Fast Proximal Gradient Methods with Node Pruning for Tree-Structured Sparse Regularization

Yasutoshi Ida[1][0000−0003−4279−9503] (✉), Sekitoshi Kanai[2], Atsutoshi Kumagai[3], Tomoharu Iwata[4], and Yasuhiro Fujiwara[5]

[1] NTT Computer and Data Science Laboratories yasutoshi.ida@ieee.org
[2] NTT Computer and Data Science Laboratories sekitoshi.kanai@ntt.com
[3] NTT Computer and Data Science Laboratories atsutoshi.kumagai@ntt.com
[4] NTT Communication Science Laboratories tomoharu.iwata@ntt.com
[5] NTT Communication Science Laboratories yasuhiro.fujwara@ntt.com

Abstract. Sparse learning with structural information is a fundamental framework for feature selection. Among the various structures, the tree is a basic one that appears in feature vectors, and tree-structured regularization has been utilized to incorporate trees into objective functions. Although proximal gradient methods (PGMs) are usually used for optimization, they incur high computation costs for deep tree structures or large datasets. We propose a fast PGM for tree-structured regularization. Our method safely skips parameter updates of PGMs for pruning unnecessary leaf nodes in the tree. In addition, it prunes unnecessary computations for internal nodes in a hierarchical manner. Our method guarantees the same optimization results and convergence rate as the original method. Furthermore, it can be applied to various PGMs for tree-structured regularization. Experiments show that our method reduces the processing time by up to 56% from the original method without degrading accuracy.

Keywords: Sparse learning · Feature selection · Pruning method.

1 Introduction

Feature selection using sparsity-inducing regularization, as in Lasso [27, 7], is a fundamental technique of data analysis. In many applications, as shown in Figure 1, a tree structure naturally appears in the feature vector wherein the features and groups of features correspond to leaf nodes and internal nodes, respectively [21, 14, 22]. Tree-structured regularization [21, 16] effectively handles such structural information by solving a regularized optimization problem. It is based on a linear regression model whose parameter vector has the same tree structure as that of the feature vector. By inducing sparsity at each node of the tree, it selects important features from all the features. Owing to its effectiveness, tree-structured regularization has been used for many applications, including multi-task learning where multiple tasks are related through a tree structure [19],

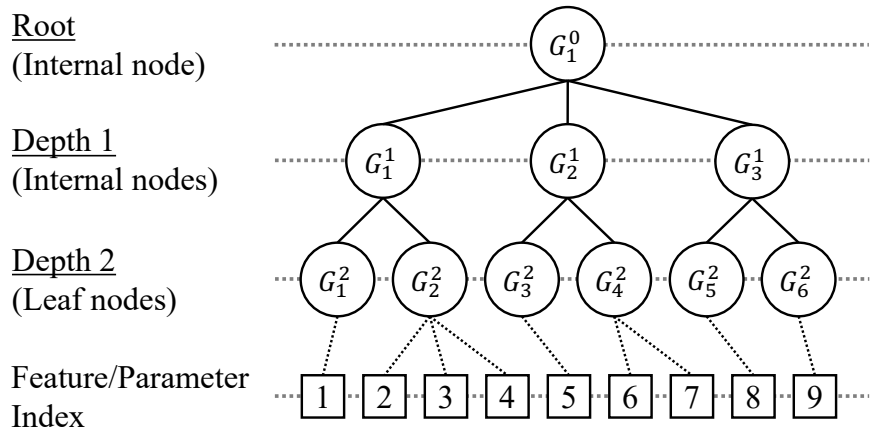


Fig. 1. Example tree structure of features.

multi-scale mining of fMRI data where each parent node contains a series of child nodes that enjoy spatial locality [14], and traffic-sign recognition where the sign categories are tree-structured [22].

Although solving the optimization problem seems difficult because of its tree-structured regularization, [21] and [16] derived a proximal operator in a closed form and applied a proximal gradient method (PGM), called the iterative shrinkage thresholding algorithm (ISTA), to the problem. Specifically, the PGM updates the parameters and applies the proximal operator in each iteration of a gradient method. The proximal operator applies a soft-thresholding function to parameters corresponding to each node of the tree in bottom-up breadth-first order. Intuitively, this function shrinks the parameters to zero if the ℓ_2 norm of the passed parameters is less than or equal to a threshold. As a result, the entire parameter vector becomes sparse and its nonzero parameters correspond to important features.

From the perspective of computation cost, the proximal operator for tree-structured regularization requires high computation cost when the number of features p or the depth of the tree d is large. Specifically, it is $\mathcal{O}(pd)$ time for each iteration of the PGM [21]. This is because it repeatedly applies a soft-thresholding function to each node in the tree. In addition to the proximal operator, we have to take the gradient with respect to each of the parameters before the proximal operator is applied. Since the gradient computation requires $\mathcal{O}(p^2)$ or $\mathcal{O}(np)$ time, where n is the number of data points, it also incurs high computation costs on large datasets.

This paper proposes a fast proximal gradient descent for tree-structured regularization. The method is based on two ideas as follows:

Pruning leaf nodes to reduce the cost of gradient computations. The first idea is to prune unnecessary parameter updates, including gradient computations which require $\mathcal{O}(p^2)$ or $\mathcal{O}(np)$ time. Specifically, we prune updates for parameters that turn to be zeros during optimization. Since a leaf node in the tree corresponds to a set of parameters as shown in Figure 1, this idea corresponds to pruning unnecessary computations for leaf nodes. We utilize an upper bound of the ℓ_2 norm of the parameters passed to the soft-thresholding function for pruning. Specifically, if this upper bound is less than or equal to the threshold, we prune the computation of the leaf node.

Pruning internal nodes to reduce the cost of the proximal operator. The second idea is to prune unnecessary computations for internal nodes to reduce the computation cost of the proximal operator which requires $\mathcal{O}(pd)$ time. The key is that an upper bound of a parent node can be computed by summing the upper bounds of its child nodes. It enables us to efficiently compute upper bounds for the internal nodes and prune unnecessary computations in the same way as the first idea. Because our upper bound requires $\mathcal{O}(1)$ time for one node, our method requires only $\mathcal{O}(p)$ time for computing the upper bounds of the entire tree.

