Learning Interpretable Metric between Graphs: Convex Formulation and Computation with Graph Mining

Tomoki Yoshida
Nagoya Institute of Technology
yoshida.t.mllab.nit@gmail.com

Ichiro Takeuchi
Nagoya Institute of Technology
National Institute for Material Science
RIKEN Center for Advanced Intelligence Project
takeuchi.ichiro@nitech.ac.jp

Masayuki Karasuyama
Nagoya Institute of Technology
National Institute for Material Science
Japan Science and Technology Agency
karasuyama@nitech.ac.jp

ABSTRACT

Graph is a standard approach to modeling structured data. Although many machine learning methods depend on the metric of the input objects, defining an appropriate distance function on graph is still a controversial issue. We propose a novel supervised metric learning method for a subgraph-based distance, called *interpretable graph metric learning* (IGML). IGML optimizes the distance function in such a way that a small number of important subgraphs can be adaptively selected. This optimization is computationally intractable with naive application of existing optimization algorithms. We construct a graph mining based efficient algorithm to deal with this computational difficulty. Important advantages of our method are 1) guarantee of the optimality from the convex formulation, and 2) high interpretability of results. To our knowledge, none of the existing studies provide an interpretable subgraph-based metric in a supervised manner. In our experiments, we empirically verify superior or comparable prediction performance of IGML to other existing graph classification methods which do not have clear interpretability. Further, we demonstrate usefulness of IGML through some illustrative examples of extracted subgraphs and an example of data analysis on the learned metric space.

CCS CONCEPTS

• **Computing methodologies** → **Supervised learning by classification**; **Feature selection**.

KEYWORDS

metric learning, structured data, graph mining

ACM Reference Format:
Tomoki Yoshida, Ichiro Takeuchi, and Masayuki Karasuyama. 2019. Learning Interpretable Metric between Graphs: Convex Formulation and Computation with Graph Mining. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3292500.3330845

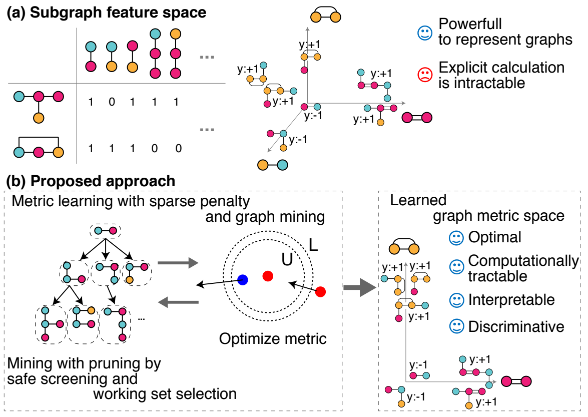
(a) Subgraph feature space
1 0 1 1 1
1 1 1 0 0
Powerfull to represent graphs
Explicit calculation is intractable
y:+1
y:-1
(b) Proposed approach
Metric learning with sparse penalty and graph mining
U
L
Optimize metric
Mining with pruning by safe screening and working set selection
Learned graph metric space
Optimal
Computationally tractable
Interpretable
Discriminative

Figure 1: Overview of our approach. (a): The subgraph-based feature is an effective way to represent graphs, but naively enumerating all possible subgraphs is intractable. (b): Our method optimizes the metric on graph, in which a sparse penalty selects a small number of discriminative subgraphs. We propose efficient optimization algorithm by using safe screening, working set selection, and their pruning extensions on the graph mining tree.

1 INTRODUCTION

The growing diversity of data science applications increases importance of the structured data analysis, in which it is difficult to create usual numerical vector representations of data. A standard approach to modeling structured data is to use *graph*. For example, in domains such as chemo- and bio-informatics, a large amount of graph structured data is becoming prevalent. In this paper, we particularly focus on the case in which a data instance is represented as a pair of a graph and an associated class label.

Although many machine learning methods explicitly/implicitly depend on how to measure difference between input objects, defining an appropriate distance metric on graph is still a controversial issue in the community. A widely accepted approach is *graph kernel* [12, 40]. This approach makes it possible to apply machine learning methods to graph data without having explicit vector representations. Another popular approach would be to use neural network [2, 24] by which suitable representation is expected to be learned in the network and then we can avoid explicitly defining metric. However, in those existing approaches, it is difficult to explicitly extract significant sub-structures, i.e., subgraphs, for prediction though identifying them should be quite informative. For example, finding a subgraph in molecular graph which has strong effect on toxicity would be insightful. See Section 3 for further details of existing work.

We propose a supervised method which obtains a metric of graph by which both of high predictive performance and interpretability can be achieved. Our method, called *interpretable graph metric learning* (IGML), combines an idea of *metric learning* [41] with a subgraph representation of graph in which each graph is represented through a set of subgraphs contained in it. Let $\phi_H(G)$ be a feature of the graph G, which is monotonically non-decreasing with respect to the frequency of subgraph H contained in G (e.g., frequency itself). We consider the squared distance between two graphs G and G': $d_{\boldsymbol{m}}(G, G') := \sum_{H \in \mathcal{G}} m_{i(H)} \left(\phi_H(G) - \phi_H(G')\right)^2$,where $\mathcal{G}$ is a set of all connected graphs, and $m_{i(H)}$ is a weight for the subgraph H. Although it is known that the subgraph approach has strong representation capability of graph (e.g., [12]), naive calculation is obviously infeasible except when the weight parameters has some special structure.

We formulate IGML as a supervised learning problem of the distance function $d_{\boldsymbol{m}}(G, G')$ with a sparse penalty on $m_{i(H)}$. The resulting optimization problem is computationally infeasible at a glance because the number of the weight parameters is equal to the number of possible subgraphs which is usually intractable. We overcome this difficulty by introducing *safe screening* [13] and *working set selection* [9] approach. Both of these approaches can drastically reduce the number of variables which we need to optimize, and further, both of them can be combined with a *pruning* strategy on the tree traverse of *graph mining*. These optimization tricks are inspired by two recent papers [23] and [22] which are safe screening- and working set- based pruning for a linear prediction model with the LASSO penalty. By combining these two techniques, we construct a path-wise optimization method which can obtain the sparse solution of the weight parameter $m_{i(H)}$ without directly enumerating all of possible subgraphs.

To our knowledge, none of the existing studies can provide an interpretable subgraph-based metric in a supervised manner. The overview of our approach is illustrated in Figure 1. Advantages of IGML can be summarized as follows

- Since the problem is formulated as a convex optimization, the global optimal can be found by the standard gradient-based optimization.
- The safe screening- and working set- based optimization algorithm makes our problem practically tractable without sacrificing the optimality.
- We can identify a small number of important subgraphs which discriminate different classes. This means that the resulting metric is easy to compute and highly interpretable, and thus useful for a variety of subsequent data analysis. For example, applying nearest neighbor classification or decision tree on the learned space would be effective.

In our experiments, we empirically verify superior or comparable prediction performance of IGML to other existing graph classification methods which do not have interpretability. We further show some examples of extracted subgraphs and data analysis on the learned metric space.

2 PROPOSED METHOD: INTERPRETABLE GRAPH METRIC LEARNING (IGML)

Here, we describe our proposed graph metric learning method, called *interpretable graph metric learning* (IGML). In Section 2.1, we first show the convex optimization formulation of the learning problem. Next, properties of the optimal solution is analyzed in Section 2.2, based on which we derive several techniques for accelerating the optimization process in Section 2.3. Section 2.4 summarizes the entire computational procedure, and Section 2.5 describes post-processing on learned metric. The proofs for all the lemma and the theorems are in appendix.

2.1 Problem Formulation

Let $\mathcal{G}$ be a set of all connected graphs, for each of which vertices and edges can be labeled. If $H \in \mathcal{G}$ is an induced connected subgraph of $G \in \mathcal{G}$, we write $H \sqsubseteq G$. If G and G' are isomorphic as labeled graphs, we write $G' \simeq G'$. Then, the frequency of the subgraph H contained in G is written as $|\{G' \mid G' \simeq H, G' \sqsubseteq G\}|$. As a representation of a graph G, we consider the following subgraph-based feature representation:

$$\phi_H(G) = g\left(|\{G' \mid G' \simeq H, G' \sqsubseteq G\}|\right), \text{ for } H \in \mathcal{G},$$

where g is some monotonically non-decreasing function such as the identity function $g(x) = x$ or the indicator function $g(x) = 1_{x>0}$ which takes 1 if $x > 0$ otherwise 0. It is widely known that the subgraph-based feature is an effective way to represent graphs. For example, $g(x) = x$ allows to distinguish all non-isomorphic graphs [12]. However, this feature space is practically infeasible to calculate because the possible number of subgraphs is quite huge.

