

# Fast Algorithm for the Lasso based $L_1$ -Graph Construction

Yasuhiro Fujiwara<sup>†‡\*</sup>, Yasutoshi Ida<sup>†</sup>, Junya Arai<sup>†</sup>, Mai Nishimura<sup>†</sup>, Sotetsu Iwamura<sup>†</sup>

<sup>†</sup>NTT Software Innovation Center, 3-9-11 Midori-cho Musashino-shi, Tokyo, Japan  
<sup>‡</sup>NTT Communication Science Laboratories, 2-4 Seika-cho Soraku-gun, Kyoto, Japan

<sup>\*</sup>Osaka University, 1-5 Yamadaoka, Suita-shi, Osaka, Japan

{fujiwara.yasuhiro, ida.yasutoshi, arai.junya, nishimura.mai,  
iwamura.sotetsu}@lab.ntt.co.jp

## ABSTRACT

The lasso-based  $L_1$ -graph is used in many applications since it can effectively model a set of data points as a graph. The lasso is a popular regression approach and the  $L_1$ -graph represents data points as nodes by using the regression result. More specifically, by solving the  $L_1$ -optimization problem of the lasso, the sparse regression coefficients are used to obtain the weights of the edges in the graph. Conventional graph structures such as  $k$ -NN graph use two steps, adjacency searching and weight selection, for constructing the graph whereas the lasso-based  $L_1$ -graph derives the adjacency structure as well as the edge weights simultaneously by using a coordinate descent. However, the construction cost of the lasso-based  $L_1$ -graph is impractical for large data sets since the coordinate descent iteratively updates the weights of all edges until convergence. Our proposal, *Castnet*, can efficiently construct the lasso-based  $L_1$ -graph. In order to avoid updating the weights of all edges, we prune edges that cannot have nonzero weights before entering the iterations. In addition, we update edge weights only if they are nonzero in the iterations. Experiments show that *Castnet* is significantly faster than existing approaches.

## 1. INTRODUCTION

We are now living in the big data era. With the rapid development of database technologies, high-dimensional data has become prevalent in many application domains which demand techniques to process data in an effective manner [21, 17]. From the data mining perspective, it is important to detect the hidden structures of high-dimensional data that can be collectively revealed only by processing large amounts of data. This is because the hidden structures allow us to associate, and thus utilize, the semantic information of data; data belonging to the same cluster share the same semantic information [9]. From a practical perspective, finding the hidden structures of high-dimensional data is a fundamental process in computational biology, human-computer

interaction, medical analysis, scientific data exploration, and information retrieval [10].

Graph is a fundamental data model in finding the hidden structures of high-dimensional data. The nodes and edges of a graph represent data points and the relationships among them, respectively. Since graphs are useful, various approaches to exploit graphs have been proposed in many communities [8, 9, 11, 19]. However, the database community has paid relatively little attention to developing graph construction approaches although it is a quite important research problem in real-world applications. Conceptually, a graph should capture the intrinsic complex structures induced by high dimensionality of the data. A primitive approach for graph construction is  $k$ -nearest neighbor ( $k$ -NN). In a  $k$ -NN graph, each node is connected to its  $k$ -nearest neighbor nodes by exploiting Euclidean distance. However, Euclidean distance is sensitive to data noise, a problem that is inherent in real-world applications [4].

In order to address this problem, Meinshausen et al. proposed to construct the  $L_1$ -graph by solving the  $L_1$ -optimization problem of the lasso [16]. The lasso is a popular least squares regression approach that represents each data point as the linear combination of the remaining data points along with coefficients [13]. By adding an  $L_1$ -norm regularization term, the lasso achieves the sparsity of the solution; it assigns exactly zero coefficients to the most irrelevant data points in the regression. The idea of Meinshausen et al. is to exploit the lasso to construct graphs; they applied the lasso to each node, where the regression coefficients are used as edge weights in the graph. In the approach, the neighborhood relationships and edge weights are simultaneously obtained by solving the  $L_1$ -optimization problem of the lasso. Thus, the resulting graph is referred as the  $L_1$ -graph. The  $L_1$ -graph is fundamentally different from traditional  $k$ -NN graphs. Since the  $L_1$ -graph utilizes the higher order relationships among data points, it is robust to data noise unlike  $k$ -NN graphs [4]. In addition, the  $L_1$ -graph has small memory consumption since the lasso can sparsely represent each node by using the small number of remaining nodes [16].

Even though the lasso-based  $L_1$ -graph is used in many applications, its construction incurs high computation cost [4, 14, 15]. The graph construction approach proposed by Meinshausen et al. picks up nodes one by one and computes the lasso for each node. In terms of processing speed, the coordinate descent [5] is an efficient approach to the lasso that iteratively updates edge weights one at a time to compute the regression result. However, it is infeasible to apply the coordinate descent to large-scale data sets. This is be-

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-nd/4.0/>. For any use beyond those covered by this license, obtain permission by emailing [info@vldb.org](mailto:info@vldb.org).

*Proceedings of the VLDB Endowment*, Vol. 10, No. 3  
Copyright 2016 VLDB Endowment 2150-8097/16/11.

cause it iteratively computes the weights of all edges until convergence even if weights of almost all edges are zero due to the  $L_1$ -norm regularization term of the lasso.

## 1.1 Problem Statement

If  $N$  and  $M$  are the numbers of nodes and dimensions, respectively, a set of high-dimensional data is represented as matrix  $\mathbf{X}$  of  $N \times M$  size. We address the following problem:

**Problem** ( $L_1$ -GRAPH CONSTRUCTION).

**Given:** Matrix  $\mathbf{X}$  of the high-dimensional data and tuning parameter  $\lambda$ .

**Construct:** the  $L_1$ -graph of  $N$  nodes with edge weights computed by the lasso, efficiently.

In this problem, tuning parameter  $\lambda$  is a positive regularization parameter that balances the trade-off between the loss function and the  $L_1$ -norm regularization of the lasso. Our approach can be used in various applications as shown below due to the generality of the problem.

**Brain Analysis.** Recent advances in computational neuroscience allow researchers to use large-scale gene expression databases in analyzing genes and anatomies. It is crucial to detect gene-anatomy association for understanding brain function from molecular and genetic information. The lasso-based  $L_1$ -graph has been used to model the anatomical organization of brain structure [14]. In the graph, each node represents the spatial location of a gene, and the edges between nodes encode the correlations between locations in the brain. Since spatially adjacent regions tend to exhibit correlated expression patterns, most of the edges connect to spatially adjacent regions in the graph. However, the graph revealed an apparent exception; the nodes annotated as dentate gyrus (DG) were found to be highly correlated to many other nodes in distant regions of the brain. DG is an important area for learning and memory to process spatial information. According to neuroanatomy, DG receives multiple sensory inputs including visual and auditory. Thus, DG plays an important role in blocking and filtering excitatory activity from the inputs.

**Motion Segmentation.** In the field of computer vision, motion segmentation is a fundamental process in decomposing a video into moving objects and a background. Motion segmentation is a pre-processing step in surveillance, tracking, and action recognition. It is important to develop a motion segmentation approach that is robust against noise since data is never clean in real world applications. For example, in human image recognition, the data exhibits different degrees of noise depending on rotation, lighting, scale, and so on. Although many approaches have been proposed in the literature, the approach based on the affine camera model is arguably the most popular for motion segmentation. An important intuition behind the approach is to use motion; we can easily discern independently moving objects by simply seeing their motion. In the approach, all the trajectories associated with a single rigid object lie in a linear subspace. Therefore, motion segmentation can be achieved by clustering trajectories into different motion subspaces [4]. More specifically, the approach constructs the  $L_1$ -graph from the trajectory data to apply spectral clustering since the  $L_1$ -graph is robust against noise unlike other types of graphs. The approach is more effective than existing approaches.

**Lesion Detection.** Automatic lesion detection is an important process in early medical screening or providing second

opinions for decision making. In computer-aided diagnosis, analyzing CT images is an essential step in generating 3D models. Although it is conceptually simple that lesions are just regions whose features differ from those of normal anatomical structures, it is technically challenging to detect lesions in CT images due to the poor contrast between the lesion and surrounding tissues, and the high variability of lesion shape. The lasso-based  $L_1$ -graph is a robust approach to lesion detection where a voxel in a CT image corresponds to a node in the graph [15]. The approach identifies lesion regions in CT images by applying a semi-supervised learning approach that propagates labels of manually-labeled nodes to unlabeled nodes in the graph. The approach is motivated by the superior discriminant power of the sparse representations yielded by the lasso. Due to the sparsity constraint imposed by the  $L_1$ -norm regularization term, the  $L_1$ -graph have a sparse structure where only nodes of similar features are connected to each other. As a result, the approach can effectively determine the lesion regions from noisy data. The approach experimentally shows that it can distinctly identify prostate cancer; one of the leading causes of male cancer death in the United States.

## 1.2 Contributions

In this paper, we propose *Castnet*, a novel and efficient approach to constructing the lasso-based  $L_1$ -graph. Before entering the iterations, we prune those edges whose weights cannot be nonzero. In addition, we efficiently update nonzero weights in each iteration by pruning edges whose weights are zero. Although our approach skips unnecessary weight updates similarly to the state-of-the-art approach of the lasso [7], we substantially improve the efficiency compared to the previous approach by effectively pruning the inner product computations used for graph construction. As a result, our approach is 130 times faster than the state-of-the-art approach. Even though the lasso-based  $L_1$ -graph was designed to overcome the sensitivity of  $k$ -NN graphs to data noise, it is seldom applied to large-scale data sets due to the high computation cost. However, our approach can efficiently construct graphs, which will improve the usefulness of many applications such as brain analysis, motion segmentation, and lesion detection.

The remainder of this paper is organized as follows: Section 2 describes related work. Section 3 gives an overview of the background. Section 4 introduces our approach. Section 5 reviews our experiments and the results. Section 6 details case-studies. Section 7 provides our conclusions.

## 2. RELATED WORK

Meinshausen et al. proposed the lasso-based  $L_1$ -graph as more robust approach against noise than traditional  $k$ -NN graphs [16]. Their approach picks up nodes one by one and uses the lasso to compute weights for the node. A popular approach to solving the  $L_1$ -optimization problem of the lasso is the coordinate descent [5]. To resolve the  $L_1$ -optimization problem, it iteratively updates edge weights one at a time. In terms of the efficiency, Sling is a state-of-the-art approach based on the coordinate descent although it is not so useful to construct the  $L_1$ -graph [7]. Sling uses the sequential strong rule to discard unnecessary edges. It is based on the assumption that a grid of tuning parameters ( $\lambda_1 > \lambda_2 > \lambda_3 > \dots$ ) is used in solving the lasso the

**Table 1: Definitions of main symbols.**

Symbol	Definition
$N$	Number of nodes in the graph
$M$	Number of dimensions of each data point
$m$	Target rank of SVD
$\lambda$	Tuning parameter
$p$	Node for weight computation
$e_p[u]$	Edge from node $u$ to $p$
$w_p[u]$	Weight of edge $e_p[u]$
$d_p$	Number of edges to node $p$
$K[u \mathbf{w}_p]$	KKT condition score of node $u$ for weight vector $\mathbf{w}_p$
$\bar{K}[u \mathbf{w}_p]$	Upper bound of KKT condition score $K[u \mathbf{w}_p]$
$\underline{K}[u \mathbf{w}_p]$	Lower bound of KKT condition score $K[u \mathbf{w}_p]$
$\mathbf{x}_p$	$p$ -th row vector of matrix $\mathbf{X}$
$\mathbf{w}_p$	$1 \times N$ weight vector of node $p$
$\mathbf{X}$	$N \times M$ matrix of high-dimensional data
$\mathbb{V}$	Set of nodes in the graph
$\mathbb{M}[\mathbf{w}_p]$	Set of edges that must have nonzero weights for $\mathbf{w}_p$
$\mathbb{C}[\mathbf{w}_p]$	Set of edges that can have nonzero weights for $\mathbf{w}_p$

same as the other screening techniques [24]; the tuning parameter controls the sparsity of the solution. In each iteration, Sling estimates parameters of the soft-thresholding operator used in updating weights. In addition, Sling employs the covariance-based approach to efficiently update weights by computing the inner products of all nodes before entering each iteration [6]. After convergence, Sling checks the Karush-Kuhn-Tucker (KKT) condition of all discarded edges since the sequential strong rule may erroneously discard edges of nonzero weights [24]. Although Sling is faster than the original coordinate descent approach for the lasso, it is not practical to construct the  $L_1$ -graph by directly using Sling. This is because the tuning parameter is constant when constructing a graph; the sequential strong rule is not effective in pruning unnecessary edges since  $\lambda$  does not dynamically change when constructing a graph. In addition, Sling computes the inner products of all nodes even if almost all edges are zero as a result of the lasso. To the best of our knowledge, Castnet is the first approach that can efficiently construct the lasso-based  $L_1$ -graph.

