 500$	–	1.3	11	62
$K = 1000$	–	1.4	21	120
$K = 2000$	–	1.4	22	290
$K = 5000$	–	–	23	400
exact	0.0079	1.4	24	–

Table 2. Root mean squared deviation (RMSD) accuracy comparison for the two-dimensional FLSA. The accuracy of the approximate version of the path algorithm and the component-wise algorithm is compared to the exact solution using the RMSD. The largest error of the 50 values of λ_2 is reported. The results are averaged over 10 runs for the 10×10 and 50×50 datasets and 4 simulation runs for the rest.

	Image size			
	10×10	50×50	100×100	200×200
CVX	0	0	0	0
Component-wise alg.	0.056	0.066	0.041	0.030
Path alg.				
$K = 1$	0.059	0.067	0.045	0.031
$K = 2$	0.056	0.066	0.044	0.030
$K = 5$	0.041	0.059	0.042	0.029
$K = 10$	0.029	0.053	0.039	0.027
$K = 50$	$<10^{-5}$	0.035	0.027	0.020
$K = 100$	0	0.022	0.022	0.015
$K = 500$	0	0.0050	0.0059	0.0078
$K = 1000$	0	$<10^{-5}$	0.00026	0.0053
$K = 2000$	0	$<10^{-5}$	$<10^{-5}$	0.0028
$K = 5000$	0	0	$<10^{-5}$	0.0019
exact	0	0	0	–

With respect to accuracy, we measure the root mean squared deviation (RMSD). It is calculated for each value of λ_2 and the largest values, averaged over several simulation runs, are displayed in Table 2. In addition to this, we also provide the results of the sup-norm error in Table 4 in Section 8.5 in the Online Supplement. `CVX` returns the exact solution in all cases, although at the cost of a rather slow speed. The component-wise algorithm, on the other hand, is quite fast; however, it only yields an approximate solution, although with a rather small error rate in terms of RMSD. With varying K, the path algorithm is in between the other two methods. However, for values of K around 500–1000, the path algorithm is very accurate but still a lot faster than `CVX`. For most practical application, this time–accuracy trade-off may be worthwhile.

In the speed comparisons of the approximate and exact versions of the algorithms, we have seen the big influence the solution of the maximum-flow problem has on computation times. In the case of a one-dimensional graph $\mathcal{G}$, the maximum-flow problem is not necessary as sets of fused variables cannot break up for increasing values of λ_2. Therefore, in the next section we specialize our results to this simpler case.

3. ONE-DIMENSIONAL FUSED LASSO SIGNAL APPROXIMATOR

The special case of the general FLSA where the graph $\mathcal{G}$ is just a straight line is both of practical relevance and allows for an especially simple algorithm. An example for a possible application was already presented in the Introduction by using the one-dimensional FLSA to analyze CGH data.

The key to the simplification of the algorithm is that for a one-dimensional graph, it is not necessary to check if a set of fused variables has to be split. This is guaranteed by the following theorem:

Theorem 3. *Let $\hat{\beta}_k(\lambda_2)$ be the optimal solution to the one-dimensional FLSA problem for coefficient k and penalty parameter λ_2. Then if for some k and λ_2^0 it holds that $\hat{\beta}_k(\lambda_2^0) = \hat{\beta}_{k+1}(\lambda_2^0)$, then for any $\lambda_2 > \lambda_2^0$ it holds that $\hat{\beta}_k(\lambda_2) = \hat{\beta}_{k+1}(\lambda_2)$.*

A proof of this theorem is provided by Friedman et al. (2007) and an alternative proof based on the techniques for the general FLSA is given in the Online Supplement in Section 9.4.

Therefore, it is not necessary as in the previous section to apply the maximum-flow algorithm in order to check if a group $F_i(\lambda_2)$ needs to split. This considerably simplifies the whole computation and this more efficient version of the procedure is shown as Algorithm 2. Alternatively to starting the path computations at $\lambda_2 = 0$, it is also possible to start at $\lambda_2 = \infty$. This may be desirable if the user is only interested in solutions with a low number of different "levels" for the coefficients. As those solutions will all have relatively large values of λ_2, the computation could be stopped early. This alternative approach is explained in detail in the Supplement in Section 9.5.