In this paper, we focus on the problem of how to measure the distance between two graphs which is essential for a variety of machine learning methods. We consider the following weighted squared distance between two graphs:

$$d_{\boldsymbol{m}}(G, G') := \sum_{H \in \mathcal{G}} m_{i(H)} \left(\phi_H(G) - \phi_H(G')\right)^2, \tag{1}$$

where $i(H)$ is the index of the subgraph H for a weight parameter $m_{i(H)} \geq 0$. To obtain an effective and computable distance metric, we adaptively estimate $m_{i(H)}$ so that only a small number of important subgraphs have non-zero $m_{i(H)}$.

Suppose that the training dataset $\{(G_i, y_i)\}_{i \in [n]}$ consists of n pairs of a graph G_i and a class label y_i, where $[n] := \{1, \ldots, n\}$. Let $\boldsymbol{x}_i \in \mathbb{R}^p$ be the feature vector defined by concatenating $\phi_H(G_i)$ for all $H \in \mathcal{G}$ included in the training dataset. Then, we see

$$d_{\boldsymbol{m}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i - \boldsymbol{x}_j)^\top \mathrm{diag}(\boldsymbol{m})(\boldsymbol{x}_i - \boldsymbol{x}_j) = \boldsymbol{m}^\top \boldsymbol{c}_{ij},$$

where $\boldsymbol{m} \in \mathbb{R}_+^p$ is a vector of $m_{i(H)}$, and $\boldsymbol{c}_{ij} \in \mathbb{R}^p$ is defined as $\boldsymbol{c}_{ij} := (\boldsymbol{x}_i - \boldsymbol{x}_j) \circ (\boldsymbol{x}_i - \boldsymbol{x}_j)$ by using the element-wise product $\circ$.

Let $\mathcal{S}_i \subseteq [n]$ be a subset of indices which are in the same class as $\boldsymbol{x}_i$, and $\mathcal{D}_i \subseteq [n]$ be a subset of indices which are in different classes from $\boldsymbol{x}_i$. For both of these sets, we select K most similar inputs to $\boldsymbol{x}_i$ by using some default metric (e.g., using graph kernel). As a loss function for $\boldsymbol{x}_i$, we consider

$$\ell_i(\boldsymbol{m}; L, U) := \sum_{l \in \mathcal{D}_i} \ell_L(\boldsymbol{m}^\top \boldsymbol{c}_{il}) + \sum_{j \in \mathcal{S}_i} \ell_{-U}(-\boldsymbol{m}^\top \boldsymbol{c}_{ij}),$$

where $L, U \in \mathbb{R}_+$ are the constant parameters satisfying $U \leq L$, and $\ell_t(x) = [t - x]_+^2$ is the standard squared hinge loss function with the threshold $t \in \mathbb{R}$. This loss function is a variant of the pairwise loss function which is widely used in metric learning [7]. The first term in the loss function gives a penalty if $\boldsymbol{x}_i$ and $\boldsymbol{x}_l$ are closer than L for $l \in \mathcal{D}_i$, and the second term gives a penalty if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are distant than U for $j \in \mathcal{S}_i$.

Let $R(\boldsymbol{m}) = \|\boldsymbol{m}\|_1 + \frac{\eta}{2}\|\boldsymbol{m}\|_2^2 = \boldsymbol{m}^\top \boldsymbol{1} + \frac{\eta}{2}\|\boldsymbol{m}\|_2^2$ be an elastic-net type sparsity inducing penalty, where $\eta \geq 0$ is a non-negative parameter. We define our proposed IGML as the following regularized loss minimization problem:

$$\min_{\boldsymbol{m} \geq \boldsymbol{0}} P_\lambda(\boldsymbol{m}) := \sum_{i \in [n]} \ell_i(\boldsymbol{m}; L, U) + \lambda R(\boldsymbol{m}), \tag{2}$$

where $\lambda > 0$ is the regularization parameter. The solution of this problem can provide discriminative and interpretable metric, but the optimization problem is computationally intractable because of the huge dimensionality p of $\boldsymbol{m}$. To construct an efficient algorithm, we first see some analytical properties of the optimal solution of this problem.

2.2 Analytical Properties of Optimal Solution

Let $\boldsymbol{\alpha} \in \mathbb{R}_+^{2nK}$, in which dual variables α_{il} and α_{ij} for $i \in [n], l \in \mathcal{D}_i$ and $j \in \mathcal{S}_i$ are concatenated. The dual problem of the primal problem (2) can be written as follows (see appendix A for detail):

$$\max_{\boldsymbol{\alpha} \geq \boldsymbol{0}} D_\lambda(\boldsymbol{\alpha}) := -\frac{1}{4}\|\boldsymbol{\alpha}\|_2^2 + \boldsymbol{t}^\top \boldsymbol{\alpha} - \frac{\lambda\eta}{2}\|\boldsymbol{m}_\lambda(\boldsymbol{\alpha})\|_2^2, \tag{3}$$

where

$$\boldsymbol{m}_\lambda(\boldsymbol{\alpha}) := \frac{1}{\lambda\eta}[\boldsymbol{C}\boldsymbol{\alpha} - \lambda \boldsymbol{1}]_+, \tag{4}$$

$\boldsymbol{t} := [L, \ldots, L, -U, \ldots, -U]^\top \in \mathbb{R}^{2nK}$ and $\boldsymbol{C} := [\ldots, \boldsymbol{c}_{il}, \ldots, -\boldsymbol{c}_{ij}, \ldots] \in \mathbb{R}^{p \times 2nK}$. Then, from the optimality condition, we obtain the following relationship between primal and dual variables:

$$\alpha_{il} = -\ell'_L(\boldsymbol{m}^\top \boldsymbol{c}_{il}), \ \alpha_{ij} = -\ell'_{-U}(\boldsymbol{m}^\top \boldsymbol{c}_{ij}), \tag{5}$$

where, $\ell'_t(x) = -2[t - x]_+$ is the derivative of ℓ_t. When the regularization parameter λ is larger than certain $\lambda_{\max}$, the optimal solution is $\boldsymbol{m} = \boldsymbol{0}$. Then, the optimal dual variables are $\alpha_{il} = -\ell'_L(0)$ and $\alpha_{ij} = -\ell'_{-U}(0)$. By substituting these equations into (4), we can obtain $\lambda_{\max}$ as

$$\lambda_{\max} = \max_k \boldsymbol{C}_{k,:}\boldsymbol{\alpha}. \tag{6}$$

In subsequent sections, we build an efficient algorithm by confining the region in which optimal $\boldsymbol{m}$ must exist. The following theorem provides a hyper-sphere in which optimal dual variable $\boldsymbol{\alpha}^\star$ must lie:

Theorem 2.1 (DGB). *For any pair of $\boldsymbol{m} \geq \boldsymbol{0}$ and $\boldsymbol{\alpha} \geq \boldsymbol{0}$, the optimal dual variable $\boldsymbol{\alpha}^\star$ must satisfy*

$$\|\boldsymbol{\alpha} - \boldsymbol{\alpha}^\star\|_2^2 \leq 4(P_\lambda(\boldsymbol{m}) - D_\lambda(\boldsymbol{\alpha})).$$

This bound is called *duality gap bound* (DGB), and the parameters $\boldsymbol{m}$ and $\boldsymbol{\alpha}$ used to construct the bound is called *reference solution.*

If the optimal solution for λ_0 is available as a reference solution to construct the bound for λ_1, the following bound, called *regularization path bound* (RPB), can be obtained:

Theorem 2.2 (RPB). *Let $\boldsymbol{\alpha}_0^\star$ be the optimal solution for λ_0, and $\boldsymbol{\alpha}_1^\star$ be the optimal solution for λ_1.*

$$\left\|\boldsymbol{\alpha}_1^\star - \frac{\lambda_0 + \lambda_1}{2\lambda_0}\boldsymbol{\alpha}_0^\star\right\|_2^2 \leq \left\|\frac{\lambda_0 - \lambda_1}{2\lambda_0}\boldsymbol{\alpha}_0^\star\right\|_2^2.$$

However, RPB requires the exact solution, which is difficult in practice because of numerical errors. *Relaxed RPB* (RRPB) extends RPB so that it can incorporate the approximate solution as a reference solution:

Theorem 2.3 (RRPB). *Assuming that $\boldsymbol{\alpha}_0$ satisfies $\|\boldsymbol{\alpha}_0 - \boldsymbol{\alpha}_0^\star\|_2 \leq \epsilon$. The optimal solution $\boldsymbol{\alpha}_1^\star$ for λ_1 must satisfy*

$$\left\|\boldsymbol{\alpha}_1^\star - \frac{\lambda_0 + \lambda_1}{2\lambda_0}\boldsymbol{\alpha}_0\right\|_2 \leq \left\|\frac{\lambda_0 - \lambda_1}{2\lambda_0}\boldsymbol{\alpha}_0\right\|_2 + \left(\frac{\lambda_0 + \lambda_1}{2\lambda_0} + \frac{|\lambda_0 - \lambda_1|}{2\lambda_0}\right)\epsilon.$$

For example, ϵ can be obtained by using DGB (Theorem 2.1).