### 3. PRELIMINARY

We formally define the notations and introduce the background of this paper. Table 1 lists the main symbols and their definitions. In the lasso-based  $L_1$ -graph, nodes and edges correspond to data points and their relationships, respectively. Let  $p$  be a node in the graph and  $\mathbb{V}$  be the set of nodes in the graph, the lasso is used to compute the weights of edges by picking up each node such that  $p \in \mathbb{V}$ . Since the lasso sets zero coefficients to almost all edges, the graph has a sparse structure. Let  $\mathbf{X} \in \mathcal{R}^{N \times M}$  be the matrix of  $N$  data points that have  $M$  dimensional features and  $\mathbf{x}_p = (x_p[1], x_p[2], \dots, x_p[M])$  be the  $p$ -th row vector in matrix  $\mathbf{X}$ , vector  $\mathbf{x}_p$  corresponds to the  $p$ -th data point or node  $p$ . We assume the each row vector is centered and normalized [6]; each vector has average and variance of 0 and 1, respectively. Let  $\mathbf{w}_p$  be a  $1 \times N$  weight vector whose  $u$ -th element  $w_p[u]$  is the weight of the edge from node  $u$  to  $p$ . In the lasso-based  $L_1$ -graph, each node is represented as the sparse linear superposition of other nodes by solving the lasso optimization problem. Specifically, we compute weights of edges to node  $p$  by minimizing the following objective function of the regression where  $w_p[p] = 0$  [16]:

$$\min_{\mathbf{w}_p \in \mathcal{R}^N} \frac{1}{2M} \|\mathbf{x}_p - \mathbf{w}_p \mathbf{X}\|_2^2 + \lambda \|\mathbf{w}_p\|_1 \quad (1)$$

where  $\|\cdot\|_1$  and  $\|\cdot\|_2$  are the  $L_1$ -norm and  $L_2$ -norm of a vector, respectively, and  $\lambda > 0$  is a tuning parameter. Note that the graph does not have self-loop edges since  $w_p[p] = 0$  for each picked up node. In Equation (1), the first term corresponds to the squared loss by regression; the second term corresponds to an  $L_1$ -norm constraint for the coefficients. Therefore, Equation (1) implements the regularization of weight vector  $\mathbf{w}_p$  by trading off the accuracy of the regression for a reduction in the sparsity of coefficients; it has the effect of giving a solution to the objective function with few edges of nonzero weights [23]. It is clear from Equation (1) that the graph has improved sparsity as we raise tuning parameter  $\lambda$ . As described in the previous paper [6], if some nodes are strongly correlated, a single node would be used to represent node  $p$ ; only an edge from the single node is likely to have nonzero weight in that case. If matrix  $\mathbf{X}$  has full row rank, we have a unique solution for the optimization problem of Equation (1) [13]. Otherwise, which is necessarily the case when  $N > M$ , there may not be a unique solution.

In order to efficiently solve the lasso, Friedman et al. proposed the coordinate descent that updates weights of edges one at a time until they reach convergence [5]. The coordinate descent partially conducts optimization with respect to weight  $w_p[u]$  by supposing that it has already estimated other weights for all nodes  $u$  such that  $u \in \mathbb{V} \setminus p$ . Specifically, the coordinate descent updates weights as follows:

$$w_p[u] \leftarrow S[z_p[u|\mathbf{w}_p], \lambda] \quad (2)$$

where  $S[\cdot, \cdot]$  is the following soft-thresholding operator [5]:

$$S[z_p[u|\mathbf{w}_p], \lambda] = \begin{cases} z_p[u|\mathbf{w}_p] - \lambda & (z_p[u|\mathbf{w}_p] > 0 \text{ and } |z_p[u|\mathbf{w}_p]| > \lambda) \\ z_p[u|\mathbf{w}_p] + \lambda & (z_p[u|\mathbf{w}_p] < 0 \text{ and } |z_p[u|\mathbf{w}_p]| > \lambda) \\ 0 & (|z_p[u|\mathbf{w}_p]| \leq \lambda) \end{cases} \quad (3)$$

In addition,  $z_p[u|\mathbf{w}_p]$ , a parameter of node  $u$  for weight vector  $\mathbf{w}_p$ , is computed as follows:

$$z_p[u|\mathbf{w}_p] = \frac{1}{M} \sum_{i=1}^M x_u[i](x_p[i] - \tilde{x}_p^{(u)}[i]) \quad (4)$$

where  $\tilde{x}_p^{(u)}[i]$  is the regression result for element  $x_p[i]$  without using node  $u$  computed as follows:

$$\tilde{x}_p^{(u)}[i] = \sum_{v \in \mathbb{V} \setminus \{p, u\}} w_p[v] x_v[i] \quad (5)$$

As shown by the above equations,  $z_p[u|\mathbf{w}_p]$  and  $\tilde{x}_p^{(u)}[i]$  are required to update a weight by Equation (2). Since  $\tilde{x}_p^{(u)}[i]$  is different from each weight, above equations indicate that the original coordinate descent approach needs different regression results for each weight. It takes  $O(M)$  time to compute  $z_p[u|\mathbf{w}_p]$  from Equation (4). In addition, Equation (4) additionally requires  $O(d_p)$  time to compute  $\tilde{x}_p^{(u)}[i]$  for each dimension by Equation (5) if  $d_p$  is the number of nonzero weight edges to node  $p$ ;  $d_p$  is degree of node  $p$ . As a result, if  $T$  is the number of updates until convergences, it takes  $O(d_p MT)$  time to compute edge weights to node  $p$  by using Equation (2). As a result, the computation cost of construct the  $L_1$ -graph can be prohibitive for large datasets.

### 4. PROPOSED METHOD

This section presents Castnet, which can efficiently construct the lasso-based  $L_1$ -graph. Section 4.1 overviews the ideas that underlie our approach. Section 4.2 describes our approach to determine the edge sets to update weights by

using upper and lower bounds of KKT condition score. In Section 4.3, we introduce the approach that computes the bounds of KKT condition score. Section 4.4 shows the graph construction algorithm along with its theoretical properties.

## 4.1 Main Ideas

In constructing the lasso-based  $L_1$ -graph, the original coordinate descent approach incurs high computation cost. This is because (1) it updates the weights of all edges until the convergence although almost all edges have zero weights due to the  $L_1$ -norm constraint of the lasso and (2) it updates each weight by computing a different regression result for each weight as described in Section 3.

For efficient graph construction, our approach is to prune unnecessary edges that cannot have nonzero weights so as to determine a set of edges for weight updating. More specifically, we first limit weight updates to the set of edges that *must* have nonzero weight until convergence. After that, we update weights of the set of edges that *can* have nonzero weight. As a result, we can efficiently obtain weights for each node. Theoretically, we efficiently determine edge sets that must/can have nonzero weight by computing the upper and lower bounds of KKT condition scores; KKT condition was originally used to find edges that are erroneously discarded by the sequential screening rule as described in Section 2 [24]. In addition, we exploit two types of efficient update computations for weights. The first type of update computation is based on the residual of the lasso that is the same for each weight. Since we do not need to compute different regression results in each weight update, we can greatly enhance computation speed comparing to the original coordinate descent approach. We use this type of computation for edges that have zero weight. Otherwise, we update the weight using the covariance-based approach [6]; this is the second type of update computation. Our approach can more efficiently update weights than the previous approach even though we exploit the same covariance-based approach [7]. This is because we use the residual-based update computation if the weight of an edge is zero. As described in Section 2, the covariance-based approach requires the computations of inner products for all edges to update weights before entering the iterations. Since we do not update all edge weights by the covariance-based approach, we can effectively prune the inner product computations unlike the previous approach. Furthermore, we skip update computations for edges if they do not have nonzero weight. This is because the edge sets to update may include zero weight edges. By skipping the update computations for such edges, we improve the efficiency with which nonzero weights are updated in the iterations.

Even though we adopt a different strategy from the previous approach in constructing the  $L_1$ -graph, we can probably guarantee to output the same graph as the previous approach if matrix  $\mathbf{X}$  has full row rank. This is because our approach converges on the unique lasso solution in that case. Therefore, our approach is effective and efficient in constructing the lasso-based  $L_1$ -graph.

## 4.2 Weight Updates

This section describes the approach to updating weights. We first introduce the definitions of edge sets as well as their properties in Section 4.2.1. We then describe the weight update computations in Section 4.2.2. We finally show our

approach of skipping update computations for zero weight edges in Section 4.2.3.

### 4.2.1 Definitions

Before entering the iterations, we compute the two edge sets of  $\mathbb{M}[\mathbf{w}_p]$  and  $\mathbb{C}[\mathbf{w}_p]$ .  $\mathbb{M}[\mathbf{w}_p]$  is the set of edges whose weights *must* be nonzero when updated with weight vector  $\mathbf{w}_p$ . Similarly,  $\mathbb{C}[\mathbf{w}_p]$  is the set of edges that *can* have nonzero weights when updated with weight vector  $\mathbf{w}_p$ . Theoretically, edge set  $\mathbb{M}[\mathbf{w}_p]$  and  $\mathbb{C}[\mathbf{w}_p]$  are based on the KKT condition [24]. We obtain these edge sets by computing the upper and lower bounds of KKT condition scores that are given in Section 4.3. Before defining the edge sets, we show how the KKT condition can check whether weight  $w_p[u]$  has nonzero score or not for weight vector  $\mathbf{w}_p$  as follows:

**Definition 1** (KKT CONDITION [24]). *If  $K[u|\mathbf{w}_p]$  is the KKT score of node  $u$  for weight vector  $\mathbf{w}_p$ , we have*

$$K[u|\mathbf{w}_p] = \frac{1}{\lambda}w_p[u] + \frac{1}{\lambda M}(\mathbf{x}_p - \mathbf{w}_p\mathbf{X})\mathbf{x}_u^\top \quad (6)$$

where  $\mathbf{x}_u^\top$  is the transpose of  $\mathbf{x}_u$ . For all nodes  $u$  such that  $u \in \mathbb{V}/p$ , if the weight of the edge from node  $u$  to  $p$  is updated with weight vector  $\mathbf{w}_p$ ,  $K[u|\mathbf{w}_p]$  has the following scores based on weight  $w_p[u]$  after the update:

$$\begin{cases} K[u|\mathbf{w}_p] > 1 & (\text{iff } w_p[u] > 0) \\ K[u|\mathbf{w}_p] < -1 & (\text{iff } w_p[u] < 0) \\ -1 \leq K[u|\mathbf{w}_p] \leq 1 & (\text{iff } w_p[u] = 0) \end{cases} \quad (7)$$