Since our method only changes the computation of the proximal operator, it can be used with various PGMs, such as ISTA, fast ISTA (FISTA) [3], optimized ISTA (OISTA) [18], and modified FISTA (FISTA-Mod) [20]. In addition, it guarantees the same optimization results and convergence rate as the original methods because we can safely prune unnecessary computations for zero parameters. Experiments show that our method reduces the processing time by up to 56% from the original PGM while achieving the same objective values.

2 Preliminary

2.1 Tree-structured Regularization

Let $X \in \mathbb{R}^{n \times p}$ be a matrix of features, where n is the number of data points and each data point is represented by a p-dimensional feature vector. $y \in \mathbb{R}^n$ is a set of continuous responses. We consider the following optimization problem with sparsity-inducing regularization:

$$\min_{\beta \in \mathbb{R}^p} \frac{1}{2}\|y - X\beta\|_2^2 + \lambda\phi(\beta), \tag{1}$$

where $\|\cdot\|_2$ is the ℓ_2 norm, $\beta \in \mathbb{R}^p$ is the parameter vector, $\lambda > 0$ is the regularization constant, and $\phi(\beta)$ is the sparsity-inducing regularization term for β. Here, we will let $f(\beta) = \frac{1}{2}\|y - X\beta\|_2^2$ for simplicity. Owing to $\phi(\beta)$, β will be sparse after optimization and we can select important features for predicting y from among all the features. To utilize the tree structure of the features, [21] incorporate structural information into $\phi(\beta)$. The structure is defined using the following index tree:

Algorithm 1 FISTA

1: **Input:** A Lipschitz constant L of ∇f
2: **Initialization:** $\beta_0 \in \mathbb{R}^p$, $\hat{\beta}_0 = \beta_0$, $s_0 = 1$, $t = 0$, $\eta = 1/L$
3: **repeat**
4: $\quad \beta_{t+1} = \mathrm{prox}^{\phi}_{\lambda\eta}(\hat{\beta}_t - \eta\nabla f(\hat{\beta}_t))$
5: $\quad s_{t+1} = \frac{1+\sqrt{1+4s_t^2}}{2}$
6: $\quad \hat{\beta}_{t+1} = \beta_{t+1} + (\frac{s_t - 1}{s_{t+1}})(\beta_{t+1} - \beta_t)$
7: $\quad t = t + 1$
8: **until** convergence

Definition 1 (Index Tree [21]). *For an index tree $\mathbb{T}$ of depth d, let $\mathbb{T}_i = \{G_1^i, G_2^i, ..., G_{n_i}^i\}$ contain all the nodes corresponding to depth i, where $n_0 = 1$, $G_1^0 = \{1, 2, ..., p\}$ and $n_i \geq 1$, $i = 1, 2, ..., d$. Each node G_j^i corresponds to a subset of $\{1, 2, ..., p\}$. The nodes satisfy the following conditions: 1) the nodes from the same depth level have non-overlapping indices, i.e., $G_j^i \cap G_k^i = \emptyset$, $\forall i = 1, ..., d$, $j \neq k$, $1 \leq j, k \leq n_i$; and 2) $G_j^i \subseteq G_{j0}^{i-1}$, where G_{j0}^{i-1} is the parent node of a non-root node G_j^i.*

Figure 1 shows an example of the index tree. Tree-structured regularization is defined using the index tree as follows:

$$\phi(\beta) = \sum_{i=0}^{d} \sum_{j=1}^{n_i} w_j^i \|\beta[G_j^i]\|_2, \tag{2}$$

where $w_j^i \geq 0$ ($i = 0, 1, ..., d$, $j = 1, 2, ..., n_i$) is a pre-defined weight for node G_j^i, and $\beta[G_j^i]$ is a vector composed of the entries of β whose indices are in G_j^i. For instance, if $G_j^i = \{1, 3, 5\}$, $\beta[G_j^i]$ consists of the first, third and fifth elements of β. Intuitively, $\beta[G_j^i]$ tends to have zeros after optimization because $\phi(\beta)$ consists of the l_2 norms of the parameters, the same as group Lasso [30, 10].

2.2 Proximal Gradient Method

Problem (1) can be solved by using PGMs such as ISTA and FISTA [3, 21]. As a representative example, we describe FISTA, the pseudocode of which is in Algorithm 1. FISTA consists of a parameter update (line 4) and a linear combination of parameters (lines 5–6). Note that ISTA, FISTA, OISTA, and FISTA-Mod share part of this parameter update (line 4). The parameter vector $\hat{\beta}_t$ is updated as $\hat{\beta}_t - \eta\nabla f(\hat{\beta}_t)$, where the gradient $\nabla f(\hat{\beta}_t)$ is computed as follows:

$$\nabla f(\hat{\beta}_t) = X^\top X\hat{\beta}_t - X^\top y. \tag{3}$$

Then, the updated parameter vector is passed to the proximal operator, $\mathrm{prox}^{\phi}_{\lambda\eta}(\cdot)$, which yields β_{t+1} (line 4). We will describe the specific procedure of the proximal operator later. Lines 5–6 compute a linear combination of β_{t+1} and β_t on the basis of Nesterov's acceleration method [24].