Similar bounds to these bounds derived here are previously considered for the triplet screening of metric learning on usual numerical data [44]. We here extend a similar idea for deriving subgraph screening.

2.3 Safe Screening and Working Set Selection with Pruning Strategy

For making the optimization problem tractable, we work with only a small subset of features during the optimization process. Let $\mathcal{F} \subseteq [p]$ be a subset of features. By fixing $m_i = 0$ for $i \notin \mathcal{F}$, we define sub-problems of the original primal P_λ and dual D_λ problems as follows

$$\min_{\boldsymbol{m}_\mathcal{F} \geq \boldsymbol{0}} P_\lambda^\mathcal{F}(\boldsymbol{m}_\mathcal{F}) := \sum_{i \in [n]} [\sum_{l \in \mathcal{D}_i} \ell_L(\boldsymbol{m}_\mathcal{F}^\top \boldsymbol{c}_{il\mathcal{F}}) + \sum_{j \in \mathcal{S}_i} \ell_{-U}(-\boldsymbol{m}_\mathcal{F}^\top \boldsymbol{c}_{ij\mathcal{F}})] + \lambda R(\boldsymbol{m}_\mathcal{F}) \tag{7}$$

$$\max_{\boldsymbol{\alpha} \geq \boldsymbol{0}} D_\lambda^\mathcal{F}(\boldsymbol{\alpha}) := -\frac{1}{4}\|\boldsymbol{\alpha}\|_2^2 + \boldsymbol{t}^\top \boldsymbol{\alpha} - \frac{\lambda\eta}{2}\|\boldsymbol{m}_\lambda(\boldsymbol{\alpha})_\mathcal{F}\|_2^2, \tag{8}$$

where $\boldsymbol{m}_\mathcal{F}$, $\boldsymbol{c}_{ij\mathcal{F}}$, and $\boldsymbol{m}_\lambda(\boldsymbol{\alpha})_\mathcal{F}$ are sub-vectors specified by $\mathcal{F}$. If the size of $\mathcal{F}$ is reasonably small, these sub-problems are computationally much easier than the original problems.

In this section, we introduce several criteria which determine whether the feature k is included in $\mathcal{F}$ by using techniques called *safe screening* [13] and *working set selection* [9]. A general form of our criteria can be written as

$$\boldsymbol{C}_{k,:}\boldsymbol{q} + r\|\boldsymbol{C}_{k,:}\|_2 \leq T, \tag{9}$$

where $\boldsymbol{q} \in \mathbb{R}_+^{2nK}$, $r \geq 0$, and $T \in \mathbb{R}$ are constants which take different values depending on the type of the criterion. If this inequality holds for k, we exclude the k-th feature from $\mathcal{F}$. An important property of our approach is that although our algorithm only solves these small sub-problems, we can guarantee the optimality of the final solution as we see later.

However, selecting $\mathcal{F}$ itself is computationally quite expensive since the evaluation of (9) needs $O(n)$ computations for each k. Our key idea is to utilize the tree structure of graphs for determining $\mathcal{F}$. Figure 2 shows an example of the tree, which can be constructed by graph mining algorithms such as gSpan [42]. Suppose that the k-th node corresponds to the k-th dimension of $\boldsymbol{x}$ (Note that here the node index is not the order of the visit). If a graph corresponding to the k-th node is a subgraph of the k'-th node, the node k' is a

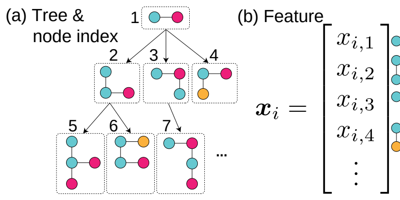

Figure 2: Schematic illustration of tree and feature.

descendant of k which is denoted as $k' \supseteq k$. Then, the following monotonic relation is immediately derived from the monotonicity of ϕ_H:

$$x_{i,k'} \leq x_{i,k} \text{ if } k' \supseteq k. \tag{10}$$

Based on this property, the following lemma enables us to prune a node during the tree traverse:

Lemma 2.1. *Let*

$$\text{Prune}(k|\boldsymbol{q}, r) := \sum_{i \in [n]} \sum_{l \in \mathcal{D}_i} q_{il} \max\{x_{i,k}, x_{l,k}\}^2 + r\sqrt{\sum_{i \in [n]} [\sum_{l \in \mathcal{D}_i} \max\{x_{i,k}, x_{l,k}\}^4 + \sum_{j \in \mathcal{S}_i} \max\{x_{i,k}, x_{j,k}\}^4]}$$

be a pruning criterion. Then, if the inequality

$$\text{Prune}(k|\boldsymbol{q}, r) \leq T, \tag{11}$$

holds, for any descendant node $k' \supseteq k$, the following inequality holds

$$C_{k',:}\boldsymbol{q} + r\|C_{k',:}\|_2 \leq T,$$

where $\boldsymbol{q} \in \mathbb{R}_+^{2nK}$ and $r \geq 0$ are arbitrary constant vector and scalar variable.

This lemma indicates that if the condition (11) is satisfied, we can determine that all the descendant nodes are not included in $\mathcal{F}$. Next Sections 2.3.1, 2.3.2 and 2.3.3, we introduce different types of pruning strategies based on this lemma.

2.3.1 Safe Screening (SS) and Safe Pruning (SP). Theorem 2.1 and 2.3 identify the region, in which optimal solution exists, by using a current feasible solution $\boldsymbol{\alpha}$. Based on those regions, the following *safe screening* (SS) rule can be directly derived:

Theorem 2.4 (SS Rule). *If the optimal solution $\boldsymbol{\alpha}^\star$ exists in the bound $\mathcal{B} = \{\boldsymbol{\alpha} \mid \|\boldsymbol{\alpha} - \boldsymbol{q}\|_2^2 \leq r^2\}$, the following rule holds*

$$C_{k,:}\boldsymbol{q} + r\|C_{k,:}\|_2 \leq \lambda \Rightarrow m_k^\star = 0. \tag{12}$$

Proof. From (4), when $C_{k,:}\boldsymbol{\alpha}^\star \leq \lambda$, we see $m_k^\star = 0$. Then, $\max_{\boldsymbol{\alpha} \in \mathcal{B}} C_{k,:}\boldsymbol{\alpha} \leq \lambda \Rightarrow m_k^\star = 0$.By solving this problem analytically, we obtain SS Rule. □

Theorem 2.4 indicates that we can eliminate unnecessary features by evaluating the condition shown in (12). An important property of this rule is that it guarantees optimality, meaning that the subproblems (7) and (8) have the exactly same optimal solution to the original problem if $\mathcal{F}$ is defined through this rule. However, it is still necessary to evaluate the rule for all p features which is currently intractable. To avoid this problem, we derive a pruning strategy on the graph mining tree, which we call *safe pruning* (SP) rule:

Theorem 2.5 (SP Rule). *If the optimal solution $\boldsymbol{\alpha}^\star$ is in the bound $\mathcal{B} = \{\boldsymbol{\alpha} \mid \|\boldsymbol{\alpha} - \boldsymbol{q}\|_2^2 \leq r^2, \boldsymbol{q} \geq \boldsymbol{0}\}$, the following rule holds*

$$\text{Prune}(k|\boldsymbol{q}, r) \leq \lambda \Rightarrow m_{k'}^\star = 0. \text{ for } \forall k' \supseteq k. \tag{13}$$

This theorem is direct consequence of lemma 2.1. If this condition holds for a node k during the tree traverse, a subtree below that node can be pruned. This means that we can eliminate unnecessary subgraphs safely even without enumerating it.