It is clear that we can obtain a set of edges that have nonzero weights by directly computing the KKT condition scores from Equation (6). This, however, incurs high computation cost; it takes  $O(NM)$  time for determining each KKT condition score in the worst case since the sizes of  $\mathbf{x}_p$ ,  $\mathbf{w}_p$ ,  $\mathbf{X}$ , and  $\mathbf{x}_u^\top$  in Equation (6) are  $1 \times M$ ,  $1 \times N$ ,  $N \times M$ , and  $M \times 1$ , respectively. To overcome this problem, we efficiently compute the upper and lower bounds of the KKT condition scores introduced in Section 4.3. If  $\overline{K}[u|\mathbf{w}_p]$  and  $\underline{K}[u|\mathbf{w}_p]$  are the upper and lower bounds of KKT condition score of  $K[u|\mathbf{w}_p]$ , respectively, we have  $\underline{K}[u|\mathbf{w}_p] \leq K[u|\mathbf{w}_p] \leq \overline{K}[u|\mathbf{w}_p]$ . By exploiting the bounds, the two edge sets are given as follows:

**Definition 2** (EDGE SET  $\mathbb{M}$ ). *For node  $u$  such that  $u \neq p$ , the definition of edge set  $\mathbb{M}[\mathbf{w}_p]$  is given as follows:*

$$\mathbb{M}[\mathbf{w}_p] = \{e_p[u] : \overline{K}[u|\mathbf{w}_p] < -1 \text{ or } \underline{K}[u|\mathbf{w}_p] > 1\} \quad (8)$$

**Definition 3** (EDGE SET  $\mathbb{C}$ ). *Letting  $u \neq p$ , the following equation defines edge set  $\mathbb{C}[\mathbf{w}_p]$ :*

$$\mathbb{C}[\mathbf{w}_p] = \{e_p[u] : \overline{K}[u|\mathbf{w}_p] > 1 \text{ or } \underline{K}[u|\mathbf{w}_p] < -1\} \quad (9)$$

In Equations (8) and (9),  $e_p[u]$  is the edge from node  $u$  to  $p$ . Therefore, an edge is included in  $\mathbb{M}[\mathbf{w}_p]$  if  $\overline{K}[u|\mathbf{w}_p] < -1$  or  $\underline{K}[u|\mathbf{w}_p] > 1$  holds. Similarly, when we have  $\overline{K}[u|\mathbf{w}_p] > 1$  or  $\underline{K}[u|\mathbf{w}_p] < -1$ , an edge is placed in edge set  $\mathbb{C}[\mathbf{w}_p]$ . The KKT condition yields the following properties for edge sets:

**Lemma 1** (EDGE SET  $\mathbb{M}$ ). *Each edge  $e_p[u]$  such that  $e_p[u] \in \mathbb{M}[\mathbf{w}_p]$  must have nonzero weight if its weight is updated with weight vector  $\mathbf{w}_p$ .*

**Proof** For each edge included in edge set  $\mathbb{M}[\mathbf{w}_p]$ , we have  $\overline{K}[u|\mathbf{w}_p] < -1$  or  $\underline{K}[u|\mathbf{w}_p] > 1$  if its weight is updated for weight vector  $\mathbf{w}_p$  from Definition 2. In the case

of  $\overline{K}[u|\mathbf{w}_p] < -1$ , we have  $K[u|\mathbf{w}_p] \leq \overline{K}[u|\mathbf{w}_p] < -1$  since  $\overline{K}[u|\mathbf{w}_p]$  is the upper bound of  $K[u|\mathbf{w}_p]$ . Therefore, weight  $w_p[u]$  of edge  $e_p[u]$  must have negative score from the KKT condition. If  $\underline{K}[u|\mathbf{w}_p] > 1$  holds for edge  $e_p[u]$ , we similarly have  $K[u|\mathbf{w}_p] \geq \underline{K}[u|\mathbf{w}_p] > 1$  since  $K[u|\mathbf{w}_p] \geq \underline{K}[u|\mathbf{w}_p]$  holds. As a result, edge  $e_p[u]$  must have positive weight in this case from the KKT condition.  $\square$

**Lemma 2** (EDGE SET  $\mathbb{C}$ ). *For each edge  $e_p[u]$  such that  $e_p[u] \in \mathbb{C}[\mathbf{w}_p]$ , its weight  $w_p[u]$  can have nonzero score if we perform the update computation with weight vector  $\mathbf{w}_p$ .*

**Proof** To prove Lemma 2, we show that the weight of an edge must be zero after being updated with weight vector  $\mathbf{w}_p$  if  $\overline{K}[u|\mathbf{w}_p] \leq 1$  and  $\underline{K}[u|\mathbf{w}_p] \geq -1$  hold. In the case of  $\overline{K}[u|\mathbf{w}_p] \leq 1$  and  $\underline{K}[u|\mathbf{w}_p] \geq -1$ , we have  $-1 \leq \underline{K}[u|\mathbf{w}_p] \leq K[u|\mathbf{w}_p] \leq \overline{K}[u|\mathbf{w}_p] \leq 1$ . Therefore, such an edge cannot have nonzero weight from the KKT condition. In addition, it is clear that such an edge cannot be included in edge set  $\mathbb{C}[\mathbf{w}_p]$  from Definition 3. As a result, if we have  $\overline{K}[u|\mathbf{w}_p] > 1$  or  $\underline{K}[u|\mathbf{w}_p] < -1$  for an edge, the edge can have nonzero weight and so is included in  $\mathbb{C}[\mathbf{w}_p]$ .  $\square$

In terms of the relationship between edge set  $\mathbb{M}[\mathbf{w}_p]$  and  $\mathbb{C}[\mathbf{w}_p]$ , we have the following property:

**Lemma 3** (EDGE SET  $\mathbb{M}$  AND  $\mathbb{C}$ ). *If an edge is included in  $\mathbb{M}[\mathbf{w}_p]$ , the edge must be included in  $\mathbb{C}[\mathbf{w}_p]$ .*

**Proof** From Definition 2, we have  $\overline{K}[u|\mathbf{w}_p] < -1$  or  $\underline{K}[u|\mathbf{w}_p] > 1$  for an edge if the edge is included in  $\mathbb{M}[\mathbf{w}_p]$ . If  $\overline{K}[u|\mathbf{w}_p] < -1$  holds, we have  $\underline{K}[u|\mathbf{w}_p] \leq \overline{K}[u|\mathbf{w}_p] < -1$  for the edge. In addition, if we have  $\underline{K}[u|\mathbf{w}_p] > 1$ ,  $\overline{K}[u|\mathbf{w}_p] \geq \underline{K}[u|\mathbf{w}_p] > 1$  holds for the edge. Therefore, it is clear that such the edge is also included in  $\mathbb{C}[\mathbf{w}_p]$  from Definition 3.  $\square$

Lemma 3 indicates that, if an edge is determined to have nonzero weight by  $\mathbb{M}[\mathbf{w}_p]$ , the edge is also determined to have nonzero weight by  $\mathbb{C}[\mathbf{w}_p]$ . Similarly, Lemma 3 indicates that, if an edge is determined not to have nonzero weight by  $\mathbb{C}[\mathbf{w}_p]$ , the edge is also determined not to have nonzero weight by  $\mathbb{M}[\mathbf{w}_p]$ ; an edge cannot have nonzero weight if it is not included in  $\mathbb{C}[\mathbf{w}_p]$ . We determine  $\mathbb{M}[\mathbf{w}_p]$  and  $\mathbb{C}[\mathbf{w}_p]$  before entering the iterations to efficiently perform update computations for edges that must/can have nonzero weights.

#### 4.2.2 Efficient Update Computation

The edge sets introduced in the previous section can, before entering the iterations, reduce the number of edges to update. This section describes our approach to improving the speed of a single update computation in the iterations. As shown in Equation (2), the original coordinate descent approach updates weights by exploiting soft-thresholding operator  $S[z_p[u|\mathbf{w}_p], \lambda]$ . Since parameter  $z_p[u|\mathbf{w}_p]$  requires different regression results for each weight as described in Section 3, the original coordinate descent approach incurs high computation cost. In our approach, we use two types of efficient update computations. The first type of computation updates weight by computing parameter  $z_p[u|\mathbf{w}_p]$  without demanding different regression results as follows:

$$z_p[u|\mathbf{w}_p] = \frac{1}{M} \sum_{i=1}^M x_u[i] r_p[i] + w_p[u] \quad (10)$$

where  $r_p[i]$  is the residual for element  $x_p[i]$  in the regression result by the lasso computed as follows:

$$r_p[i] = x_p[i] - \sum_{v \in \mathbb{V} \setminus p} w_p[v] x_v[i] \quad (11)$$

Since the residual used in Equation (11) is the same for each weight, we can efficiently compute parameter  $z_p[u|\mathbf{w}_p]$  if the regression result does not change after the update. While Equation (4) needs  $O(d_p M)$  time, Equation (10) requires  $O(M)$  time to compute parameter  $z_p[u|\mathbf{w}_p]$ . For Equation (10), we have the following property:

**Lemma 4** (PARAMETER  $z$ ). *Equation (4) and (10) yield the same result in computing parameter  $z_p[u|\mathbf{w}_p]$ .*

**Proof** From Equation (5) and (11), we have

$$\begin{aligned} \tilde{x}_p^{(u)}[i] &= \sum_{v \in \mathbb{V} \setminus \{p, u\}} w_p[v] x_v[i] = \sum_{v \in \mathbb{V} \setminus p} w_p[v] x_v[i] - w_p[u] x_u[i] \\ &= x_p[i] - r_p[i] - w_p[u] x_u[i] \end{aligned}$$

Therefore, from Equation (4)

$$\begin{aligned} z_p[u|\mathbf{w}_p] &= \frac{1}{M} \sum_{i=1}^M x_u[i] (x_p[i] - \tilde{x}_p^{(u)}[i]) \\ &= \frac{1}{M} \sum_{i=1}^M x_u[i] (r_p[i] + w_p[u] x_u[i]) \\ &= \frac{1}{M} \sum_{i=1}^M x_u[i] r_p[i] + \frac{1}{M} w_p[u] \sum_{i=1}^M (x_u[i])^2 \end{aligned}$$

Since the variance of vector  $\mathbf{x}_u$  is 1 as described in Section 3, we have  $\sum_{i=1}^M (x_u[i])^2 = M$ . As a result,

$$z_p[u|\mathbf{w}_p] = \frac{1}{M} \sum_{i=1}^M x_u[i] r_p[i] + w_p[u]$$

which completes the proof from Equation (10).  $\square$

We exploit Equation (10) if an edge has zero weight before the update. Since such an edge is expected to have zero weight again after the update [5], the residual is likely to have the same score after the update. Therefore, we can effectively utilize residual  $r_p[i]$  again in the next iteration in updating other weights. Even if the edge has nonzero weight after the update, we can incrementally update residual  $r_p[i]$  in  $O(1)$  time by exploiting Equation (11).