Algorithm 2 $\mathrm{prox}^{\phi}_{\lambda\eta}(v)$

1: **Initialization:** $u^{d+1} = v$
2: **for** $i = d, ..., 0$ **do**
3: **for** $j = 1, ..., n_i$ **do**
4: $\lambda^i_j = \lambda w^i_j \eta$
5: **if** $\|u^{i+1}[G^i_j]\|_2 \leq \lambda^i_j$ **then**
6: $u^i[G^i_j] = \mathbf{0}$
7: **else**
8: $u^i[G^i_j] = \frac{\|u^{i+1}[G^i_j]\|_2 - \lambda^i_j}{\|u^{i+1}[G^i_j]\|_2} u^{i+1}[G^i_j]$
9: **return** u^0

The proximal operator on line 4 is defined as

$$\mathrm{prox}^{\phi}_{\lambda\eta}(v) = \underset{u \in \mathbb{R}^p}{\mathrm{argmin}}\, \lambda\phi(u) + \tfrac{1}{2\eta}\|v - u\|_2^2, \tag{4}$$

where $v \in \mathbb{R}^p$. Although $\phi(\cdot)$ has the tree structure shown in Equation (2), [21] found an analytical solution for Equation (4) via the Moreau-Yosida regularization (Moreau envelope) [29, 25] of $\phi(\cdot)$. The procedure is described in Algorithm 2. First, u^{d+1}, which is the target of the proximal operator, is initialized (line 1). Next, u^{d+1} is updated by traversing the index tree T in bottom-up breadth-first order (lines 2–8). Then, λ^i_j is computed as a threshold of the soft-thresholding function (line 4). After that, $u^i[G^i_j]$ is updated at each traversed node G^i_j (lines 5–8). If $\|u^{i+1}[G^i_j]\|_2 \leq \lambda^i_j$, $u^i[G^i_j]$ is updated to a zero vector (lines 5–6). If not, $u^i[G^i_j]$ is shrunk by multiplying $(\|u^{i+1}[G^i_j]\|_2 - \lambda^i_j)/\|u^{i+1}[G^i_j]\|_2$ (lines 7–8). This procedure can be regarded as being the same as the soft-thresholding function of group Lasso [30]. As a result, the parameter vector is expected to be sparse during optimization.

Although we can compute the gradient and the proximal operator by Equations (3) and (4), the computation cost corresponding to line 4 of Algorithm 1 clearly dominates the other costs. Specifically, computing the proximal operator requires $\mathcal{O}(pd)$ time for Algorithm 2 in every iteration of Algorithm 1 [21]. If the tree is a perfect binary tree, it requires $\mathcal{O}(p \log p)$ time because of $d = \mathcal{O}(\log p)$. In addition, line 4 of Algorithm 1 computes $\hat{\beta}_t - \eta\nabla f(\hat{\beta}_t)$ including the gradient computation before Algorithm 2 is called. The gradient computation requires $\mathcal{O}(p^2)$ or $\mathcal{O}(np)$ time in every iteration because of Equation (3). As a result, executing line 4 of Algorithm 1 takes $\mathcal{O}(p^2 + pd)$ or $\mathcal{O}(pn + pd)$ time. Namely, the cost of Algorithm 1 is high when the depth of the tree or the dataset is large.

3 Proposed Approach

This section describes ideas of the proposed method. For the sake of simplicity, the discussion of the algorithm and the time complexity will assume the tree structure to be a perfect binary tree, although our method is applicable to any

tree structure that satisfies Definition 1. The omitted proofs can be found in the Appendix.

3.1 Overview of the Ideas

The bottleneck of the PGMs is line 4 of Algorithm 1. It can be decomposed into two problems wherein 1) the existing method computes $\hat{\beta}_t - \eta\nabla f(\hat{\beta}_t)$ at $\mathcal{O}(p^2)$ or $\mathcal{O}(np)$ time and 2) the proximal operator requires $\mathcal{O}(p \log p)$ time. These computation costs are incurred at every iteration until convergence. We tackle these two problems with the two ideas.

The first idea is to prune unnecessary computations of $\hat{\beta}_t - \eta\nabla f(\hat{\beta}_t)$ corresponding to parameters that turn out to be zero. Specifically, our method checks whether each parameter is zero or not before computing $\hat{\beta}_t - \eta\nabla f(\hat{\beta}_t)$. If a parameter is determined to be zero, we prune the corresponding computation of $\hat{\beta}_t - \eta\nabla f(\hat{\beta}_t)$. To identify parameters that turn out to be zero, we introduce an upper bound of $\|u^{d+1}[G_j^d]\|_2$ at line 5 of Algorithm 2. If this upper bound is less than or equal to λ_j^i, the corresponding parameter is zero. The upper bound can be efficiently computed because it only requires $\mathcal{O}(p)$ time for all the parameters. This pruning corresponds to pruning computations for leaf nodes because $u^{d+1} = v = \hat{\beta}_t - \eta\nabla f(\hat{\beta}_t)$ holds, from line 1 in Algorithm 2. We describe the details of this procedure in Section 3.2.

The second idea is to prune internal nodes in the tree on the basis of upper bounds. Although this idea appears to be similar to the first one, the computation of the upper bounds is different. Specifically, the upper bounds of internal nodes are computed by summing the upper bounds of their child nodes, as we will describe in Section 3.3. This method allows us to efficiently compute upper bounds for all the internal nodes in bottom-up breadth-first order at $\mathcal{O}(p)$ time. As a result, although original methods require $\mathcal{O}(p \log p)$ time to compute the proximal operator, we can efficiently prune unnecessary computations of the proximal operator by using the upper bounds.

3.2 Pruning Leaf Nodes

As described in Section 3.1, pruning unnecessary computations of $\hat{\beta}_t - \eta\nabla f(\hat{\beta}_t)$ corresponds to pruning leaf nodes because $u^{d+1} = v = \hat{\beta}_t - \eta\nabla f(\hat{\beta}_t)$ holds from line 1 in Algorithm 2. Specifically, u_t^{d+1}, which is u^{d+1} at the t-th iteration of Algorithm 1, is computed from Equation (3) as follows:

$$u_t^{d+1} = \hat{\beta}_t - \eta(X^\top X\hat{\beta}_t - X^\top y) = M\hat{\beta}_t + \eta X^\top y, \tag{5}$$

where

$$M = I - \eta X^\top X. \tag{6}$$

Then, to prune unnecessary computations for leaf nodes, we introduce $\overline{u}_t^{d+1}[G_j^d]$ as follows:

Definition 2. *Let t' be $0 \leq t' < t$ in Algorithm 1; $\overline{u}_t^{d+1}[G_j^d]$ is computed as*

$$\overline{u}_t^{d+1}[G_j^d] = \|u_{t'}^{d+1}[G_j^d]\|_2 + \|M[G_j^d]\|_F \|\Delta\hat{\beta}_t\|_2, \tag{7}$$

where $\|M[G_j^d]\|_F$ represents the Frobenius norm of the submatrix of M containing only G_j^d rows and $\Delta\hat{\beta}_t = \hat{\beta}_t - \hat{\beta}_{t'}$.