2.3.2 Range-Based Safe Screening (RSS) & Range-Based Safe Pruning (RSP). SS and SP are rules for a fixed λ. The range-based extension identifies an interval of λ in which SS/SP is guaranteed to be satisfied. This is particularly useful for the *path-wise optimization* or *regularization path* calculation in which we need to solve the problem with a sequence of λ. We assume that the sequence is sorted with the descending order because usually optimization algorithms start from the trivial solution $\boldsymbol{m} = \boldsymbol{0}$. Let $\lambda = \lambda_1 \leq \lambda_0$. By combining RRPB with the rule (12), we obtain the following theorem:

Theorem 2.6 (Range-Based Safe Screening (RSS)). *For any k, the following rule holds*

$$\lambda_a \leq \lambda \leq \lambda_0 \Rightarrow m_k^\star = 0, \tag{14}$$

where $\lambda_a := \frac{\lambda_0(2\epsilon\|C_{k,:}\|_2 + \|\boldsymbol{\alpha}_0\|_2\|C_{k,:}\|_2 + C_{k,:}\boldsymbol{\alpha}_0)}{2\lambda_0 + \|\boldsymbol{\alpha}_0\|_2\|C_{k,:}\|_2 - C_{k,:}\boldsymbol{\alpha}_0}$.

For SP, the range-based rule can also be derived from (13):

Theorem 2.7 (Range-Based Safe Pruning (RSP)). *For any $k' \supseteq k$, the following pruning rule holds:*

$$\lambda'_a := \frac{\lambda_0(2\epsilon b + \|\boldsymbol{\alpha}_0\|_2 b + a)}{2\lambda_0 + \|\boldsymbol{\alpha}_0\|_2 b - a} \leq \lambda \leq \lambda_0 \Rightarrow m_{k'}^\star = 0, \tag{15}$$

where $a := \sum_{i \in [n]} \sum_{l \in \mathcal{D}_i} \alpha_{0il} \max\{x_{l,k}, x_{i,k}\}^2$, *and* $b := \sqrt{\sum_{i \in [n]} [\sum_{l \in \mathcal{D}_i} \max\{x_{i,k}, x_{l,k}\}^4 + \sum_{j \in \mathcal{S}_i} \max\{x_{i,k}, x_{j,k}\}^4]}$.

After once we calculate λ_a and λ'_a of (14) and (15) for some λ, we can store them at each node of the tree. Then, those λ_a and λ'_a can be used for the next tree traverse with different λ'. If the condition (14) or (15) is satisfied, the node can be skipped (RSS) or pruned (RSP). Otherwise, we update λ_a and λ'_a by using the current reference solution.

2.3.3 Working Set Selection (WS) & Working Set Pruning (WP). Safe rules are quite strong rules in a sense that they can remove features completely, and thus sometimes they are too conservative to fully accelerate the optimization. On the other hand, *working set selection* is a widely accepted heuristic approach to selecting a subset of features, called working set. Working set method optimizes the problem only with respect to selected working set features. Then, if the optimality condition for the original problem is not satisfied, the working set is selected again and the optimization on the new working set re-starts. This process iterates until the optimality on the original problem is achieved.

Besides safe rules, we further employ this heuristic approach based on the following criterion:

$$C_{k,:}\boldsymbol{\alpha} \leq \lambda. \tag{16}$$

If this inequality is satisfied, the k-th dimension is predicted as $m_k^\star = 0$. In other words, the working set is defined by

$$\mathcal{W} := \{k \mid C_{k,:}\boldsymbol{\alpha} > \lambda\}.$$

Although $m_i^\star = 0$ for $i \notin \mathcal{W}$ is not directly guaranteed, the convergence can be guaranteed by the following way:

Theorem 2.8 (Convergence of WS). *Assuming that there is a solver for the sub-problem (7) (or equivalently (8)) which returns the optimal solution for given $\mathcal{F}$. Working set method, which iterates optimizing the sub-problem with $\mathcal{F} = \mathcal{W}$ and updating $\mathcal{W}$ alternately, returns the optimal solution of the original problem with finite steps.*

However, here again, the inequality (16) needs to be evaluated for all features which is computationally intractable.

The same pruning strategy as SS/SP can be incorporated into working set selection. The criterion (16) is also a special case of (9), and lemma 2.1 indicates that if the following inequality

$$\text{Prune}_{\text{WP}}(k) := \text{Prune}(k|\boldsymbol{\alpha}, 0) \leq \lambda,$$

holds, any $k' \supseteq k$ is not included in the working set.

Note that for working set method, we may need to update $\mathcal{W}$ multiple times unlike safe screening approaches as we see in theorem 2.8. Instead of that, working set method usually can exclude a larger number of features compared with safe screening approaches (more rigorous discussion for the tightness of the selection rules is possible but we omit here due to the space limitation). We build an algorithm which combines safe screening and working set approach described so far.

2.4 Computations

2.4.1 Training with Path-wise Optimization. We employ *path-wise optimization* [11] in which the optimization starts from $\lambda = \lambda_{\max}$ and gradually decreases λ while optimizing $\boldsymbol{m}$. As we see in (6), $\lambda_{\max}$ is defined by the maximum of the inner product $C_{k,:}\boldsymbol{\alpha}$. This value can also be found by the tree search with pruning. Suppose that we calculate $C_{k,:}\boldsymbol{\alpha}$ while traversing the tree, and $\hat{\lambda}_{\max}$ is the current maximum value during the traverse. Using lemma 2.1, we can derive the pruning rule

$$\text{Prune}(k|\boldsymbol{\alpha}, 0) \leq \hat{\lambda}_{\max}.$$

If this condition holds, descendant nodes of k can not be the maximum, and thus we can identify $\lambda_{\max}$ without calculating $C_{k,:}\boldsymbol{\alpha}$ for all candidate features.

Algorithm 1 shows the outer loop of our path-wise optimization. TRAVERSE and SOLVE function in Algorithm 1 are shown in Algorithm 2 and 3, respectively. Algorithm 1 first calculates $\lambda_{\max}$ which is the minimum λ at which the optimal solution is $\boldsymbol{m}^\star = \boldsymbol{0}$ (line 3). The outer loop in line 5-14 is the process of decreasing λ with the decreasing rate R. The TRAVERSE function in line 7 determines the subset of features $\mathcal{F}$ by traversing tree with safe screening and working set selection. The inner loop (line 9-13) alternately solves the optimization problem with the current $\mathcal{F}$ and updates $\mathcal{F}$, until the duality gap becomes less than the given threshold eps.

Algorithm 2 shows the TRAVERSE function, which recursively visits the tree node to determine $\mathcal{F}$. The variable `node.pruning` contains λ'_a of RSP, and if the RSP condition (15) is satisfied (line 3), the function returns the current $\mathcal{F}$ (the node is pruned). The variable `node.screening` contains λ_a of RSS, and if the RSS condition (14) is satisfied (line 5), this node can be skipped and the function goes to the next node. If these two conditions are not satisfied, the function performs 1) updating `node.pruning` and `node.screening` if `update` is true, and 2) evaluating the conditions of RSP and WP (line 10), and RSS and WS (line 14). At line 17-18, gSpan expands children of the current node, and for each child node, the TRAVERSE function is called recursively.