If an edge has nonzero weight before the update, we use the covariance-based approach to update weights since it can efficiently compute parameter  $z_p[u|\mathbf{w}_p]$  in  $O(d_p)$  time [6]:

$$z_p[u|\mathbf{w}_p] = w_p[u] + \frac{1}{M} (\langle \mathbf{x}_p, \mathbf{x}_u \rangle - \sum_{v: w_p[v] > 0} w_p[v] \langle \mathbf{x}_v, \mathbf{x}_u \rangle) \quad (12)$$

where  $\langle \mathbf{x}_p, \mathbf{x}_u \rangle$  is the inner product of vector  $\mathbf{x}_p$  and  $\mathbf{x}_u$ , i.e.,  $\langle \mathbf{x}_p, \mathbf{x}_u \rangle = \sum_{i=1}^M x_p[i] x_u[i]$ . Note that Equation (4) and (12) give the same result. Although we exploit the same covariance-based approach, our approach has less computation cost than the previous approach [7]. Before updating weights, the covariance-based approach requires inner product computations of nonzero weights to all edges undergoing update computations. Since the previous approach updates the weights of all the edges by the covariance-based approach, it needs  $O(d_p N M)$  time to compute inner products. On the other hand, we exploit the covariance-based approach only if the edge has nonzero weight before the update. As a result, our approach takes  $O(d_p^2 M)$  time to compute the inner products. Note that we have  $d_p \ll N$  since the  $L_1$ -graph has sparse structure.

#### 4.2.3 Gradual Edge Addition

As shown in Section 4.2.1, we determine edge set  $\mathbb{M}[\mathbf{w}_p]$  and  $\mathbb{C}[\mathbf{w}_p]$  before entering the iterations. If  $\mathbb{U}$  is an edge set to update weights, we can naively reduce the number of updates by setting  $\mathbb{U} = \mathbb{M}[\mathbf{w}_p]$  and  $\mathbb{U} = \mathbb{C}[\mathbf{w}_p]$  to directly apply the coordinate descent. However, this approach may iteratively perform update computations for zero weight edges since we use the upper and lower bounds to obtain  $\mathbb{M}[\mathbf{w}_p]$

and  $\mathbb{C}[\mathbf{w}_p]$ ; we exploit the approximations to determine  $\mathbb{M}[\mathbf{w}_p]$  and  $\mathbb{C}[\mathbf{w}_p]$ . This section describes the approach that skips unnecessary update computations to improve the efficiency of the  $L_1$ -graph construction<sup>1</sup>.

For efficient graph construction, we add edges one by one from edge set  $\mathbb{A}$  to edge set  $\mathbb{U}$ . In our approach, we first obtain edge set  $\mathbb{U}$  as edges of nonzero weight and perform update computations until convergence for edge set  $\mathbb{U}$ . After convergence, we compute  $\mathbb{M}[\mathbf{w}_p]$  and obtain set of adding edges as  $\mathbb{A} = \mathbb{M}[\mathbf{w}_p] \setminus \mathbb{U}$ . Then, we perform an update computation for each edge in  $\mathbb{A}$  one by one. If an edge has nonzero weight after the update computation, we add the edge to  $\mathbb{U}$  and update weights of  $\mathbb{U}$  until convergence. Otherwise, we do not iteratively perform update computations for the edge. Similarly, we compute  $\mathbb{A} = \mathbb{C}[\mathbf{w}_p] \setminus \mathbb{U}$  and update weights. Since we avoid iterative computations if an edge does not have nonzero weight, we can improve the efficiency of weight updates. Theoretically, this approach is based on the following property of converged weights:

**Lemma 5** (CONVERGED WEIGHTS). *Let  $\mathbf{w}'_p$  be a converged weight vector for edge set  $\mathbb{U}[\mathbf{w}_p] + u$  obtained by the coordinate descent based on weight vector  $\mathbf{w}_p$ . If (1) weight vector  $\mathbf{w}_p$  reaches convergence by the coordinate descent with edge set  $\mathbb{U}[\mathbf{w}_p]$  and (2)  $w_p[u] = 0$  and  $w'_p[u] = 0$  hold for node  $u$ , we have  $\mathbf{w}'_p = \mathbf{w}_p$ .*

**Proof** Since  $w_p[u] = 0$  and  $w'_p[u] = 0$ , the  $u$ -th element of  $\mathbf{w}_p$  and  $\mathbf{w}'_p$  are obviously the same. If  $w'_p[u] = 0$  holds, it is clear that weight  $w'_p[u]$  does not have any impact on the update results from Equation (5). In addition, weight vector  $\mathbf{w}_p$  has already reached convergence by exploiting edge set  $\mathbb{U}[\mathbf{w}_p]$ . Therefore, we have  $w'_p[u] = w_p[u]$  for all elements in  $\mathbb{U}[\mathbf{w}_p]$  after the update computations to obtain weight vector  $\mathbf{w}'_p$  if we use weight vector  $\mathbf{w}_p$  in the coordinate descent. As a result, we have  $\mathbf{w}'_p = \mathbf{w}_p$ .  $\square$

Lemma 5 indicates that we can skip the update computations for the edge from node  $u$  to  $p$  as its weight is zero in the iterations without changing the regression results yielded by the coordinate descent. This skipping of unnecessary computations allows us to efficiently update nonzero weights. We show the detail algorithm of this approach in Section 4.4 as a part of the graph construction algorithm.

### 4.3 Upper and Lower Bounds

As described in Section 4.2, we can prune edges that do not have nonzero weight by computing KKT condition scores. However, it needs the high cost of  $O(NM)$  time to compute the score for each edge because the size of data matrix  $\mathbf{X}$  is  $N \times M$ . This section introduces our approaches to efficiently compute the upper and lower bounds of KKT conditions scores. Section 4.3.1 describes our approach to compute the bounds by using SVD (singular value decomposition). Section 4.3.2 shows the incremental approach for bound computations.

#### 4.3.1 SVD-Based Bound Computations

In this approach, we approximate matrix  $\mathbf{X}$  by computing SVD. We adopt SVD since it gives high approximation quality in terms of squared loss [20]. Let  $\mathbf{V}$  be a unitary

<sup>1</sup> Although all the edges included in  $\mathbb{M}[\mathbf{w}_p]$  have nonzero weights after the update computation with weight vector  $\mathbf{w}_p$ , the weight vector changes after the updates. Therefore,  $\mathbb{M}[\mathbf{w}_p]$  can include edges whose weights are zero as a result of update computations.

matrix and  $\tilde{\mathbf{X}}$  be the transformed matrix of  $\mathbf{X}$ , matrix  $\tilde{\mathbf{X}}$  is represented as  $\mathbf{X} = \tilde{\mathbf{X}}\mathbf{V}$  by SVD. Similarly, let vector  $\tilde{\mathbf{x}}_u$  be the transformed vector of  $\mathbf{x}_u$ , vector  $\mathbf{x}_u$  is represented as  $\mathbf{x}_u = \tilde{\mathbf{x}}_u\mathbf{V}$ . If  $\tilde{\mathbf{r}}_p = \tilde{\mathbf{x}}_p - \mathbf{w}_p\tilde{\mathbf{X}}$ , the definitions of the upper and lower bounds are given as follows:

**Definition 4** (UPPER AND LOWER BOUNDS). *If  $\overline{K}[u|\mathbf{w}_p]$  and  $\underline{K}[u|\mathbf{w}_p]$  are the upper and lower bounds of  $K[u|\mathbf{w}_p]$ , respectively,  $\overline{K}[u|\mathbf{w}_p]$  and  $\underline{K}[u|\mathbf{w}_p]$  are given as follows:*

$$\overline{K}[u|\mathbf{w}_p] = \frac{1}{\lambda}w_p[u] + \frac{1}{2\lambda M} \{ \|\tilde{\mathbf{r}}_p\|_2^2 + M - \sum_{i=1}^m (\tilde{r}_p[i] - \tilde{x}_u[i])^2 \} \quad (13)$$

$$\underline{K}[u|\mathbf{w}_p] = \frac{1}{\lambda}w_p[u] - \frac{1}{2\lambda M} \{ \|\tilde{\mathbf{r}}_p\|_2^2 + M - \sum_{i=1}^m (\tilde{r}_p[i] + \tilde{x}_u[i])^2 \} \quad (14)$$

In Definition 4,  $\|\tilde{\mathbf{r}}_p\|_2$  is the  $L_2$ -norm of vector  $\tilde{\mathbf{r}}_p$ ,  $\tilde{r}_p[i]$  is the  $i$ -th element of  $\tilde{\mathbf{r}}_p$ , and  $m$  is the target rank of SVD. The following lemma shows that  $\overline{K}[u|\mathbf{w}_p]$  and  $\underline{K}[u|\mathbf{w}_p]$  give the upper and lower bounds, respectively:

**Lemma 6** (UPPER AND LOWER BOUNDS). *If we update the weight of the edge from node  $u$  to  $p$  for weight vector  $\mathbf{w}_p$ , we have  $\overline{K}[u|\mathbf{w}_p] \geq K[u|\mathbf{w}_p]$  and  $\underline{K}[u|\mathbf{w}_p] \leq K[u|\mathbf{w}_p]$  for the upper and lower bounds given in Definition 4.*

**Proof** Since  $\mathbf{V}\mathbf{V}^\top = \mathbf{I}$  where  $\mathbf{I}$  is the identity matrix,  $(\mathbf{x}_p - \mathbf{w}_p\mathbf{X})\mathbf{x}_u^\top = (\tilde{\mathbf{x}}_p - \mathbf{w}_p\tilde{\mathbf{X}})\mathbf{V}\mathbf{V}^\top\tilde{\mathbf{x}}_u^\top = \tilde{\mathbf{r}}_p\tilde{\mathbf{x}}_u^\top = \sum_{i=1}^M \tilde{r}_p[i]\tilde{x}_u[i]$ . Since  $\tilde{r}_p[i]\tilde{x}_u[i] = \frac{1}{2} \{ (\tilde{r}_p[i])^2 + (\tilde{x}_u[i])^2 - (\tilde{r}_p[i] - \tilde{x}_u[i])^2 \}$  holds,

$$(\mathbf{x}_p - \mathbf{w}_p\mathbf{X})\mathbf{x}_u^\top = \frac{1}{2} \sum_{i=1}^M \{ (\tilde{r}_p[i])^2 + (\tilde{x}_u[i])^2 - (\tilde{r}_p[i] - \tilde{x}_u[i])^2 \}$$

Note that  $\sum_{i=1}^M (\tilde{x}_u[i])^2 = M$  holds since SVD is orthogonal transformation and each vector is normalized as described in Section 3. Therefore, we have from Equation (6)

$$K[u|\mathbf{w}_p] = \frac{1}{\lambda}w_p[u] + \frac{1}{2\lambda M} \{ \|\tilde{\mathbf{r}}_p\|_2^2 + M - \sum_{i=1}^M (\tilde{r}_p[i] - \tilde{x}_u[i])^2 \}$$

Since  $(\tilde{r}_p[i] - \tilde{x}_u[i])^2 \geq 0$  holds, we have

$$K[u|\mathbf{w}_p] \leq \frac{1}{\lambda}w_p[u] + \frac{1}{2\lambda M} \{ \|\tilde{\mathbf{r}}_p\|_2^2 + M - \sum_{i=1}^m (\tilde{r}_p[i] - \tilde{x}_u[i])^2 \}$$

Therefore, we have  $\overline{K}[u|\mathbf{w}_p] \geq K[u|\mathbf{w}_p]$  from Equation (13). Similarly, we have  $\underline{K}[u|\mathbf{w}_p] \leq K[u|\mathbf{w}_p]$  since  $\tilde{r}_p[i]\tilde{x}_u[i] = \frac{1}{2} \{ -(\tilde{r}_p[i])^2 - (\tilde{x}_u[i])^2 + (\tilde{r}_p[i] + \tilde{x}_u[i])^2 \}$  holds.  $\square$

From the above proof, it is clear that errors of the upper and lower bounds are proportional to  $\sum_{i=m+1}^M (\tilde{r}_p[i] - \tilde{x}_u[i])^2$  and  $\sum_{i=m+1}^M (\tilde{r}_p[i] + \tilde{x}_u[i])^2$ , respectively. Although SVD gives the smallest squared error in approximating matrix  $\mathbf{X}$ , it is not guaranteed to give the tightest upper and lower bounds since vector  $\tilde{\mathbf{r}}_p$  is distorted by vector  $\mathbf{w}_p$  in the form of  $\tilde{\mathbf{r}}_p = \tilde{\mathbf{x}}_p - \mathbf{w}_p\tilde{\mathbf{X}}$ . However, it is expected that we can improve the quality of the upper and lower bounds by using SVD. This is because  $\tilde{r}_p[i]$  and  $\tilde{x}_u[i]$  are likely to take small absolute values as  $i$  increases.