$\|M[G_j^d]\|_F$ is computed only once before the optimization because it does not depend on any parameters. $\|u_{t'}^{d+1}[G_j^d]\|_2$ is computed at regular iteration intervals in the PGM, as described in Section 3.4. The following lemma shows that $\overline{u}_t^{d+1}[G_j^d]$ is an upper bound of $\|u_t^{d+1}[G_j^d]\|_2$:

Lemma 1 (Upper Bound for Leaf Nodes). *We have $\overline{u}_t^{d+1}[G_j^d] \geq \|u_t^{d+1}[G_j^d]\|_2$ when $\overline{u}_t^{d+1}[G_j^d]$ is computed by using Equation (7).*

The following lemma shows that the upper bound enables us to prune unnecessary leaf nodes:

Lemma 2. *If $\overline{u}_t^{d+1}[G_j^d] \leq \lambda_j^d$ holds, we have $u_t^d[G_j^d] = \mathbf{0}$.*

From Equation (7), the total computation cost of the upper bounds for all the leaf nodes is as follows:

Lemma 3. *The total computation cost of Equation (7) for the leaf nodes $j \in \{1, ..., n_d\}$ is $\mathcal{O}(p)$ time given precomputed $u_{t'}^{d+1}[G_j^d]$ and $\|M[G_j^d]\|_F$ for $j \in \{1, ..., n_d\}$.*

According to Lemmas 2 and 3, we can identify the indices G_j^d of the unnecessary leaf nodes in $\mathcal{O}(p)$ time by using the upper bound $\overline{u}_t^{d+1}[G_j^d]$ instead of $\|u_t^{d+1}[G_j^d]\|_2$ at line 5 in Algorithm 2. Since the total computation cost of $\|u_t^{d+1}[G_j^d]\|_2$ for all the leaf nodes is $\mathcal{O}(p^2)$ or $\mathcal{O}(np)$ time as a result of computing $u_t^{d+1} = \hat{\beta}_t - \eta\nabla f(\hat{\beta}_t)$, our upper bounds efficiently identify unnecessary leaf nodes.

We should note that $\overline{u}_t^{d+1}[G_j^d]$ is an approximation of $\|u_t^{d+1}[G_j^d]\|_2$. We have the following error bound for this approximation:

Lemma 4. *Let ϵ be $2\|M[G_j^d]\|_F \|\Delta\hat{\beta}_t\|_2$. Then, it satisfies $|\overline{u}_t^{d+1}[G_j^d] - \|u_t^{d+1}[G_j^d]\|_2| \leq \epsilon$.*

This section introduced the upper bound for leaf nodes that are the cases of $i = d$ in Algorithm 2. In the gradient computation, we can use the above upper bound for the leaf nodes. However, the bound cannot be used for the internal nodes. In the next section therefore, we derive another upper bound for internal nodes that are the cases of $i \neq d$.

3.3 Pruning Internal Nodes

First, let us introduce $\overline{u}_t^{i+1}[G_j^i]$, where $i \neq d$:

Algorithm 3 Fast Proximal Operator for Tree

1: **for** $i = d, ..., 0$ **do**
2: **for** $j = 1, ..., n_i$ **do**
3: $\lambda_j^i = \lambda w_j^i \eta$
4: **if** $i = d$ **then**
5: compute $\overline{u}_t^{i+1}[G_j^i]$ by Eqn. (7)
6: **else**
7: compute $\overline{u}_t^{i+1}[G_j^i]$ by Eqn. (8)
8: **if** $\overline{u}_t^{i+1}[G_j^i] \leq \lambda_j^i$ **then**
9: $u^i[G_j^i] = \mathbf{0}$
10: **else**
11: **if** $i = d$ **then**
12: compute $u_t^{i+1}[G_j^i]$ by Eqn. (5)
13: $u^i[G_j^i] = \max\Big(0, \frac{\|u^{i+1}[G_j^i]\|_2 - \lambda_j^i}{\|u^{i+1}[G_j^i]\|_2}\Big) u^{i+1}[G_j^i]$
14: **return** u^0

Definition 3. *Suppose that $\overline{u}_t^{d+1}[G_j^d]$ is computed by Equation (7) for an index tree T. Then, $\overline{u}_t^{i+1}[G_j^i]$, where $i = d-1, ..., 0$, is computed in the bottom-up breadth-first order as follows:*

$$\overline{u}_t^{i+1}[G_j^i] = \textstyle\sum_{k \in \mathbb{D}_j^i} \max(0, \overline{u}_t^{i+2}[G_k^{i+1}] - \lambda_k^{i+1}), \tag{8}$$

where $\mathbb{D}_j^i$ is a set of indices k such that $G_k^{i+1} \subseteq G_j^i$.

Equation (8) indicates that $\overline{u}_t^{i+1}[G_j^i]$ of node G_j^i is computed by using the upper bounds of its child nodes $G_k^{i+1} \subseteq G_j^i$. $\overline{u}_t^{i+1}[G_j^i]$ has the following property:

Lemma 5 (Upper Bound for Internal Nodes). *For $i = d-1, ..., 0$, we have $\overline{u}_t^{i+1}[G_j^i] \geq \|u_t^{i+1}[G_j^i]\|_2$ when $\overline{u}_t^{i+1}[G_j^i]$ is computed using Equation (8).*

By using the upper bound, we can identify unnecessary internal nodes as follows:

Lemma 6. *If $\overline{u}_t^{i+1}[G_j^i] \leq \lambda_j^i$ holds for $i \in \{d-1, ..., 0\}$, we have $u_t^i[G_j^i] = \mathbf{0}$.*

The cost of computing the upper bounds of all the internal nodes is as follows:

Lemma 7. *The computation cost of Equation (8) for all the internal nodes is $\mathcal{O}(p)$ time.*