Algorithm 1: Path-wise Optimization

```
1  function PathwiseOptimization($R$, $T$, freq, MaxIter, eps)
2      $\boldsymbol{m}_0 = \boldsymbol{0}$, $\boldsymbol{\alpha}_0 = [2L, \ldots, 2L, 0, \ldots, 0]$, $\epsilon = 0$
3      $\lambda_0 = \lambda_{\max} = \max_k C_{k,:}\boldsymbol{\alpha}_0$                ▷ Compute $\lambda_{\max}$
4      Initialize root node as root.children = empty, root.screening =
         $\infty$, and root.pruning = $\infty$
5      for $t = 1, 2, \ldots, T$ do
6          $\lambda_t = R\lambda_{t-1}$
7          $\mathcal{F}$ = Traverse($\lambda_{t-1}$, $\lambda_t$, $\boldsymbol{\alpha}_{t-1}$, $\epsilon$, root, true)      ▷ get
            working set & update range of $\lambda$
8          $\boldsymbol{m}_t = \boldsymbol{m}_{t-1}$
9          repeat
10             $\boldsymbol{m}_t$, $\boldsymbol{\alpha}_t$, $P$ = Solve($\lambda_t$, $\boldsymbol{m}_t$, $\mathcal{F}$, freq, MaxIter, eps)
11             $\mathcal{F}$ = Traverse(null, $\lambda_t$, $\boldsymbol{\alpha}_t$, null, root, false)
                ▷ update working set
12             gap = $P - D_{\lambda_t}^{\mathcal{F}}(\boldsymbol{\alpha}_t)$
13         until $\frac{\text{gap}}{P} \leq$ eps                       ▷ check optimality
14         $\epsilon = 2\sqrt{\text{gap}}$
15     return $\{\boldsymbol{m}_t\}_{t=0}^{t=T}$
```

Algorithm 3 shows a solver for the primal problem with the subset of features $\mathcal{F}$. Although we here employ a simple projected gradient algorithm, any optimization algorithm can be used in this process. In line 7-10, the SS rule is evaluated at every freq iterations. Note that this SS is only for sub-problems (7) and (8) created by current $\mathcal{F}$ (not for the original problems).

2.4.2 Enumerating Subgraphs for Test Data. To obtain a feature vector for test data, we only need to enumerate subgraphs which have $m_k \neq 0$. When gSpan is used as a mining algorithm, a unique code, called *minimum DFS code*, is assigned to each node. If a DFS code for a node is $(a_1, a_2, \ldots, a_n)$, a child node is represented as $(a_1, a_2, \ldots, a_n, a_{n+1})$. This enables us to prune a node which does not generate subgraphs with $m_k \neq 0$. Suppose that a subgraph $(a_1, a_2, a_3) = (x, y, z)$ has to be enumerated, and the currently we are in the node $(a_1) = (x)$. Then, a child with $(a_1, a_2) = (x, y)$ should be traversed, but a child with $(a_1, a_2) = (x, w)$ can not generate (x, y, z) and then we can stop the traverse of this node.

2.5 Post-processing

2.5.1 Learning Mahalanobis Distance for Selected Subgraphs. Instead of $\boldsymbol{m}$, the following Mahalanobis distance can be considered

$$d_{\boldsymbol{M}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i - \boldsymbol{x}_j)^\top \boldsymbol{M}(\boldsymbol{x}_i - \boldsymbol{x}_j), \tag{17}$$

where $\boldsymbol{M}$ is a positive definite matrix. Directly optimizing $\boldsymbol{M}$ requires $O(p^2)$ primal variables and semi-definite constraint, and then the problem is computationally quite expensive even for relatively

Algorithm 2: Traverse gSpan with RSS&RSP+WS&WP

```
function Traverse($\lambda_0$, $\lambda$, $\boldsymbol{\alpha}_0$, $\epsilon$, node, update)
    $\mathcal{F} = \{\}$, $k$ = node.feature
    if node.pruning $\leq \lambda$ then                                  ▷ RSP Rule
        return $\mathcal{F}$
    else if node.screening $\leq \lambda$ then                           ▷ RSS Rule
        do nothing
    else                              ▷ Update the range of $\lambda$ if update = true
        if update = true then
            node.pruning = $\frac{\lambda_0(2\epsilon b+\|\boldsymbol{\alpha}_0\|_2 b+a)}{2\lambda_0+\|\boldsymbol{\alpha}_0\|_2 b-a}$     ▷ Eq.(15)
        if node.pruning $\leq \lambda$ or Prune$_{\mathrm{WP}}(k) \leq \lambda$ then
            return $\mathcal{F}$
        if update = true then
            node.screening = $\frac{\lambda_0(2\epsilon\|C_{k,:}\|_2+\|\boldsymbol{\alpha}_0\|_2\|C_{k,:}\|_2+C_{k,:}\boldsymbol{\alpha}_0)}{2\lambda_0+\|\boldsymbol{\alpha}_0\|_2\|C_{k,:}\|_2-C_{k,:}\boldsymbol{\alpha}_0}$ ▷ Eq.(14)
        if node.screening $> \lambda$ and $C_{k,:}\boldsymbol{\alpha}_0 > \lambda$ then
            $\mathcal{F} = \mathcal{F} \cup \{k\}$
        createChildren(node)
        for child = node.children do
            $\mathcal{F} = \mathcal{F} \cup$ Traverse($\lambda_0$, $\lambda$, $\boldsymbol{\alpha}_0$, $\epsilon$, child, update)
    return $\mathcal{F}$
function createChildren(node)
    if node.children = empty then
        Set node.children by gSpan
        for child = node.children do
            child.children = empty, child.screening = child.pruning = $\infty$
```

Algorithm 3: Gradient descent with dynamic screening

```
function Solve($\lambda$, $\boldsymbol{m}$, $\mathcal{F}$, freq, MaxIter, eps)      ▷ Solve primal
  problem $P_\lambda^{\mathcal{F}}$, which considered for only feature set $\mathcal{F}$
    for iter = 0, 1, . . . , MaxIter do
        Compute $\boldsymbol{\alpha}$ by (5)
        gap = $P_\lambda^{\mathcal{F}}(\boldsymbol{m}) - D_\lambda^{\mathcal{F}}(\boldsymbol{\alpha})$
        if $\frac{\text{gap}}{P_\lambda^{\mathcal{F}}(\boldsymbol{m})} \leq$ eps then                         ▷ convergence
            return $\boldsymbol{m}$, $\boldsymbol{\alpha}$, $P_\lambda^{\mathcal{F}}(\boldsymbol{m})$
        if mod(iter, freq) = 0 then
            for $k \in \mathcal{F}$ do                      ▷ perform dynamic screening
                if $C_{k,:}\boldsymbol{\alpha} + 2\sqrt{\text{gap}}\|C_{k,:}\|_2 \leq \lambda$ then        ▷ SS by DGB
                    $\mathcal{F} = \mathcal{F} - \{k\}$
        $\boldsymbol{m} = [\boldsymbol{m} - \gamma\nabla P_\lambda^{\mathcal{F}}(\boldsymbol{m})]_+$        ▷ update $\boldsymbol{m}$ ($\gamma$: step-size)
    return $\boldsymbol{m}$, $\boldsymbol{\alpha}$, $P_\lambda^{\mathcal{F}}(\boldsymbol{m})$
```

small p. Thus, in this paper, as an optional post-processing, we consider optimizing the Mahalanobis distance (17) for a small number of subgraphs selected by the optimized $\boldsymbol{m}$. Let $\mathcal{H} \subseteq \mathcal{G}$ be a set of subgraphs $m_{i(H)} > 0$ for $H \in \mathcal{H}$, and $\boldsymbol{z}_i$ be a $h := |\mathcal{H}|$ dimensional feature vector consisting of $\phi_H(G_i)$ for $H \in \mathcal{H}$. For $\boldsymbol{M} \in \mathbb{R}^{h\times h}$, we consider the following metric learning problem:

$$\min_{\boldsymbol{M}\succeq \boldsymbol{O}} \sum_{i\in[n]} [\sum_{l\in\mathcal{D}_i} \ell_L(d_{\boldsymbol{M}}(\boldsymbol{z}_i, \boldsymbol{z}_l)) + \sum_{j\in\mathcal{S}_i} \ell_{-U}(-d_{\boldsymbol{M}}(\boldsymbol{z}_i, \boldsymbol{z}_j))] + \lambda R(\boldsymbol{M}),$$

where $R : \mathbb{R}^{h\times h} \to \mathbb{R}$ is a regularization term for $\boldsymbol{M}$ for which a typical setting is $R(\boldsymbol{M}) = \mathrm{tr}\boldsymbol{M} + \frac{\eta}{2}\|\boldsymbol{M}\|_F^2$, where tr is the trace of a matrix. This metric can be more discriminative because it is optimized to the training data with the higher degree of freedom.