It takes  $O(m)$  time to compute the bounds from Equation (13) and (14) if we have norm  $\|\tilde{\mathbf{r}}_p\|_2$ . Note that norm  $\|\tilde{\mathbf{r}}_p\|_2$  is computed in  $O(d_p m)$  time and used for all edges to node  $p$  once we compute the norm. In addition, we can efficiently compute the SVD of matrix  $\mathbf{X}$  in  $O(NM \log m)$  time by using an existing technique [12]. Note that we compute SVD only once in constructing the graph.

#### 4.3.2 Incremental Computation

As described in Section 4.2.3, we update weights by adding edges one by one to edge set  $\mathbb{U}$  to update from edge set  $\mathbb{A}$  by computing  $\mathbb{A} = \mathbb{M}[\mathbf{w}_p] \setminus \mathbb{U}$  and  $\mathbb{A} = \mathbb{C}[\mathbf{w}_p] \setminus \mathbb{U}$ . We can reduce the computation time to obtain edge set  $\mathbb{M}[\mathbf{w}_p]$  and

$\mathcal{C}[\mathbf{w}_p]$  by exploiting the SVD-based approach introduced in the previous section. However, the efficiency of the approach would only be moderate since it requires  $O(m)$  time to compute the bounds for each edge every time we compute the edge sets. How can we cut down the computation cost for determining the bounds? This is the motivation behind our approach of incremental computation; we incrementally compute the bounds at  $O(1)$  time. This approach is based on the approach that adds edges one by one as introduced in Section 4.2.3; we can effectively update the bounds since the weight vector is not so different before and after the edge additions. This section first describes the relation between KKT condition score and parameter  $z_p[u|\mathbf{w}_p]$ . Then, it shows the incremental approach to computing the bounds.

The following lemma confirms that we can compute the KKT condition score from parameter  $z_p[u|\mathbf{w}_p]$ :

**Lemma 7** (EQUIVALENCE OF KKT CONDITION SCORE). *For node  $u$  and weight vector  $\mathbf{w}_p$ , we can compute KKT condition score  $K[u|\mathbf{w}_p]$  from parameter  $z_p[u|\mathbf{w}_p]$  as follows:*

**Proof** From Equation (6), we have

$$\begin{aligned} K[u|\mathbf{w}_p] &= \frac{1}{\lambda} z_p[u|\mathbf{w}_p] \\ \lambda K[u|\mathbf{w}_p] &= w_p[u] + \frac{1}{M} (\mathbf{x}_p - \mathbf{w}_p \mathbf{X}) \mathbf{x}_u^\top \\ &= \frac{1}{M} (w_p[u]M + \langle \mathbf{x}_u, \mathbf{x}_p \rangle - \langle \mathbf{x}_u, \mathbf{w}_p \mathbf{X} \rangle) \end{aligned} \quad (15)$$

where  $\langle \mathbf{x}_p, \mathbf{x}_u \rangle$  is the inner product of vector  $\mathbf{x}_p$  and  $\mathbf{x}_u$ . Therefore, we have

$$\begin{aligned} \lambda K[u|\mathbf{w}_p] &= \frac{1}{M} \{ w_p[u]M + \sum_{i=1}^M (x_u[i]x_p[i] - x_u[i] \sum_{v \in \mathbb{V}} w_p[v]x_v[i]) \} \\ &= \frac{1}{M} \{ w_p[u]M + \sum_{i=1}^M x_u[i] (x_p[i] - \sum_{v \in \mathbb{V}} w_p[v]x_v[i]) \} \end{aligned}$$

Since  $\sum_{i=1}^M (x_u[i])^2 = M$  and  $w_p[p] = 0$  as described in Section 3, we have the following equation from Equation (5):

$$\begin{aligned} \lambda K[u|\mathbf{w}_p] &= \frac{1}{M} \{ \sum_{i=1}^M x_u[i] (w_p[u]x_u[i] + x_p[i] - \sum_{v \in \mathbb{V}} w_p[v]x_v[i]) \} \\ &= \frac{1}{M} \{ \sum_{i=1}^M x_u[i] (x_p[i] - \sum_{v \in \mathbb{V} \setminus \{p, u\}} w_p[v]x_v[i]) \} \\ &= \frac{1}{M} \sum_{i=1}^M x_u[i] (x_p[i] - \tilde{x}_p^{(u)}[i]) \end{aligned}$$

Therefore,  $\lambda K[u|\mathbf{w}_p] = z_p[u|\mathbf{w}_p]$  holds from Equation (4).  $\square$

Lemma 7 indicates that we can compute KKT condition score in  $O(1)$  time if (1) an edge is included in the edge set and (2) we have parameter  $z_p[u|\mathbf{w}_p]$  of the edge. We incrementally compute the upper and lower bounds after edge addition as follows:

**Definition 5** (INCREMENTAL COMPUTATION). *Let  $\nu$  be the number of edges in edge set  $\mathbb{A}$  that are added to edge set  $\mathbb{U}$  and  $\mathbf{w}_p^i = (w_p^i[1], w_p^i[2], \dots, w_p^i[n])$  be the converged weight vector after the  $i$ -th edge addition ( $1 \leq i \leq \nu$ ). In addition, let  $\mathbf{r}_p^i = (r_p^i[1], r_p^i[2], \dots, r_p^i[n])$  be a residual vector that corresponds to weight vector  $\mathbf{w}_p^i$  given as follows:*

$$\mathbf{r}_p^i = \mathbf{x}_p - \mathbf{w}_p^i \mathbf{X} \quad (16)$$

Let  $\delta_p^i$  be the score that corresponds to the difference of KKT condition score before and after the  $i$ -th edge addition:

$$\delta_p^i = \frac{1}{\lambda} \{ \|\mathbf{r}_p^i - \mathbf{r}_p^{i-1}\|_2^2 + \|\mathbf{w}_p^i - \mathbf{w}_p^{i-1}\|_1 \} \quad (17)$$

We compute the upper and lower bounds after the iterations based on Lemma 7 as follows if an edge is included in edge set  $\mathbb{A}$  and added to edge set  $\mathbb{U}$  in the  $\kappa$ -th addition ( $1 \leq \kappa \leq \nu$ ):

$$\overline{K}[u|\mathbf{w}_p^\nu] = \frac{1}{\lambda} z_p[u|\mathbf{w}_p^\kappa] + \Delta_p \quad (18)$$

$$\underline{K}[u|\mathbf{w}_p^\nu] = \frac{1}{\lambda} z_p[u|\mathbf{w}_p^\kappa] - \Delta_p \quad (19)$$

In Equation (18) and (19),  $\Delta_p = \sum_{i=1}^\kappa \delta_p^i$ . If the edge is not included in edge set  $\mathbb{A}$ , the bounds are computed as follows:

$$\overline{K}[u|\mathbf{w}_p^\nu] = \overline{K}[u|\mathbf{w}_p^0] + \Delta_p \quad (20)$$

$$\underline{K}[u|\mathbf{w}_p^\nu] = \underline{K}[u|\mathbf{w}_p^0] - \Delta_p \quad (21)$$

In Definition 5,  $\overline{K}[u|\mathbf{w}_p^0]$  and  $\underline{K}[u|\mathbf{w}_p^0]$  correspond to the bounds before the edge additions;  $\overline{K}[u|\mathbf{w}_p^\nu]$  and  $\underline{K}[u|\mathbf{w}_p^\nu]$  correspond to the bounds after the edge additions. Therefore, we can incrementally update the bounds by using  $z_p[u|\mathbf{w}_p^\kappa]$ ,  $\overline{K}[u|\mathbf{w}_p^0]$ , and  $\underline{K}[u|\mathbf{w}_p^0]$  from Definition 5. We have the following property for the bounds:

**Lemma 8** (INCREMENTAL COMPUTATION). *After the  $\nu$ -th edge addition, we have  $\overline{K}[u|\mathbf{w}_p^\nu] \geq K[u|\mathbf{w}_p^\nu]$  and  $\underline{K}[u|\mathbf{w}_p^\nu] \leq K[u|\mathbf{w}_p^\nu]$  for the bounds given by Definition 5 if the weights of edges are updated for weight vector  $\mathbf{w}_p^\nu$ .*

**Proof** From Equation (6) and (16), we have the following equation from the Cauchy-Schwarz inequality [22] for the  $i$ -th edge addition ( $1 \leq i \leq \nu$ ):

$$\begin{aligned} |K[u|\mathbf{w}_p^i] - K[u|\mathbf{w}_p^{i-1}]| &= \frac{1}{\lambda} \frac{1}{M} (\mathbf{r}_p^i - \mathbf{r}_p^{i-1}) \mathbf{x}_u^\top + (w_p^i[u] - w_p^{i-1}[u]) \\ &\leq \frac{1}{\lambda} \left( \frac{1}{M} \|\mathbf{r}_p^i - \mathbf{r}_p^{i-1}\|_2^2 \|\mathbf{x}_u^\top\|_2^2 + \|\mathbf{w}_p^i - \mathbf{w}_p^{i-1}\|_1 \right) = \frac{1}{\lambda} \{ \|\mathbf{r}_p^i - \mathbf{r}_p^{i-1}\|_2^2 + \|\mathbf{w}_p^i - \mathbf{w}_p^{i-1}\|_1 \} \end{aligned}$$

This is because we have  $\|\mathbf{x}_u^\top\|_2^2 = M$ . Therefore, from Equation (17),  $|K[u|\mathbf{w}_p^i] - K[u|\mathbf{w}_p^{i-1}]| \leq \delta_p^i$ . As a result, for an edge included in the edge set,

$$\begin{aligned} |K[u|\mathbf{w}_p^\nu] - K[u|\mathbf{w}_p^0]| &= \left| \sum_{i=\kappa+1}^\nu (K[u|\mathbf{w}_p^i] - K[u|\mathbf{w}_p^{i-1}]) \right| \\ &\leq \left| \sum_{i=\kappa+1}^\nu \delta_p^i \right| \leq \left| \sum_{i=1}^\nu \delta_p^i \right| = \Delta_p \end{aligned}$$

Thus, from Lemma 7, we have

$$\frac{1}{\lambda} z_p[u|\mathbf{w}_p^\kappa] - \Delta_p \leq K[u|\mathbf{w}_p^\nu] \leq \frac{1}{\lambda} z_p[u|\mathbf{w}_p^\kappa] + \Delta_p$$

Similarly, for an edge that is not included in the edge set,

$$\underline{K}[u|\mathbf{w}_p^0] - \Delta_p \leq K[u|\mathbf{w}_p^\nu] \leq \overline{K}[u|\mathbf{w}_p^0] + \Delta_p$$

which completes the proof.  $\square$

As described in Section 4.2.2, we compute the residuals for the added edges every time the edges change the regression result to efficiently update the weights. Since the coordinate descent updates weights one at a time, we can incrementally compute  $\delta_p^i$  and  $\Delta_p$  at  $O(1)$  time from Equation (17). As a result, we can incrementally compute the bounds in  $O(1)$  time by exploiting Definition 5.