Taking this simplification into account, the computational complexity of the one-dimensional FLSA for calculating the whole solution path is then only $n\log(n)$ and a compact version of the solution can be saved with a memory usage of $O(n)$ (see Supplement Section 9).

Algorithm 2: One-dimensional FLSA path algorithm.

initialize
- $\lambda_2 = 0$;
- $\beta_k = y_k$ for $k = 1, \ldots, n$;
- $F_i = \{i\}$ for $i = 1, \ldots, n$;
- $n_F = n$;

end

while $n_F > 1$ **do**
- Calculate next hitting time $h(\lambda_2)$;
- Let $(i_0(\lambda_2), i_0(\lambda_2) + 1) = \arg\min_{h_{i,i+1}(\lambda_2) > \lambda_2} h_{i,i+1}(\lambda_2)$ be the indices of the sets to fuse next;
- Fuse the two sets $F_{i_0(\lambda_2)}$ and $F_{i_0(\lambda_2)+1}$;
- Set $\lambda_2 := h(\lambda_2)$;
- Update the values for $\beta_k(\lambda_2)$, $\frac{\partial \beta_k(\lambda_2)}{\partial \lambda_2}$ and set $n_F = n_F - 1$;

end

Table 3. Time in seconds for a one-dimensional FLSA problem of size n. All three algorithms calculate the solution for 50 equally spaced values of λ_2 from 0 to 1. Results averaged over 10 simulations.

	n					
	10^2	10^3	10^4	10^5	10^6	10^7
CVX	17.1	30.2	210	3600	>5 hours	>5 hours
Component-wise alg.	0.071	0.081	0.24	1.1	10	98
Path alg.	0.0006	0.003	0.030	0.52	7.8	108

3.1 SPEED COMPARISON

As above, we will compare our procedure to the component-wise algorithm presented by Friedman et al. (2007) and CVX (see Grant and Boyd 2008a, 2008b).

As datasets of a wide range of sizes are needed, the speed comparisons will be performed on simulated data again. They consist of datapoints with values of 0, 1, and 2. Roughly 20% of the datapoints will have value 1 and 20% value 2. An example plot of a simulated dataset of size $n = 100$ can be seen in Figure 3.

As in the comparison for the general FLSA, we again calculate the time these approaches take to calculate the solutions of 50 values that are equally spaced here between 0 and 1, chosen to cover the whole relevant parameter range. The results of the comparison can be found in Table 3.

As it can be seen, the path algorithm is consistently faster than the component-wise optimization algorithm for all but the largest problems. They are also both much faster than the general convex solver CVX. In addition to this, the path algorithm also returns an object that stores the complete solution path in a compact form and can be used to extract solutions for additional values of λ_2 very quickly.

4. CONCLUSION

In this article we develop a path algorithm for the Fused Lasso Signal Approximator in its one-dimensional and general form. We compared the speed and accuracy of the FLSA algorithm to other available methods and conclude that our method has advantages in terms of speed and the amount of information gathered and stored. Especially compared to standard convex solvers, our path algorithm is much faster for the FLSA. It is also very easy and quick to extract results for additional penalty parameter values.

For the exact path algorithm, it is necessary to compute the exact breakpoints of the solution paths. In order to do this, we use maximum-flow algorithms in an iterative fashion (i.e., they have to be applied repeatedly), which allows us to determine when we have to split sets that have been previously fused. As, for large graphs, the computation of a maximum flow can be slow compared to the rest of our procedure, we also propose an approximate version of our method that trades off speed for some accuracy. In our simulation, we have shown that for a significant advantage in speed, the loss in accuracy is quite small.