2.5.2 *Vector Representation of Graph.*

An explicit vector representation of an input graph can be obtained by using optimized $\boldsymbol{m}$:

$$\boldsymbol{x}_i' = \sqrt{\boldsymbol{m}} \circ \boldsymbol{x}_i \tag{18}$$

Unlike the original $\boldsymbol{x}_i$, the new representation $\boldsymbol{x}_i'$ is computationally tractable because of the sparsity of $\boldsymbol{m}$, and also this space should be highly discriminative. This property is beneficial for further analysis of the graph data. We show an example of applying decision tree on the learned space in our later experiment.

In the case of the general Mahalanobis distance shown in Section 2.5.1, we can obtain further transformation. Let $\boldsymbol{M} = \boldsymbol{V\Lambda V}^\top$ be the eigenvalue decomposition of the learned $\boldsymbol{M}$. When we use the regularization term $R(\boldsymbol{M}) = \mathrm{tr}\boldsymbol{M} + \frac{\eta}{2}\|\boldsymbol{M}\|_F^2$, a part of the eigenvalues of $\boldsymbol{M}$ can be shrunk to 0 because tr$\boldsymbol{M}$ is equal to the sum of the eigenvalues. If $\boldsymbol{M}$ has $h' < h$ non-zero eigenvalues, $\boldsymbol{\Lambda}$ can be written as a $h' \times h'$ diagonal matrix, and $\boldsymbol{V}$ is a $h \times h'$ matrix in which each column is the eigenvector of the non-zero eigenvalue. Then, a representation of graph is

$$\sqrt{\boldsymbol{\Lambda}}\boldsymbol{V}^\top \boldsymbol{z}_i. \tag{19}$$

This can be considered as a supervised dimensionality reduction from h to h' dimensional space. Although each dimension is no longer corresponding to a subgraph in this representation, the interpretation is still clear because each dimension of the transformed vector is just a linear combination of $\boldsymbol{z}_i$.

3 RELATED WORK

The most closely related approach with IGML would be subgraph-based graph kernels. Graphlet kernel [33] creates a kernel through small subgraphs having only about 3-5 vertices, called graphlet. Neighborhood subgraph pairwise distance kernel [6] selects pairs of subgraphs from a graph, and counts the number of identical pairs with another graph. Subgraph matching kernel [18] identifies common subgraphs based on cliques in the product graph of two graphs. Based on these approaches, subgraph-based feature vectors can be extracted. However, since these approaches are unsupervised, it is impossible to eliminate subgraphs which are unnecessary for classification. As a result, all candidate subgraphs must be once enumerated before applying a learning method unlike IGML.

There are many other kernels including shortest path- [4], random walk- [36, 40, 46], and spectrum- [15–17, 39] based approaches. Weisfeiler-Lehman (WL) kernel [31, 32], which is based on graph isomorphism test, is a popular and empirically successful kernel, and thus many studies have utilized it as a part of procedures [24, 27, 43, 45]. These approaches are again unsupervised, and it is quite difficult to interpret results from the perspective of substructures of a graph. Although several kernels deal with continuous attributes on vertices [10, 21, 29, 35], we only focus on the cases where vertex labels are discrete because of its high interpretability.

Since obtaining a good metric is an essential problem in data analysis, metric learning has been extensively studied so far as reviewed in [20]. However, due to computational difficulty, metric learning for graph data has not been widely studied. A few studies considered *edit distance* approaches. For example, [3] is a method of learning a similarity function through an edit distance in a supervised manner. Another approach is that [25] probabilistically formulates the editing process of the graph, and estimates the parameters by using labeled data. However, these approaches also can not provide any clear interpretation of resulting metric in a sense of subgraph.

Deep neural network (DNN) is also a standard approach to graph data analysis. Deep graph kernel [43] incorporates neural language modeling in which decomposed sub-structures of a graph is regarded as a sentence. PATCHY-SAN [27] and DGCNN [27] convert a graph to a tensor by using WL-Kernel and convolute it, and several other studies also have combined popular convolution techniques with the graph data [2, 34, 38]. These approaches are supervised, but obviously interpretability of these DNN is quite low. *Attention* enhances interpretability of deep learning, but extracting important subgraph is difficult because the attention for graph [19] only provides the significance of the vertex transition on a graph. Another related DNN approach would be the representation learning. For example, sub2vec [1] and graph2vec [24] can embed graph data into a continuous space, but they are unsupervised and it is difficult to extract substructures that characterize different classes. There are other fingerprint learning methods for graph by neural networks (e.g., [8]) in which the contribution from each node can be evaluated for each dimension of fingerprint. Although it is possible to highlight sub-structures for the given input graph, it does not directly produce important subgraphs for prediction (in other words, it does not create a feature space spanned by subgraphs as illustrated in Figure 1).

Supervised pattern mining [5, 28, 37] can be used for a similar purpose to our problem setting because it enumerates patterns (including graph) based on some discriminative score. However, these approaches usually 1) employ the greedy strategy to add a pattern by which global optimality can not be guaranteed, and 2) does not optimize metric or representation. A few other studies [23, 30] consider optimizing a linear model on the subgraph feature with the LASSO penalty by using graph mining, but these are specific to the parameter optimization of the specific linear model.

4 EXPERIMENTS

We evaluate performance of IGML using the benchmark datasets shown in Table 2. These datasets can be obtained from [14]. We did not use edge label because implementations of compared methods can not deal with edge label, and the maximum connected graph is used if the graph is not connected. #maxvertices in the table is the size (number of vertices) of the maximum subgraph considered in IGML. The sets $\mathcal{S}_i$ and $\mathcal{D}_i$ are selected as ten nearest neighborhoods of $\boldsymbol{x}_i$ ($K = |\mathcal{S}_i| = |\mathcal{D}_i| = 10$) by using WL-Kernel, respectively. A sequence of the regularization coefficients are created by equally spaced 100 grid points on the logarithmic scale between $\lambda_{\max}$ and $0.01\lambda_{\max}$. The gSpan search tree (in which minimum support is set as 0) is usually traversed only just after λ changes. In the working-set method, after convergence, it is necessary to traverse the tree again in order to confirm the overall optimality. If the termination condition is not satisfied, optimization with a new working set must

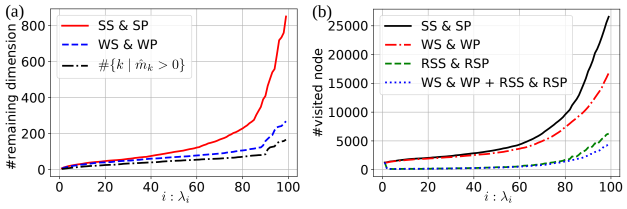

Figure 3: (a): The size of $\mathcal{F}$, and (b): the number of visited nodes. Both are evaluated at the first traverse of each λ for which the index is shown as the horizontal axis. The dataset is AIDS.

be performed. The termination condition for the optimization is that the relative duality gap is less than 10^{-6}. In this experiment, we used $g(x) = 1_{x>0}$ in $\phi_H(G)$. The dataset is randomly divided in such a way that the ratio of partitioning is train : validation : test = 0.6 : 0.2 : 0.2, and our experimental result is an average value of 10 times.

4.1 Evaluating Computational Efficiency

In this section, we confirm the effect of proposed pruning methods. We evaluated four settings, i.e., safe feature screening "SS&SP", its range based extention "RSS&RSP", working set selection "WS&WP", and the combination "WS&WP+RSS&RSP". Each method performs dynamic screening with DGB at every update of $\boldsymbol{m}$. We here used the AIDS dataset, in which #maxvertices=30. In this dataset, when we fully traverse the gSpan tree without safe screening/working set selection, the number of tree nodes was more than 9.126×10^7 (at which our implementation with gSpan stopped because of out of memory).

Figure 3 (a) shows the size of $\mathcal{F}$ after the first traverse at each λ, and the number of non-zero m_k after the optimization is also shown as a baseline. We first see that the both approaches reduced the number of features drastically. Even for the largest case, at which about 200 of features finally selected by m_k, only less than a thousand of features were remained. We see that WS&WP shows a much smaller values than SS&SP. Instead, WS&WP may need to perform the entire tree search again because it can not guarantee the sufficiency of current $\mathcal{F}$ while SS&SP does not need to search the tree again because it guarantees that $\mathcal{F}$ must contain all $m_k \neq 0$.