3.4 Algorithm

Algorithm 3 presents the pseudocode of our method for computing the gradient and the proximal operator based on the discussion in Sections 3.2 and 3.3. It computes upper bounds (lines 4–7) and decides whether to prune the nodes (lines 8–13). The computation is performed in bottom-up breadth-first order (lines 1–2). In particular, it computes the upper bound by using Equation (7) (lines 4–5) when the node is a leaf node and Equation (8) (lines 6–7) when the

Algorithm 4 Pruned FISTA with Algorithm 3

1: **Input:** A Lipschitz constant L of ∇f, $r > 1$
2: **Initialization:** $\beta_0 = \hat{\beta}_0$, $s_0 = 1$, $t = 0$, $\eta = 1/L$
3: **for** $j = 1, ..., n_d$ **do**
4: compute $\|M[G_j^d]\|_F$
5: **repeat**
6: $\hat{\beta}_{t'} = \hat{\beta}_t$
7: **for** $j = 1, ..., n_d$ **do**
8: compute $\|u_{t'}^{d+1}[G_j^d]\|_2$ by Eqn. (5)
9: **for** $m = 1, ..., r$ **do**
10: compute β_{t+1} by Algorithm 3
11: $s_{t+1} = \frac{1+\sqrt{1+4s_t^2}}{2}$
12: $\hat{\beta}_{t+1} = \beta_{t+1} + (\frac{s_t - 1}{s_{t+1}})(\beta_{t+1} - \beta_t)$
13: $t = t + 1$
14: **until** convergence

node is an internal node. If the upper bound is less than or equal to the threshold $\lambda_j^i = \lambda w_j^i \eta$, the node is pruned (lines 8–9). If not, the exact value is computed (lines 10–13). In this case, if the node is a leaf node, the value is computed with Equation (5) only for the dimension corresponding to G_j^i. (lines 11–12).

Algorithm 4 is the pseudocode of FISTA with Algorithm 3. First, it precomputes $\|M[G_j^d]\|_F$, which is used for computing the upper bounds (lines 3–4). The main loop of the algorithm is lines 5–14. To compute the upper bounds for the leaf nodes (Equation (7)), the algorithm computes $\hat{\beta}_{t'}$ and $\|u_{t'}^{d+1}[G_j^d]\|_2$ (lines 6–8). It computes these values at regular iteration intervals $r > 1$ (line 9). Note that, $r = 2$ in this study. The performance variation with r is shown in Section 5.1. Lines 10–13 are the same procedure as in the original FISTA (Algorithm 1) except for the computation of the proximal operator. Specifically, the proximal operator is computed using Algorithm 3.

Although Algorithm 4 is based on FISTA, our method can be also used with other PGMs such as ISTA, OISTA [18], and FISTA-Mod [20] as shown in the Appendix. This is because our approach only changes the computation of the proximal operator in these methods.

3.5 Theoretical Analysis

Algorithm 4 has the following property:

Theorem 1 (Optimization Result). *Suppose that Algorithm 4 has the same hyperparameters and initial parameter vector as those of Algorithm 1. Then, Algorithm 4 converges to the same objective value and solution as Algorithm 1.*

This theorem suggests that our method returns the same results as the original method. This property also holds for ISTA, OISTA, and FISTA-Mod when they use our method. From Theorem 1 and its proof, our method has the following property regarding the convergence rate:

Corollary 1. *Algorithm 4 achieves the same convergence rate as Algorithm 1.*

Since the computation results of Algorithm 4 for the gradient and the proximal operator are the same as those of Algorithm 1 in every iteration, the proof is the same as in [3] and the above corollary clearly holds. Since Algorithm 4 is based on FISTA (Algorithm 1), it runs in $\mathcal{O}(1/t^2)$.

The computation cost of Algorithm 4 is as follows:

Theorem 2 (Computation Cost). *Let n_u be the total number of dimensions of vectors corresponding to un-pruned internal nodes and let n_l be that for un-pruned leaf nodes. If T is the total number of loops of lines 9–13, the computation cost of Algorithm 4 is $\mathcal{O}(p^2(n + \frac{T}{r}) + p(n + T + n_l) + n_u)$ time.*

This theorem suggests that if n_u and n_l are small, the processing time will be reduced by using Algorithm 4. Since the cost of the original FISTA with the tree-structured regularization (Algorithm 1) is $\mathcal{O}(p^2(n + T) + p(n + Td))$ time, our method also reduces the coefficient of p^2 in its cost.

The error bound of the upper bound for Algorithm 4 has the following property:

Theorem 3 (Convergence of Error Bound). *We have $\epsilon = 0$ when $\hat{\beta}_t$ in Algorithm 4 converges.*

The above theorem suggests that $\overline{u}_t^{d+1}[G_j^d]$ matches $\|u_t^{d+1}[G_j^d]\|_2$ if $\hat{\beta}_t$ converges. Namely, the upper bounds can accurately identify unnecessary leaf nodes when $\hat{\beta}_t$ converges.

4 Related Work

Several algorithms have been proposed to solve optimization problems with tree-structured regularizers [16]. [31, 32] used a boosting-like technique based on BLasso [33]. Since it utilizes a path-following strategy, it can obtain regularization paths although it has difficulty computing them in parallel for each regularization constant. [19] used a reweighted least-squares scheme based on a variational formulation [1]. However, it cannot yield sparse parameters [16].