In terms of computing the upper and lower bounds, the incremental approach of Definition 5 is the same as the SVD-based approach of Definition 4. However, they yield different bounds since they adopt totally different processes to compute the bounds. In addition, the incremental approach can compute the bounds more efficiently. Therefore, we first exploit the incremental approach to obtain the bounds for each edge. Then, we use the SVD-based approach only for edges that are determined to update weights by the incremental approach as described in the next section.

## 4.4 Graph Construction Algorithm

Algorithm 1 gives a full description of Castnet, which efficiently constructs the lasso-based  $L_1$ -graph. In Algorithm 1,  $\mathbf{W}$  is an  $N \times N$  matrix whose  $p$ -th row corresponds to weight vector  $\mathbf{w}_p$ , and  $\mathbb{P}$  is the set of nodes selected for weight computation. Castnet initializes weights based on the observation that the weight of an edge from node  $u$  to  $v$  is similar to that from node  $v$  to  $u$ ;  $w_v[u] \approx w_u[v]$  [4].

---

**Algorithm 1** Castnet

---

**Input:** matrix  $\mathbf{X}$ , tuning parameter  $\lambda$ , target rank of SVD  $m$   
**Output:** matrix  $\mathbf{W}$

```
1:  $\mathbb{P} = \emptyset$ ;  
2: compute rank- $m$  SVD of matrix  $\mathbf{X}$ ;  
3: for  $i = 1$  to  $N$  do  
4:    $p = \operatorname{argmax}(\|\mathbf{w}_u\|_1 | u \in \mathbb{V} \setminus \mathbb{P})$ ;  
5:    $\mathbb{U} = \{e_p[u] | w_p[u] \neq 0\}$ ;  
6:   update weights for  $\mathbb{U}$  by Equation (12);  
7:   for  $j = 1$  to  $N$  do  
8:      $\overline{K}[u_j|\mathbf{w}_p] = -\infty$ ,  $\underline{K}[u_j|\mathbf{w}_p] = \infty$ ;  
9:    $step = 1$ ;  
10:  while  $step \leq 2$  do  
11:    for each  $u \in \mathbb{V} \setminus p$  do  
12:      compute the bounds by Definition 5;  
13:    if  $step = 1$  then  
14:       $\mathbb{M}[\mathbf{w}_p] = \emptyset$ ;  
15:      for each  $u \in \mathbb{V} \setminus p$  do  
16:        if  $\overline{K}[u|\mathbf{w}_p] < -1$  or  $\underline{K}[u|\mathbf{w}_p] > 1$  then  
17:          compute the bounds of node  $u$  by Definition 4;  
18:          if  $\overline{K}[u|\mathbf{w}_p] < -1$  or  $\underline{K}[u|\mathbf{w}_p] > 1$  then  
19:            add edge  $e_p[u]$  to  $\mathbb{M}[\mathbf{w}_p]$ ;  
20:         $\mathbb{A} = \mathbb{M}[\mathbf{w}_p] \setminus \mathbb{U}$ ;  
21:      else  
22:         $\mathbb{C}[\mathbf{w}_p] = \emptyset$ ;  
23:        for each  $u \in \mathbb{V} \setminus p$  do  
24:          if  $\overline{K}[u|\mathbf{w}_p] > 1$  or  $\underline{K}[u|\mathbf{w}_p] < -1$  then  
25:            compute the bounds of node  $u$  by Definition 4;  
26:            if  $\overline{K}[u|\mathbf{w}_p] > 1$  or  $\underline{K}[u|\mathbf{w}_p] < -1$  then  
27:              add edge  $e_p[u]$  to  $\mathbb{C}[\mathbf{w}_p]$ ;  
28:           $\mathbb{A} = \mathbb{C}[\mathbf{w}_p] \setminus \mathbb{U}$ ;  
29:        for each  $u \in \mathbb{A}$  do  
30:          compute  $w_p[u]$  for weight vector  $\mathbf{w}_p$  by Equation (10);  
31:          if  $w_p[u] \neq 0$  then  
32:            add edge  $e_p[u]$  to edge set  $\mathbb{U}$ ;  
33:          update weights for  $\mathbb{U}$  by Equation (12);  
34:        if  $\mathbb{A} = \emptyset$  then  
35:           $step ++$ ;  
36:      add node  $p$  to  $\mathbb{P}$ ;  
37:      for each  $u \in \mathbb{V} \setminus \mathbb{P}$  do  
38:         $w_u[p] = w_p[u]$ ;  
return weights of the graph;
```

---

Algorithm 1 starts by initializing node set  $\mathbb{P}$  and computing SVD of data matrix  $\mathbf{X}$  to allow the upper and lower bounds to be determined in the iterations (lines 1-2). It picks up the node that has the maximum  $L_1$  norm for the weight vector since we can accurately perform the regression for the node from the observation (line 4). It then performs update computations for edges that have nonzero weights by using Equation (12) and initializes the bounds (lines 5-8). Our approach has two steps in computing the weights of selected nodes; the first step is based on edge set  $\mathbb{M}[\mathbf{w}_p]$  and the second step exploits edge set  $\mathbb{C}[\mathbf{w}_p]$ . In order to prune unnecessary updates, it computes the bounds determined by the incremental approach as described in Section 4.3.2 (lines 11-12) and checks the conditions of edge set  $\mathbb{M}[\mathbf{w}_p]$  and  $\mathbb{C}[\mathbf{w}_p]$  (line 16 and line 24). If an edge can meet the conditions of the edge sets (Definition 2 and 3), it computes the bounds again by using the SVD-based approach to determine the edge sets (lines 17-19 and lines 25-27). As described in Section 4.2.3, it adds edges one by one to edge set  $\mathbb{U}$  so as to increase the efficiency. Therefore, it computes edge set  $\mathbb{A}$  after determining edge sets  $\mathbb{M}[\mathbf{w}_p]$  and  $\mathbb{C}[\mathbf{w}_p]$  (line 20 and line 28). If the weight of an edge is zero, we can efficiently compute its weight by using Equation (10) (line 30); such edges have no impact on the regression result as shown in Lemma 5. Therefore, it updates weights until convergence only if the added edge has nonzero weight (lines 31-33). The weight vector clearly reaches convergence if we

have no edge to add. Therefore, it proceeds the step of constructing the graph if we have  $\mathbb{A} = \emptyset$  (lines 34-35). After the iterations, it sets the weights of edges to unselected nodes based on the observation (lines 37-38).

In Algorithm 1, we implicitly assumed a single thread where we pick up nodes one by one from the entire set of nodes. However, Algorithm 1 is easily parallelized; we divide the set of nodes into several sets and parallelly perform Algorithm 1 for each obtained set. Since we compute edges from all the nodes for each obtained set, we can obtain the same graph as the single-thread approach even if we use the parallelization approach. We use the k-means clustering to effectively divide the set according to the hidden structure of data points. Although the k-means clustering performed in a single thread, we can efficiently divide the entire set of nodes since (1) we use SVD to reduce the number of dimensions and (2) we apply k-means++ to increase the processing speed of the k-means clustering [1].

For Algorithm 1, we have the following properties:

**Theorem 1** (COMPUTATION COST). *Let  $t$  be the number of update computations in our approach, Castnet takes  $O(d_p t + M(N + \log m + t + (d_p)^2))$  time to compute weights of a selected node in the lasso-based  $L_1$ -graph.*

**Proof** Castnet first computes the SVD of matrix  $\mathbf{X}$  used in computing the bounds at  $O(M \log m)$  time amortized on the number of nodes [12]. In the SVD-based bound computations, it takes  $O(d_p m)$  time to compute norm  $\|\tilde{\mathbf{r}}_p\|_2$  and  $O(Nm)$  time to compute the bounds. Similarly, we need  $O(N)$  time to compute the bounds by the incremental computation approach. In addition, we can compute edge sets  $\mathbb{M}[\mathbf{w}_p]$ ,  $\mathbb{C}[\mathbf{w}_p]$ , and  $\mathbb{A}$  in  $O(N)$  time. In performing update computations for edges whose weights are zero, it takes  $O(Mt)$  time to compute the residuals and  $O(M(N - d_p))$  time to update computations. In updating nonzero weights, it needs  $O(M(d_p)^2)$  time to compute the inner products and  $O(d_p t)$  time to update weights. Therefore, Castnet needs  $O(d_p t + M(N + \log m + t + (d_p)^2))$  time.  $\square$

In practice, the number of update computations  $t$  increases with the number of nodes  $N$ . However, we can reduce  $t$  if a node has small degree  $d_p$ . We show these relationships in Section 5.

**Theorem 2** (CONSTRUCTION RESULT). *If matrix  $\mathbf{X}$  has full row rank, the proposed approach converges on the same regression result as the original coordinate descent approach in computing edge weights for a selected node.*

**Proof** As shown in Algorithm 1, Castnet first update weights if  $\overline{K}[u|\mathbf{w}_p] < -1$  or  $\underline{K}[u|\mathbf{w}_p] > 1$  holds. It then update weights if  $\overline{K}[u|\mathbf{w}_p] > 1$  or  $\underline{K}[u|\mathbf{w}_p] < -1$  hold. As shown in Lemma 3, we have  $\overline{K}[u|\mathbf{w}_p] > 1$  or  $\underline{K}[u|\mathbf{w}_p] < -1$  if  $\overline{K}[u|\mathbf{w}_p] < -1$  or  $\underline{K}[u|\mathbf{w}_p] > 1$  holds. Therefore, these two processes of the proposed approach indicate that we do not perform update computations for weights of edges such that  $\overline{K}[u|\mathbf{w}_p] \leq 1$  and  $\underline{K}[u|\mathbf{w}_p] \geq -1$ . It is clear that such edges cannot have nonzero weights from the KKT condition (Definition 1). As a result, our approach cannot prune edges that have nonzero weights. Since there is a unique regression result if matrix  $\mathbf{X}$  has full row rank, the coordinate descent approach converges to the unique result [6, 24]. Since our approach is based on the coordinate descent, it is clear that Castnet converges the same result as the original coordinate descent approach if matrix  $\mathbf{X}$  has full row rank.  $\square$



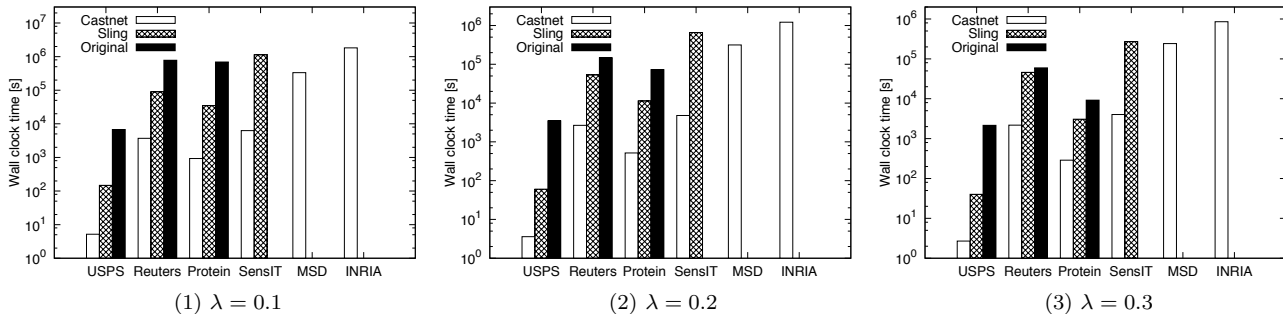


Figure 1: Graph construction time of each approach.