The number of visited nodes in the first traverse at each λ is shown in Figure 3 (b). We see that #visited nodes of SS&SP is largest than the others, but it is less than about 27000 ($27000/9.126 \times 10^7 \approx 0.0003$). Comparing SS&SP and WS&WP, we see that WS&WP pruned a larger number of nodes. On the other hand, #visited nodes of RSS&RSP is less than 6000. The difference between SS&SP and RSS&RSP indicates that a larger number of nodes can be skipped by range based method. Therefore, by combining the node skip by RSS&RSP and stronger pruning of WS&WP, #visited nodes was further reduced.

The total time in the path-wise optimization is shown in Table 1. RSS&RSP were fast in the traversing time, and WS&WP were fast in the solving time. In total, RSS&RSP+WS&WP was fastest. The result

Table 1: Total time in the path-wise optimization (sec).

Method \ Process	Traverse	Solve	Total
SS&SP	25.9±4.0	86.7±14.1	112.7±16.5
RSS&RSP	7.7±1.6		94.4±15.1
WS&WP	39.1±3.7	**55.0**±12.1	94.1±11.6
WS&WP+RSS&RSP	**7.4**±1.1		**62.5**±12.3

indicates that our method only took about 1 minute for solving the optimization problem with more than 9.126×10^7 variables.

4.2 Predictive Accuracy Comparison

In this section, we compare the prediction accuracy of IGML with Graphlet-Kernel (GK)[33], Shortest-Path Kernel (SPK)[4], Random-Walk Kernel (RW)[40], Weisfeiler-Lehman Kernel (WL)[32], and Deep Graph Convolutional Neural Network (DGCNN)[45]. We use the implementation available at URLs [1] for graph kernels and DGCNN, respectively. The prediction of the kernel method and IGML is made by the k-nearest neighbor (k-nn). The values of k in the k-nn is $k = 1, 3, 5, 7, \ldots, 49$ and hyperparameters of each method are selected by using validation data, and prediction accuracy is evaluated on test data. The graphlet size in GK is fixed to 3. The parameter λ_{RW} in RW is set by the recommended $\lambda_{\mathrm{RW}} = \max_{i \in \mathbb{Z}: 10^i < 1/d^2} 10^i$ where d is the maximum degree. The loop parameter h of WL is selected from $0, 1, 2, \ldots, 10$ by using the validation data. In DGCNN, the number of hidden unit and its sort-pooling are also selected by the validation data, each of which is from 64, 128, 256 and from 40%, 60%, 80%, respectively.

The micro-F1 score for each dataset is shown in Table 2. "IGML (Diag)" is IGML with the weighted squared distance (1), and "IGML (Diag→Full)" indicates that the post-processing with the Mahalanobis distance (17) was performed. We first focus on "IGML (Diag)" which shows the best score on the six of the nine datasets except for "IGML (Diag→Full)". Among those six datasets, "IGML (Diag→Full)" further improves the accuracy for the four datasets. The second best would be WL which shows superior performance to the other methods except for the proposed method on six of nine datasets. DGCNN shows high accuracy with DBLP_v1, which has a large number of samples, while for the other data, accuracy was not high.

4.3 Illustrative Examples

Figure 4 shows an illustrative example of IGML on Mutagenicity dataset, in which mutagenicity are predicted from a graph representation of molecules. Figure 4 (a) is a graphical representation of subgraphs, each of which has a weight shown in (b). For example, we can clearly see that the subgraph #2 is estimated as an important sub-structure to discriminate different classes. Figure 4 (c) shows a heatmap of the transformation matrix $\sqrt{\Lambda} V^\top$ optimized for these thirteen features, in which three non-zero eigenvalues exist. For example, we see that two subgraphs #10 and #12 have similar columns in the heatmap. This indicates that these two similar subgraphs (#10 contains #12) are shrunk to almost same representation by the effect of the regularization term $R(\boldsymbol{M})$.

As another example of graph data analysis on the learned representation, we applied the decision tree algorithm to the obtained

[1] http://mlcb.is.tuebingen.mpg.de/Mitarbeiter/Nino/Graphkernels/, https://github.com/muhanzhang/pytorch_DGCNN

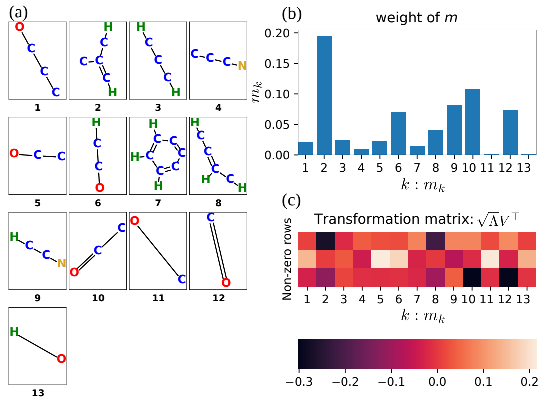

Figure 4: An example of selected subgraphs. (a): Illustration of subgraphs. (b): Learned weight of subgraph. (c): A transformation matrix (19).

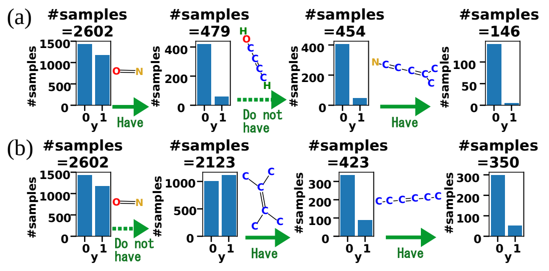

Figure 5: Examples of paths on the decision tree constructed by the selected subgraphs. #samples indicates the number of samples which satisfies all the preceding conditions.

feature (18) on Mutagenicity dataset. Although there is a study constructing the decision tree directly for graph data [26], it needs a severe restriction on the pattern to be considered for computational reasons. On the other hand, since (18) is a simple vector representation with reasonable dimension, it is quit easy to apply the decision tree algorithm. Due to space limitation, we select two paths from the obtained decision tree as shown in Figure 5. For example, in the path (a), if a given graph contains "$O=N$", and does not contain "$H-O-C-C=C-C-H$", and contains "$N-C=C-C=C<^{C}_{C}$", the given graph is predicted as $y = 0$ with probability 140/146. Both of two rules clearly separate two classes, and it would be highly insightful because we can trace the process of the decision based on the subgraphs.

5 CONCLUSIONS

We proposed an interpretable metric learning method for graph data, called *interpretable graph metric learning* (IGML). To avoid computational difficulty, we build an optimization algorithm which combines safe screening, working set selection, and their pruning extensions. We empirically evaluated the performance of IGML compared with existing graph classification methods. Although IGML was an only method which has clear interpretability, it showed superior or comparable prediction performance to other state-of-the-art

Table 2: Comparison of micro-F1 score. OOM means out of memory. ">1week" indicates that it did not finish within a week. "±" is a standard deviation. Every dataset has two classes.

Method \ Dataset	AIDS	BZR	DD	DHFR	FRANKENSTEIN	Mutagenicity	NCI1	COX2	DBLP_v1
#samples	2000	405	1178	467	4337	4337	4110	467	19456
#maxvertices	30	15	30	15	15	10	15	30	30
GK	0.985±0.006	0.815±0.034	0.632±0.021	0.688±0.037	0.603±0.012	0.747±0.017	0.703±0.011	0.782±0.045	OOM
SPK	0.994±0.003	0.842±0.039	>1week	0.737±0.040	0.640±0.012	0.719±0.014	0.722±0.012	0.774±0.034	0.784±0.012
RW	**0.998**±0.002	0.811±0.025	OOM	0.659±0.032	0.616±0.013	0.679±0.018	0.649±0.017	0.770±0.038	OOM
WL	0.995±0.003	0.854±0.039	0.769±0.027	0.780±0.045	0.694±0.017	0.768±0.012	0.772±0.015	**0.790**±0.040	0.814±0.014
DGCNN	0.985±0.005	0.791±0.020	0.773±0.023	0.678±0.030	0.615±0.016	0.705±0.018	0.706±0.016	0.764±0.039	**0.927**±0.003
IGML (Diag)	0.976±0.006	**0.860**±0.030	0.778±0.026	**0.797**±0.035	0.696±0.014	0.783±0.016	0.775±0.012	0.777±0.037	0.860±0.005
IGML (Diag→Full)	0.977±0.008	0.830±0.029	**0.783**±0.022	0.794±0.042	**0.699**±0.013	**0.790**±0.023	**0.782**±0.014	0.773±0.038	0.856±0.005

methods. Further, usefulness of IGML was also shown through some illustrative examples.