PGMs for tree-structured regularizers overcome the above drawbacks [15, 21]. Together with the PGM, an acceleration technique with momentum can be utilized to reduce the processing time. A popular one is FISTA [3], which leverages Nesterov's acceleration method [24]. It decreases the function value at the rate $\mathcal{O}(1/t^2)$. FISTA is the most widely used accelerated PGM, and numerous improvements to it have been proposed. A typical example is Monotone FISTA (MFISTA), which does not increase the function value during optimization by introducing an extra computation of the objective function in each iteration [2, 34]. Another example is OISTA, which improves on the convergence of FISTA on the basis of a performance estimation problem (PEP) [5, 18, 26, 8]. FISTA-Mod incorporates hyperparameters to control s_{t+1}, which leads to enhanced convergence speed and improved stability [20]. [4] proposed smoothing proximal gradient (SPG) to deal with structured sparsity-inducing regularization. Although it

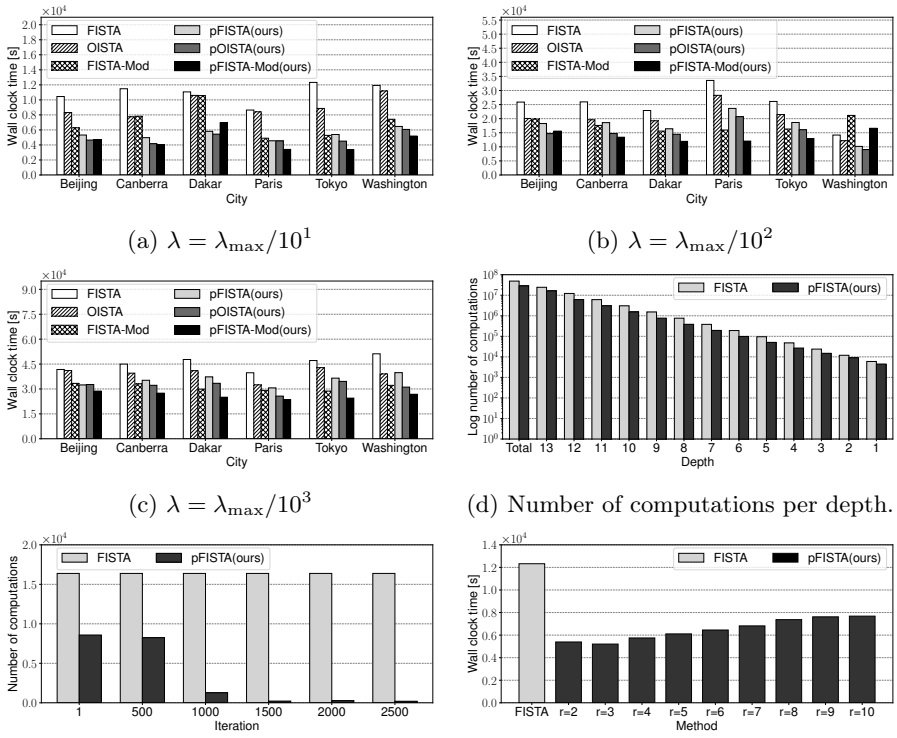
can be also used with FISTA, the dimension of the gradient vector increases if there are a lot of overlapping groups such as the tree structure of Definition 1. Specifically, the space complexity is $\mathcal{O}(pd)$ space while the approach in this paper requires $\mathcal{O}(p)$ space [15, 21].

5 Experiment

We evaluated the processing times and the objective values of our method. FISTA [3], OISTA [18], and FISTA-Mod [20] were chosen as baselines for comparison. Note that ISTA was removed from the comparison because it had not finished executing within a month. Since our approach is applicable to FISTA, OISTA, and FISTA-Mod, we also evaluated their combinations, i.e., pruned FISTA (pFISTA), pruned OISTA (pOISTA), and pruned FISTA-Mod (pFISTA-Mod). We tried $\lambda w_j^i = \{\lambda_{\max}/10, \lambda_{\max}/10^2, \lambda_{\max}/10^3\}$ after consulting previous papers [6, 12]. $\lambda_{\max}$ is the smallest regularization constant for which all the parameters are zero at the optimal solutions [28]. We stopped each algorithm when the relative tolerance of the parameter vector dropped below 10^{-5} [9, 11, 13]. For additional hyperpameters of FISTA-Mod, we used the same values as those in the experiment of the original paper [20]. We used the climate dataset from NCEP/NCAR Reanalysis 1 [17, 23]. It contains the means of climate data measurements spread across the globe in a grid of $2.5^\circ \times 2.5^\circ$ resolution for each month from 1948/1/1 to 2022/6/1. Each grid point has a group of seven variables: air temperature, precipitable water, relative humidity, pressure, sea level pressure, horizontal wind speed, and vertical wind speed. The size of the data matrix was 894×57344. This dataset has spatial locality in the features of adjacent grid points and a hierarchical structure of areas due to the spatial data. To exploit this property, we constructed a perfect binary tree with adjacent coordinates in grids. Since we could handle the seven variables as a group, each leaf node corresponded to a group and the depth of the tree was 13. As targeted responses, we used air temperatures in neighborhoods of Beijing, Canberra, Dakar, Paris, Tokyo, and Washington. Therefore, our experiments consisted of six regression tasks. All the experiments were conducted with one CPU core and 264 GB of main memory on a 2.20 GHz Intel Xeon server running Linux.

5.1 Processing Time

Figures 2 (a)–(c) show the wall clock times of the six regression tasks for each hyperparameter. Our methods, pFISTA, pOISTA, and pFISTA-Mod, were faster than FISTA, OISTA, and FISTA-Mod, respectively. The efficiency of our methods increased as we set larger hyperparameters. Specifically, since the hyperparameter setting in Figure 2 (a) strengthens the regularization, many parameters turned out to be zero. In this case, our method pruned unnecessary computations relatively easily; it reduced the processing time by up to 56% form the existing methods. For all cases, our methods uniformly accelerated FISTA, OISTA, and FISTA-Mod, demonstrating the generality of method.



(a) $\lambda = \lambda_{\max}/10^1$

(b) $\lambda = \lambda_{\max}/10^2$

(c) $\lambda = \lambda_{\max}/10^3$

(d) Number of computations per depth.

(e) Number of computations per iteration.

(f) Processing times for each hyperparameter r.

Fig. 2. (a)–(c): Processing times for each hyperparameter λ. (d)–(e): Numbers of computations at nodes per depth and iteration. (f): Processing times for each hyperparameter r.