If matrix  $\mathbf{X}$  does not have full row rank, which is necessarily the case when  $N > M$ , there may not be a unique regression result. In this case, our approach yields almost the same result as the original coordinate descent approach. In the next section, we detail experiments that show the efficiency and effectiveness of the proposed approach.

## 5. EXPERIMENTAL EVALUATION

We performed experiments to demonstrate the effectiveness of our approach. In the experiments, we used the six datasets taken from various domains: *USPS*, *Reuters*, *Protein*, *SensIT*, *MSD*, and *INRIA*. *USPS* is a popular dataset of handwritten digits captured from envelopes by the U.S. Postal Service<sup>2</sup>. This dataset contains 2,007 grayscale images, each  $16 \times 16$  pixels; the number of features is 256. *Reuters* is a corpus of newswire stories that contains 8,293 documents<sup>3</sup>. In this dataset, tf-idf is used as the document feature; it has 18,933 dimensions. *Protein* is a set of amino acid sequences where the number of data and features are 17,766 and 357, respectively<sup>4</sup>. *SensIT* is a dataset obtained from a distributed wireless sensor network for vehicle type classification<sup>5</sup>. In this dataset, the number of data and features are 78,823 and 100, respectively. *MSD* is a music dataset that contain 515,345 songs of 90 features<sup>6</sup>. In this dataset, songs are mostly western and commercial tracks. *INRIA* consists of 1,000,000 image features extracted from both personal and Flickr photos, which of each is represented by a 128-D SIFT descriptor<sup>7</sup>. Since we have  $N > M$ , matrix  $\mathbf{X}$  is not full row rank except for *Reuters* while matrix  $\mathbf{X}$  is full row rank for the dataset. The numbers of nodes in the datasets increase in the order of *USPS*, *Reuters*, *Protein*, *SensIT*, *MSD*, and *INRIA*.

In what follows, “Castnet”, “Sling”, and “Original” represents our approach, the state-of-the-art approach for the lasso [7], and the original coordinate descent approach [5], respectively. We set the target rank of SVD to 10 for the proposed approach. All the experiments were conducted on a Linux 2.70 GHz Intel Xeon server with 1 TB of main memory. We implemented all the approaches using GCC.

### 5.1 Graph Construction Time

We assessed the graph construction time needed for each approach. Figure 1 shows the results where we set tuning

<sup>2</sup> <https://www.otexts.org/1577>

<sup>3</sup> <http://www.cad.zju.edu.cn/home/dengcai/Data/data.html>

<sup>4</sup> <http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/>

<sup>5</sup> <http://mldata.org/repository/data/>

<sup>6</sup> <http://labrosa.ee.columbia.edu/millionsong/>

<sup>7</sup> <http://corpus-texmex.irisa.fr/>

Table 2: Degree and updates in *INRIA*.

	$\lambda$	$N = 250,000$	$N = 500,000$	$N = 1,000,000$
Degree	0.1	16.77	15.02	15.32
	0.2	9.91	9.34	9.59
	0.3	6.76	6.82	7.17
Updates	0.1	$5.44 \times 10^5$	$1.03 \times 10^6$	$2.04 \times 10^6$
	0.2	$4.82 \times 10^5$	$9.27 \times 10^5$	$1.82 \times 10^6$
	0.3	$4.22 \times 10^5$	$7.56 \times 10^5$	$1.53 \times 10^6$

parameter  $\lambda$  to 0.1, 0.2, and 0.3. In Figure 1, we omit the result of Sling for *MSD* and *INRIA* datasets since it could not construct the  $L_1$ -graph within a month. Similarly, we omit the results of the original approach for *SensIT*, *MSD*, and *INRIA* datasets for the same reason. Table 2 shows the average degree and update computations of each node; *INRIA* was used as the dataset and we changed the number of nodes in the graph. To evaluate the scalability offered by our parallelization approach, we plot speedup over single thread vs. the number of threads for each graph in Figure 2. Figure 3 shows the computation overhead of k-means clustering in the parallelization approach.

Figure 1 shows that our approach is much faster than the previous approaches. Specifically, Castnet is up to 1,300 times faster than the original approach. In order to obtain edge weights, the original approach requires a different regression result for each weight as described in Section 3. Therefore, the original approach must compute the regression result every time it updates each weight. In addition, the original approach iteratively updates weights until convergence even though almost all edges have zero weights due to the sparse structure of the  $L_1$ -graph, which explains its high computation cost. To improve the computation speed for the lasso, Sling updates weights by using the covariance-based approach as described in Section 2. Since the covariance-based approach does not need the different regression results, Sling can more efficiently update weights in each iteration than the original approach. However, the covariance-based approach needs inner product computations for all edges to update the weights before entering the iterations. On the other hand, our approach can avoid the heavy computations of inner products by using the two types of update computations as described in Section 4.2.2. In addition, our approach selectively updates weights by computing edge sets from upper and lower bounds of KKT condition scores as described in Section 4.2.1. As a result, our approach is up to 130 times faster than the state-of-the-art approach. As shown in Figure 1, computation time of our approach increases with dataset size. This is because we need more update computations as the number of nodes increases as

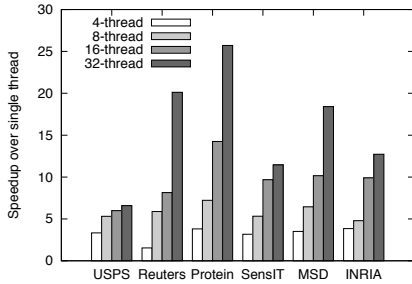


Figure 2: Effect of parallelization approach.

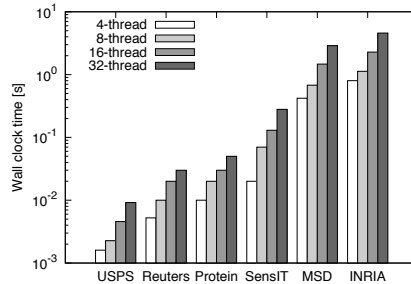


Figure 3: Overhead with parallelization.

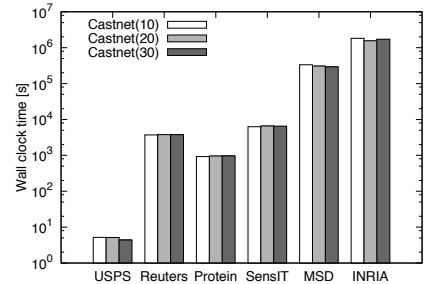


Figure 4: Effect of target rank number.

Table 3: Number of update computations.

Dataset	$m = 10$	$m = 20$	$m = 30$
USPS	$2.362 \times 10^7$	$2.296 \times 10^7$	$1.833 \times 10^7$
Reuters	$1.471 \times 10^8$	$1.470 \times 10^8$	$1.469 \times 10^8$
Protein	$1.400 \times 10^9$	$1.380 \times 10^9$	$1.372 \times 10^9$
SensIT	$1.373 \times 10^{10}$	$1.341 \times 10^{10}$	$1.092 \times 10^{10}$
MSD	$4.258 \times 10^{11}$	$1.975 \times 10^{11}$	$7.196 \times 10^{10}$
INRIA	$2.038 \times 10^{12}$	$2.029 \times 10^{12}$	$1.992 \times 10^{12}$

shown in Table 2. On the other hand, Figure 1 indicates that we can increase the efficiency by increasing tuning parameter  $\lambda$ . This is because, as shown in Table 2, the number of update computations decreases if we reduce the degree of each node by assigning large values to the tuning parameter. Since our approach can effectively prune unnecessary computations, it can more efficiently construct the lasso-based  $L_1$ -graph than the previous approaches.

As described in Section 4.4, Castnet can be parallelized by using multiple threads. As shown in Figure 2, our parallelization approach is up to 25 times faster than the single-thread approach with 32-thread execution. Although our parallelization approach needs the additional process of k-means clustering in a single thread, its computation overhead has negligible impact on the graph construction time. This is because we can efficiently perform k-means clustering by using SVD and k-means++ as shown in Figure 3; it takes at most 5 seconds to perform the clustering for the largest dataset. Therefore, we can effectively increase the processing speed by exploiting the parallelization approach.

## 5.2 Efficiency vs. Target Rank Number

As described in Section 4.3, our approach computes SVD of rank  $m$  for the data matrix to increase the computation speed of the upper and lower bounds. Since the bounds are used in computing the edge sets for updating weights, the target rank of SVD is expected to impact the graph construction time. In this section, we show the graph construction times recorded for different target rank values of SVD. Figure 4 shows the results where we set  $\lambda = 0.1$  as the tuning parameter of the lasso. In this figure, the results of our approach are indicated by “Castnet( $m$ )” where  $m$  is the target rank of SVD. In addition, Table 3 shows the number of update computations for all the nodes needed by our approach to construct the  $L_1$ -graph when  $\lambda = 0.1$ .

As shown in Figure 4, the graph construction time varies only slightly against the target rank of SVD for the data matrix; our approach is not sensitive to rank  $m$  in terms of efficiency. As described in Section 4.3, it takes  $O(m)$  time to compute the upper and lower bounds by SVD. Therefore,

Table 4: Regression results of each approach.

Dataset	Approach	Objective function	Squared loss	$L_1$ -norm constraint
USPS	Castnet	0.1355	0.0348	0.1007
	Sling	0.1355	0.0348	0.1007
	Original	0.1355	0.0348	0.1007
Reuters	Castnet	0.3985	0.3360	0.0625
	Sling	0.3985	0.3360	0.0625
	Original	0.3985	0.3360	0.0625
Protein	Castnet	0.3323	0.1651	0.1672
	Sling	0.3323	0.1651	0.1672
	Original	0.3323	0.1651	0.1672

we can increase the efficiency of computing the bounds if rank  $m$  has a small score. On the other hand, as shown in Table 3, we can reduce the number of update computations if rank  $m$  has a large score. This is because we can improve the accuracy of SVD in approximating the data matrix by increasing the target rank. In conclusion, we can efficiently compute the bounds by reducing the target rank at the sacrifice of the number of update computations whereas we can efficiently update weights with the large target rank that incurs high computation cost for the bounds. Due to the trade-off derived from rank  $m$ , our approach is not sensitive to the target rank of SVD.

## 5.3 Effectiveness of the Graph Structure

As described in Section 4.4, one major advantage of our approach is that it yields the same  $L_1$ -graph as the original approach if matrix  $\mathbf{X}$  has full row rank. However, our approach can output different results from the original approach if matrix  $\mathbf{X}$  does not have full row rank since there may not be a unique lasso solution in terms of the objective function introduced in Section 3 (Equation (1)). In this section, we evaluate the effectiveness of the regression result in constructing the  $L_1$ -graph of each approach. Table 4 shows the averages of the objective function in computing edge weights for each node in the  $L_1$ -graph when  $\lambda = 0.1$ . In the table, “Squared loss” and “ $L_1$ -norm constraint” correspond to the first and second terms of the objective function, respectively. Note that the datasets used in this experiment do not have full row rank due to  $N > M$  except for Reuters dataset. In addition, the same as our approach, Sling can output different regression results from the original approach if matrix  $\mathbf{X}$  does not have full row rank [7].