Apart from the work presented here, there are several ways how we plan to expand on it in the future. It is possible to expand the Fused Lasso by allowing each summand in the penalty terms to have separate weights, that is, a loss function of the form

$$L_{\lambda_1,\lambda_2}(\mathbf{y},\mathbf{X},\boldsymbol{\beta}) = \frac{1}{2}(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})^T(\mathbf{y}-\mathbf{X}\boldsymbol{\beta}) + \lambda_1 \sum_{k=1}^{p} w_k|\beta_k| + \lambda_2 \sum_{(k,l)\in E, k<l} w_{kl}|\beta_k - \beta_l|.$$

In order to solve this Fused Lasso problem with a general matrix $\mathbf{X}$ and weights, we propose to adapt the component-wise approach of Friedman et al. (2007) to this setting in order to compute solutions on a grid of λ-values efficiently.

We hope that our algorithms will be used to analyze data and as a building block for other new models. In order to facilitate this we will be publishing implementations of the algorithms in the form of the R package `flsa` on CRAN and the author's website.

SUPPLEMENTARY MATERIALS

Proofs and details: proofs and further details for the methods developed in the article. Supplementary materials are available via single download. (supplement.zip)

[Received November 2009. Revised September 2010.]

REFERENCES

Bertsekas, D. P. (1999), *Nonlinear Programming*, Nashua, NH: Athena Scientific. [991]

Cormen, T. H., Leierson, C. E., Rivest, R. L., and Stein, C. (2001), *Introduction to Algorithms* (2nd ed.), New York: MIT Press/McGraw-Hill. [993]

Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. (2004), "Least Angle Regression" (with discussion), *The Annals of Statistics*, 32 (2), 407–499. [984]

Friedman, J., Hastie, T., Hoefling, H., and Tibshirani, R. (2007), "Pathwise Coordinate Optimization," *The Annals of Applied Statistics*, 2 (1), 302–332. [987,1000,1003-1005]

Grant, M., and Boyd, S. (2008a), "CVX: Matlab Software for Disciplined Convex Programming," available at *http://stanford.edu/~boyd/cvx*. [1000,1004]

——— (2008b), *Graph Implementations for Nonsmooth Convex Programs. Recent Advances in Learning and Control (tribute to M. Vidyasagar)*, London: Springer. [1000,1004]

Hastie, T., Rosset, S., Tibshirani, R., and Zhu, J. (2004), "The Entire Regularization Path for the Support Vector Machine," *The Journal of Machine Learning Research*, 5, 1391–1415. [985]

Jungnickel, D. (2007), *Graphs, Networks and Algorithms* (3rd ed.), Berlin: Springer. [990,993]

Osborne, M. R., Presnell, B., and Turlach, B. A. (2000), "A New Approach to Variable Selection in Least Squares Problems," *IMA Journal of Numerical Analysis*, 20, 389–404. [984]

Park, M.-Y., and Hastie, T. (2007), "An L1 Regularization-Path Algorithm for Generalized Linear Models," *Journal of the Royal Statistical Society, Ser. B*, 69, 659–677. [985]

Pollak, I., Willsky, A. S., and Huang, Y. (2005), "Nonlinear Evolution Equations as Fast and Exact Solvers of Estimation Problems," *IEEE Transactions on Signal Processing*, 53, 484–498. [986]

Rosset, S., and Zhu, J. (2007), "Piecewise Linear Regularized Solution Paths," *The Annals of Statistics*, 35 (3), 1012–1030. [985]

Sleator, D. D., and Tarjan, R. E. (1983), "A Data Structure for Dynamic Trees," *Journal of Journal of Computer and System Sciences*, 24, 362–391. [998]

Tibshirani, R. (1996), "Regression Shrinkage and Selection via the Lasso," *Journal of the Royal Statistical Society, Ser. B*, 58, 267–288. [984]

Tibshirani, R., Saunders, M., Zhu, J., and Rosset, S. (2005), "Sparsity and Smoothness via the Fused Lasso," *Journal of the Royal Statistical Society, Ser. B*, 67, 91–108. [985]