Number of Computations at Nodes per Depth. The main idea of our method is to prune unnecessary computations at nodes in the tree. Therefore, we compared the numbers of computations at nodes of the original method with those of our method. Figure 2 (d) shows the number of computations per depth and their total. As a representative example, we compared FISTA and pFISTA (ours) for predicting the air temperature in Tokyo when $\lambda w_j^i = \lambda_{\max}/10$. Note that the trend was almost the same for other cities and methods. Our method reduced the number of the total computations to 58.35% compared with the original method. Specifically, regarding the first idea of pruning leaf nodes, the number was reduced to 66.37% as shown at depth 13 in the figure. Regarding the second idea of pruning internal nodes, the number was reduced to 50.32% from the sum of computations for depths 1 to 12. The result reveals that our ideas effectively pruned unnecessary computations.

Number of Computations at Nodes per Iteration. We also investigated the number of computations at nodes for each iteration under the experimental

Table 1. Comparison of objective values for FISTA and pFISTA (ours).

City	λ	FISTA	pFISTA(ours)
Canberra	$\lambda_{\max}/10^1$	1311×10^{-1}	1311×10^{-1}
	$\lambda_{\max}/10^2$	9824×10^{-3}	9824×10^{-3}
	$\lambda_{\max}/10^3$	1494×10^{-3}	1494×10^{-3}
Paris	$\lambda_{\max}/10^1$	9934×10^{-2}	9934×10^{-2}
	$\lambda_{\max}/10^2$	5389×10^{-3}	5389×10^{-3}
	$\lambda_{\max}/10^3$	9595×10^{-4}	9595×10^{-4}
Washington	$\lambda_{\max}/10^1$	1301×10^{-1}	1301×10^{-1}
	$\lambda_{\max}/10^2$	6620×10^{-3}	6620×10^{-3}
	$\lambda_{\max}/10^3$	1355×10^{-3}	1355×10^{-3}

settings described above. Specifically, we counted the numbers at the 1st, 500-th, 1000-th, 1500-th, 2000-th, 2500-th iteration. The algorithms were stopped at the 2984-th iteration. Figure 2 (e) shows the results, which suggest that our method pruned half of the computations even at the beginning of optimization. Our method further reduced the number of computations as it converged. This is because $\|\Delta\hat{\beta}_t\|_2$ in Equation (7) turned out to be small as the optimization progressed and the error bound of Lemma 4 became small. In addition, the upper bounds accurately identify unnecessary leaf nodes in the tree when the parameter vector converges as shown in Theorem 3. As a result, our approach could more accurately identify and prune unnecessary computations later in the optimization.

Impact of Hyperparameter r. Our method has a hyperparameter $r > 1$ that is the interval in which to obtain $\hat{\beta}_{t'}$ and $\|u_{t'}^{d+1}[G_j^d]\|_2$ in Equation (7) (line 9 in Algorithm 4). We investigated the effect on processing time by changing r between 2 to 10 under the same experimental settings described above. Figure 2 (f) shows the results. Our method reduced the processing time at small values of r, but the processing time increased as the value increased. This increase is because $\|\Delta\hat{\beta}_t\|_2$ in Equation (7) becomes large when the value of r is large. In this case, the error bound of Lemma 4 turned out to be large and it became difficult to prune unnecessary computations by using the upper bounds. Given these results, we recommend setting a small value for r in practice, e.g., $r = 2$.

Limitation. The reduction ratios of pFISTA-Mod (ours) relative to FISTA-Mod for the dense parameter vectors are moderate as shown in Figures 2 (c). This is because our method is based on the pruning method and cannot prune the computations due to the dense parameter vector. However, since a goal of sparse learning is to obtain a sparse parameter vector, it would be reasonable to assume the sparse settings as shown in Figure 2 (a) in practical use scenarios.

5.2 Value of Objective Function

Table 1 shows the final objective values of Canberra, Paris and Washington for FISTA and pFISTA. The full results are shown in the Appendix. Our methods achieved the same values of the objectives as those of the original methods. The indices of the selected features also matched those of the original methods though the results are omitted. This result supports Theorem 1: our method returned the same optimization results as the original method because it safely pruned unnecessary computations.

6 Conclusion

We proposed fast proximal gradient methods (PGMs) for tree-structured regularization. The bottlenecks of the original method are parameter updates including gradient computation and computation of the proximal operator with a hierarchical structure. We tackled the first bottleneck by pruning computations for leaf nodes of the tree: our method safely skips updates by identifying the parameters that must be zero on the basis of an upper bound. The second bottleneck was conquered by pruning computations for internal nodes: they are hierarchically pruned by summing the upper bounds of the child nodes. Our method provably guarantees the same optimization results and convergence rate as those of the original methods. In addition, since our method only changes the computation of the proximal operator, it can be used with various PGMs. Experiments showed that our method reduced the processing time by up to 56% from the original methods without any loss of accuracy.

References

1. Argyriou, A., Evgeniou, T., Pontil, M.: Convex Multi-task Feature Learning. Machine Learning **73**(3), 243–272 (2008)
2. Beck, A., Teboulle, M.: Fast Gradient-Based Algorithms for Constrained Total Variation Image Denoising and Deblurring Problems. Trans. Img. Proc. **18**(11), 2419–2434 (2009)
3. Beck, A., Teboulle, M.: A Fast Iterative Shrinkage-thresholding Algorithm for Linear Inverse Problems. SIAM Journal on Imaging Sciences **2**(1), 183–202 (2009)
4. Chen, X., Lin, Q., Kim, S., Carbonell, J.G., Xing, E.P.: Smoothing Proximal Gradient Method for General Structured Sparse Regression. The Annals of Applied Statistics **6**(2), 719–752 (2012)
5. Drori, Y., Teboulle, M.: Performance of First-order Methods for Smooth Convex Minimization: A Novel Approach. Math. Program. **145**(1-2), 451–482 (2014)
6. Fujiwara, Y., Ida, Y., Arai, J., Nishimura, M., Iwamura, S.: Fast Algorithm for the Lasso based L1-Graph Construction. Proc. VLDB Endow. **10**(3), 229–240 (2016)
7. Fujiwara, Y., Ida, Y., Shiokawa, H., Iwamura, S.: Fast Lasso Algorithm via Selective Coordinate Descent. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI). pp. 1561–1567 (2016)