Table 4 indicates that our approach yields the same objective function scores as the original approach for the Reuters dataset, which does have full row rank. This is because our approach provably guarantees to output the same solution

**Table 5: Label propagation.**

	Castnet	$L_1$	$k$ -NN	LN
Precision	0.581	0.576	0.433	0.554
Graph Construction [s]	19.19	78.17	3.35	7.19
Number of edges	34,804	39,329	36,926	30,426

for the  $L_1$ -optimization problem as described in Theorem 2. Although our approach can yield different regression results if matrix  $\mathbf{X}$  does not have full row rank, the graph structures obtained by our approach were almost the same as those by the original approach. This is because we perform update computations for edges that must/can have nonzero weights by computing the upper and lower bounds of KKT condition scores as shown in Algorithm 1. Since KKT condition scores correspond to gradients in coordinate descent [5], our approach can effectively identify edges that effectively improve the objective functions by obtaining the edge set that must/can have nonzero weights. Therefore, our approach can effectively compute the solution for the optimization problem of the lasso by using the upper and lower bounds. As a result, scores of the objective function of our approach and the original approach are the same in practice for USPS and Protein datasets as shown in Table 4 even though these datasets do not have full row rank. Along with Figure 1, Table 4 indicates that our approach significantly improves the efficiency to construct the  $L_1$ -graph while the quality of graph structure is as high as the previous approaches.






















































## 6. CASE STUDIES

In order to effectively find the hidden structure of data points, it is important to construct a graph that is robust against the noise. Our approach allows users to efficiently construct the lasso-based  $L_1$ -graph so as to enhance the quality of applications more than is possible with traditional  $k$ -NN graphs. This section compares our approach to a  $k$ -NN graph as well as the  $L_1$ -graph proposed by Cheng et al. [2] and the linear neighborhood graph proposed by Wang et al. [25]. The  $L_1$ -graph by Cheng et al. uses only the  $L_1$ -norm constraint as the objective function while our approach uses the sum of squared loss and  $L_1$ -norm constraint as described in Section 3. The linear neighborhood graph by Wang et al. computes the edge weight of each node as a linear combination of its neighbors.

We used the COIL-100 dataset that contains images of 100 objects<sup>8</sup>. The objects were placed on a turntable that was rotated through 360 degrees to vary object pose with respect to a fixed camera. Images of the object were taken at pose intervals of 5 degrees; 72 poses per object, resulting in 7,200 images. We used bag of keypoints of 128 dimensions as image features where we exploited SIFT descriptor to computer feature vector [3]. Since we have  $N > M$ , data matrix  $\mathbf{X}$  does not have full row rank for this dataset. In order to evaluate the effectiveness, we used label propagation [9]; it estimates labels of unlabeled data points from labeled data points. To this end, it propagates labels to the unlabeled nodes in the given graph from a labeled seed node. We randomly selected a data point from each object as the seed node. Table 5 shows labeling accuracy and graph construction time of each approach; labeling accuracy is ratio of labeled nodes that correspond to the same object as the seed node. Table 5 also shows the number of edges obtained

<sup>8</sup> <http://www1.cs.columbia.edu/CAVE/software/>

**Table 6: Connected nodes by each approach.**

Approach	Connected nodes					
						
Castnet	 0.610	 0.512	 0.653	 0.161	 0.609	 0.353
	 0.394	 0.008	 0.014	 0.002	 0.061	 0.060
$L_1$	 0.422	 0.350	 0.330	 0.273	 0.418	 0.279
	 0.313	 0.002	 0.270	 0.058	 0.255	 0.070
$k$ -NN	 0.295	 0.282	 0.252	 0.222	 0.221	 0.206
	 0.254	 0.184	 0.220	 0.219	 0.199	 0.190
LN	 0.350	 0.339	 0.374	 0.277	 0.353	 0.299
	 0.305	 0.005	 0.193		 0.194	 0.167

by each approach. In addition, Table 6 shows examples of connected nodes and their edge weights obtained by each approach. In Table 5 and 6, “ $L_1$ ” and “LN” correspond to the results of the  $L_1$ -graph by Cheng et al. [2] and the linear neighborhood graph by Wang et al. [25]. In constructing  $k$ -NN graph, we used FLANN that approximately finds the nearest neighbors; it is a popular approach used in OpenCV [18]. We set parameters of FLANN so that it yields accuracy of 90% in computing the nearest neighbors since this setting is well used in OpenCV. Note that FLANN outperforms LSH-based approaches in terms of efficiency. We set  $\lambda = 0.4$  to construct the  $L_1$ -graph and computed the top four nodes to obtain edges in the  $k$ -NN graph. As a result, each approach yielded similar number of edges as shown in Table 5. For the  $k$ -NN graph, we computed edge weights from normalized graph Laplacians [9]. Note that our approach yields the same labeling results as the original coordinate descent approach.

Table 5 indicates that our approach can more effectively identify semantically equivalent nodes than other graphs. In addition, as shown in Table 6, our approach successfully connected semantically equivalent nodes to the picked up nodes. This is because, by effectively using the regression approach, the loss-based  $L_1$ -graph can connect semantically equivalent nodes. Therefore, semantically equivalent nodes comprise a cluster structure in the high dimensional feature spaces. In terms of constructing the  $L_1$ -graph, the approach by Cheng et al. is similar to our approach. However, since their approach does not use the squared loss as the objective function, it is not robust against noise as described in

the previous study [4]. This indicates that our approach can improve the robustness against noise by using the squared loss in the objective function. For example, in the third case of Table 6 where a white car is a node, both our and their approaches connected a green car to the node. However, our approach assigns much lower edge weight to the semantically different node than semantically equivalent nodes. On the other hand, their approach assigns high edge weight to the semantically different node. In addition, Table 5 indicates that our approach is more efficient than their approach in constructing the  $L_1$ -graph. This is because our approach skips unnecessary weight updates. In terms of efficiency,  $k$ -NN graph can be constructed more efficiently than the lasso-based  $L_1$ -graph as shown in Table 5. However,  $k$ -NN graph as well as linear neighborhood graph offers lower labeling accuracy than our approach. This is because, as shown in Table 6, they are not so effective as our approach in connecting semantically equivalent nodes. For example, in the first case where a white cat is a node,  $k$ -NN graph connects red doll to the node. In addition, they give high weights to semantically different nodes as shown in Table 6.

Although the traditional  $k$ -NN graph is the most popular approach, it is sensitive to data noise as shown in Table 5. Effectiveness is quite important in many applications such as lesion detection as described in Section 1.1; we can reduce deaths caused by prostate cancer by effectively performing lesion detection. The lasso base  $L_1$ -graph was proposed to improve the sensitive to data noise. Although it overcomes the drawback, the original coordinate descent approach does not scale well to handle large data sets as shown in Figure 1. The contribution of our approach is to significantly increase the efficiency of constructing the  $L_1$ -graphs without sacrificing the usefulness of the graph structure. As a result, the proposed approach is an attractive option for the research community in constructing the lasso-based  $L_1$ -graphs.

## 7. CONCLUSIONS

We addressed the problem of efficiently constructing the lasso-based  $L_1$ -graph. Our approach, Castnet, limits update computations to the edges that must/can have nonzero weights before entering the iterations by computing the upper and lower bounds of KKT condition scores. In addition, our approach efficiently updates nonzero weights in each iteration by pruning edges whose weights are zero. Experiments show that our approach can achieve higher efficiency than existing approaches. Since the  $L_1$ -graph can effectively capture the cluster structures that share useful semantic information, it plays a fundamental role in many applications such as brain analysis, motion segmentation, and lesion detection. Our approach allows many applications to be implemented more efficiently, and will help to improve the effectiveness of future data mining applications.

## 8. REFERENCES

- [1] D. Arthur and S. Vassilvitskii.  $k$ -means++: The Advantages of Careful Seeding. In *SODA*, pages 1027–1035, 2007.
- [2] B. Cheng, J. Yang, S. Yan, Y. Fu, and T. S. Huang. Learning with  $l^1$ -graph for Image Analysis. *IEEE Trans. Image Processing*, 19(4):858–866, 2010.
- [3] C. Dance, J. Willamowski, L. Fan, C. Bray, and G. Csorika. Visual Categorization with Bags of Keypoints. In *ECCV International Workshop on Statistical Learning in Computer Vision*, 2004.
- [4] E. Elhamifar and R. Vidal. Sparse Subspace Clustering. In *CVPR*, pages 2790–2797, 2009.
- [5] J. Friedman, T. Hastie, H. Höfling, and R. Tibshirani. Pathwise Coordinate Optimization. *Annals of Applied Statistics*, 1(2):302–332, 2007.
- [6] J. H. Friedman, T. Hastie, and R. Tibshirani. Regularization Paths for Generalized Linear Models via Coordinate Descent. *Journal of Statistical Software*, 33(1):1–22, 2 2010.
- [7] Y. Fujiwara, Y. Ida, H. Shiokawa, and S. Iwamura. Fast Lasso Algorithm via Selective Coordinate Descent. In *AAAI*, pages 1561–1567, 2016.
- [8] Y. Fujiwara and G. Irie. Efficient Label Propagation. In *ICML*, pages 784–792, 2014.
- [9] Y. Fujiwara, G. Irie, S. Kuroyama, and M. Onizuka. Scaling Manifold Ranking Based Image Retrieval. *PVLDB*, 8(4):341–352, 2014.
- [10] Y. Fujiwara, M. Nakatsuji, H. Shiokawa, Y. Ida, and M. Toyoda. Adaptive Message Update for Fast Affinity Propagation. In *SIGKDD*, pages 309–318, 2015.
- [11] Y. Fujiwara, M. Nakatsuji, H. Shiokawa, T. Mishima, and M. Onizuka. Fast and Exact Top-k Algorithm for PageRank. In *AAAI*, 2013.
- [12] N. Halko, P. Martinsson, and J. A. Tropp. Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. *SIAM Review*, 53(2):217–288, 2011.
- [13] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2011.
- [14] S. Ji. Computational Network Analysis of the Anatomical and Genetic Organizations in the Mouse Brain. *Bioinformatics*, 27(23):3293–3299, 2011.
- [15] S. Liao, Y. Gao, and D. Shen. Sparse Patch Based Prostate Segmentation in CT Images. In *MICCAI*, pages 385–392, 2012.
- [16] N. Meinshausen and P. Bühlmann. High-Dimensional Graphs and Variable Selection with the Lasso. *The Annals of Statistics*, 34(3):1436–1462, June 2006.
- [17] T. Mishima and Y. Fujiwara. Madeus: Database Live Migration Middleware under Heavy Workloads for Cloud Environment. In *SIGMOD*, pages 315–329, 2015.
- [18] M. Muja and D. G. Lowe. Scalable Nearest Neighbor Algorithms for High Dimensional Data. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(11):2227–2240, 2014.
- [19] M. Nakatsuji, Y. Fujiwara, A. Tanaka, T. Uchiyama, and T. Ishida. Recommendations over Domain Specific User Graphs. In *ECAI*, pages 607–612, 2010.
- [20] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes 3rd Edition*. Cambridge University Press, 2007.
- [21] H. Shiokawa, Y. Fujiwara, and M. Onizuka. SCAN++: Efficient Algorithm for Finding Clusters, Hubs and Outliers on Large-scale Graphs. *PVLDB*, 8(11):1178–1189, 2015.
- [22] J. M. Steele. *The Cauchy-Schwarz Master Class: An Introduction to the Art of Mathematical Inequalities*. Cambridge University Press, 2004.
- [23] R. Tibshirani. Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1996.
- [24] R. Tibshirani, J. Bien, J. Friedman, T. Hastie, N. Simon, J. Taylor, and R. J. Tibshirani. Strong Rules for Discarding Predictors in Lasso-type Problems. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 74(2):245–266, 2012.
- [25] F. Wang and C. Zhang. Label Propagation through Linear Neighborhoods. *IEEE Trans. Knowl. Data Eng.*, 20(1):55–67, 2008.