8. Helou, E.S., Zibetti, M.V.W., Herman, G.T.: Fast Proximal Gradient Methods for Nonsmooth Convex Optimization for Tomographic Image Reconstruction. Sensing and imaging **21**(45) (2020)
9. Ida, Y., Fujiwara, Y., Kashima, H.: Fast Sparse Group Lasso. In: Advances in Neural Information Processing Systems (NeurIPS). pp. 1702–1710 (2019)
10. Ida, Y., Kanai, S., Adachi, K., Kumagai, A., Fujiwara, Y.: Fast Regularized Discrete Optimal Transport with Group-Sparse Regularizers. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI). pp. 7980–7987 (2023)
11. Ida, Y., Kanai, S., Fujiwara, Y., Iwata, T., Takeuchi, K., Kashima, H.: Fast Deterministic CUR Matrix Decomposition with Accuracy Assurance. In: Proceedings of International Conference on Machine Learning (ICML). pp. 4594–4603 (2020)
12. Ida, Y., Kanai, S., Kumagai, A.: Fast Block Coordinate Descent for Non-Convex Group Regularizations. In: International Conference on Artificial Intelligence and Statistics (AISTATS). pp. 2481–2493 (2023)
13. Ida, Y., Kanai, S., Kumagai, A., Iwata, T., Fujiwara, Y.: Fast Iterative Hard Thresholding Methods with Pruning Gradient Computations. In: Advances in Neural Information Processing Systems (NeurIPS). pp. 52836–52857 (2024)
14. Jenatton, R., Gramfort, A., Michel, V., Obozinski, G., Eger, E., Bach, F., Thirion, B.: Multiscale Mining of fMRI Data with Hierarchical Structured Sparsity. SIAM J. Img. Sci. **5**(3), 835–856 (2012)
15. Jenatton, R., Mairal, J., Obozinski, G., Bach, F.: Proximal Methods for Sparse Hierarchical Dictionary Learning. In: International Conference on Machine Learning (ICML). p. 487–494 (2010)
16. Jenatton, R., Mairal, J., Obozinski, G., Bach, F.: Proximal Methods for Hierarchical Sparse Coding. Journal of Machine Learning Research (JMLR) **12**(67), 2297–2334 (2011)
17. Kalnay, E., Kanamitsu, M., Kistler, R., Collins, W., Deaven, D., Gandin, L., Iredell, M., Saha, S., White, G., Woollen, J., Zhu, Y., Leetmaa, A., Reynolds, B., Chelliah, M., Ebisuzaki, W., Higgins, W., Janowiak, J., Mo, K.C., Ropelewski, C., Wang, J., Jenne, R., Joseph, D.: The NCEP/NCAR 40-Year Reanalysis Project. Bulletin of the American Meteorological Society **77**(3), 437–472 (1996)
18. Kim, D., Fessler, J.A.: An Optimized First-order Method for Image Restoration. In: IEEE International Conference on Image Processing (ICIP). pp. 3675–3679 (2015)
19. Kim, S., Xing, E.P.: Tree-Guided Group Lasso for Multi-Task Regression with Structured Sparsity. In: International Conference on Machine Learning (ICML). pp. 543–550 (2010)
20. Liang, J., Luo, T., Schönlieb, C.B.: Improving "Fast Iterative Shrinkage-Thresholding Algorithm": Faster, Smarter, and Greedier. SIAM Journal on Scientific Computing **44**(3), A1069–A1091 (2022)
21. Liu, J., Ye, J.: Moreau-Yosida Regularization for Grouped Tree Structure Learning. In: Advances in Neural Information Processing Systems (NeurIPS). pp. 1459–1467 (2010)
22. Lu, X., Wang, Y., Zhou, X., Zhang, Z., Ling, Z.: Traffic Sign recognition via Multimodal Tree-Structure Embedded Multi-Task Learning. IEEE Transactions on Intelligent Transportation Systems **18**(4), 960–972 (2017)
23. Ndiaye, E., Fercoq, O., Gramfort, A., Salmon, J.: Gap Safe Screening Rules for Sparsity Enforcing Penalties. Journal of Machine Learning Research (JMLR) **18(1)**, 4671–4703 (2017)

24. Nesterov, Y.: A Mmethod for Solving the Convex Programming Problem with Convergence Rate $\mathcal{O}(1/k^2)$. Proceedings of the USSR Academy of Sciences **269**, 543–547 (1983)
25. Parikh, N., Boyd, S.: Proximal Algorithms. Found. Trends Optim. **1**(3), 127–239 (2014)
26. Taylor, A.B., Hendrickx, J.M., Glineur, F.: Exact Worst-Case Performance of First-Order Methods for Composite Convex Optimization. SIAM Journal on Optimization **27**(3), 1283–1313 (2017)
27. Tibshirani, R.: Regression Shrinkage and Selection via the Lasso. Journal of the Royal Statistical Society **58**, 267–288 (1996)
28. Tibshirani, R., Bien, J., Friedman, J., Hastie, T., Simon, N., Taylor, J., Tibshirani, R.J.: Strong Rules for Discarding Predictors in Lasso-type Problems. Journal of the Royal Statistical Society Series B **74**(2), 245–266 (2012)
29. Yosida, K.: Functional Analysis. Springer (1968)
30. Yuan, M., Lin, Y.: Model Selection and Estimation in Regression with Grouped Variables. Journal of the Royal Statistical Society **68**(1), 49–67 (2006)
31. Zhao, P., Rocha, G., Yu, B.: Grouped and Hierarchical Model Selection through Composite Absolute Penalties. Department of Statistics, UC Berkeley, Tech. Rep **37** (05 2006)
32. Zhao, P., Rocha, G., Yu, B.: The Composite Absolute Penalties Family for Grouped and Hierarchical Variable Selection. The Annals of Statistics **37**(6A), 3468 – 3497 (2009)
33. Zhao, P., Yu, B.: Boosted Lasso. Tech. rep., Statistics Department, University of California, Berkeley (2004)
34. Zibetti, M.V.W., Helou, E.S., Regatte, R.R., Herman, G.T.: Monotone FISTA With Variable Acceleration for Compressed Sensing Magnetic Resonance Imaging. IEEE Transactions on Computational Imaging **5**, 109–119 (2019)