ACKNOWLEDGMENTS

This work was supported by MEXT KAKENHI to I.T. (16H06538, 17H00758) and M.K. (16H06538, 17H04694); from JST CREST awarded to I.T. (JPMJCR1302, JPMJCR1502) and PRESTO awarded to M.K. (JPMJPR15N2); from the Collaborative Research Program of Institute for Chemical Research, Kyoto University to M.K. (grant #2018-33); from the MI2I project of the Support Program for Starting Up Innovation Hub from JST awarded to I.T., and M.K.; and from RIKEN Center for AIP awarded to I.T.

REFERENCES

[1] B. Adhikari, Y. Zhang, N. Ramakrishnan, and B. A. Prakash. 2018. Sub2Vec: Feature Learning for Subgraphs. In *PAKDD*. Springer, 170–182.
[2] J. Atwood and D. Towsley. 2016. Diffusion-convolutional neural networks. In *Advances in NIPS*. 1993–2001.
[3] A. Bellet, A. Habrard, and M. Sebban. 2012. Good edit similarity learning by loss minimization. *Machine Learning* 89, 1-2 (2012), 5–35.
[4] K. M. Borgwardt and H.-P. Kriegel. 2005. Shortest-path kernels on graphs. In *Proc. of the 5th ICDM*. IEEE, 8–pp.
[5] H. Cheng, X. Yan, J. Han, and P. S. Yu. 2008. Direct Discriminative Pattern Mining for Effective Classification. In *Proc. of 24th ICDE*. 169–178.
[6] F. Costa and K. D. Grave. 2010. Fast neighborhood subgraph pairwise distance kernel. In *Proc. of the 27th ICML*. Omnipress, 255–262.
[7] J. V. Davis, B. Kulis, P. Jain, S. Sra, and I. S. Dhillon. 2007. Information-theoretic metric learning. In *Proc. of the 24th ICML*. ACM, 209–216.
[8] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. 2015. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In *Advances in NIPS*. Curran Associates, Inc., 2224–2232.
[9] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. 2008. LIBLINEAR: A Library for Large Linear Classification. *JMLR* 9 (2008), 1871–1874.
[10] A. Feragen, N. Kasenburg, J. Petersen, M. de Bruijne, and K. Borgwardt. 2013. Scalable kernels for graphs with continuous attributes. In *Advances in NIPS*. 216–224.
[11] J. Friedman, T. Hastie, H. Höfling, and R. Tibshirani. 2007. Pathwise coordinate optimization. *The Annals of Applied Statistics* 1, 2 (12 2007), 302–332.
[12] T. Gärtner, P. Flach, and S. Wrobel. 2003. On graph kernels: Hardness results and efficient alternatives. In *Learning Theory and Kernel Machines*. Springer, 129–143.
[13] L. E. Ghaoui, V. Viallon, and T. Rabbani. 2010. Safe feature elimination for the lasso and sparse supervised learning problems. *arXiv:1009.4219* (2010).
[14] K. Kersting, N. M. Kriege, C. Morris, P. Mutzel, and M. Neumann. 2016. Benchmark Data Sets for Graph Kernels. http://graphkernels.cs.tu-dortmund.de.
[15] R. Kondor and K. M. Borgwardt. 2008. The skew spectrum of graphs. In *Proc. of the 25th ICML*. ACM, 496–503.
[16] R. Kondor and H. Pan. 2016. The multiscale laplacian graph kernel. In *Advances in NIPS*. 2990–2998.
[17] R. Kondor, N. Shervashidze, and K. M. Borgwardt. 2009. The graphlet spectrum. In *Proc. of the 26th ICML*. ACM, 529–536.
[18] N. Kriege and P. Mutzel. 2012. Subgraph matching kernels for attributed graphs. *arXiv preprint arXiv:1206.6483* (2012).
[19] J. B. Lee, R. Rossi, and X. Kong. 2018. Graph classification using structural attention. In *Proc. of the 24th KDD*. ACM, 1666–1674.
[20] D. Li and Y. Tian. 2018. Survey and experimental study on metric learning methods. *Neural Networks* 105 (2018), 447 – 462.
[21] C. Morris, N. M. Kriege, K. Kersting, and P. Mutzel. 2016. Faster kernels for graphs with continuous attributes via hashing. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*. IEEE, 1095–1100.
[22] M. L. Morvan and J.-P. Vert. 2018. WHInter: A Working set algorithm for High-dimensional sparse second order Interaction models. In *Proc. of the 35th ICML*, Vol. 80. PMLR, 3635–3644.
[23] K. Nakagawa, S. Suzumura, M. Karasuyama, K. Tsuda, and I. Takeuchi. 2016. Safe pattern pruning: An efficient approach for predictive pattern mining. In *Proc. of the 22nd KDD*. ACM, 1785–1794.
[24] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal. 2017. graph2vec: Learning Distributed Representations of Graphs. In *Proc. of 13th MLGWorkshop*.
[25] M. Neuhaus and H. Bunke. 2007. Automatic learning of cost functions for graph edit distance. *Information Sciences* 177, 1 (2007), 239–247.
[26] P. C. Nguyen, K. Ohara, A. Mogi, H. Motoda, and T. Washio. 2006. Constructing decision trees for graph-structured data by chunkingless graph-based induction. In *PAKDD*. Springer, 390–399.
[27] M. Niepert, M. Ahmed, and K. Kutzkov. 2016. Learning convolutional neural networks for graphs. In *Proc. of the 33rd ICML*. 2014–2023.
[28] P. K. Novak, N. Lavrač, and G. I. Webb. 2009. Supervised Descriptive Rule Discovery: A Unifying Survey of Contrast Set, Emerging Pattern and Subgroup Mining. *JRML* 10 (2009), 377–403.
[29] F. Orsini, P. Frasconi, and L. De Raedt. 2015. Graph invariant kernels. In *Proc. of the 24th IJCAI*. 3756–3762.
[30] H. Saigo, S. Nowozin, T. Kadowaki, T. Kudo, and K. Tsuda. 2009. gBoost: a mathematical programming approach to graph classification and regression. *Machine Learning* 75, 1 (2009), 69–89.
[31] N. Shervashidze and K. M. Borgwardt. 2009. Fast subtree kernels on graphs. In *Advances in NIPS*. 1660–1668.
[32] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt. 2011. Weisfeiler-lehman graph kernels. *JMLR* 12, Sep (2011), 2539–2561.
[33] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt. 2009. Efficient graphlet kernels for large graph comparison. In *AIStats*. 488–495.
[34] M. Simonovsky and N. Komodakis. 2017. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proc. CVPR*.
[35] Y. Su, F. Han, R. E. Harang, and X. Yan. 2016. A fast kernel for attributed graphs. In *Proc. of the 2016 SDM*. SIAM, 486–494.
[36] M. Sugiyama and K. Borgwardt. 2015. Halting in random walk kernels. In *Advances in NIPS*. 1639–1647.
[37] M. Thoma, H. Cheng, A. Gretton, J. Han, H.-P. Kriegel, A. Smola, L. Song, P. S. Yu, X. Yan, and K. M. Borgwardt. 2010. Discriminative frequent subgraph mining with optimality guarantees. *Stat. Anal. Data Min.* 3, 5 (2010), 302–318.
[38] A. J.-P. Tixier, G. Nikolentzos, P. Meladianos, and M. Vazirgiannis. 2018. Graph Classification with 2D Convolutional Neural Networks. (2018).
[39] S. Verma and Z.-L. Zhang. 2017. Hunt For The Unique, Stable, Sparse And Fast Feature Learning On Graphs. In *Advances in NIPS*. 88–98.
[40] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt. 2010. Graph kernels. *JMLR* 11, Apr (2010), 1201–1242.
[41] K. Q. Weinberger and L. K. Saul. 2009. Distance metric learning for large margin nearest neighbor classification. *JMLR* 10, Feb (2009), 207–244.
[42] X. Yan and J. Han. 2002. gspan: Graph-based substructure pattern mining. In *Proc. of the 2nd ICDM*. IEEE, 721–724.
[43] P. Yanardag and S. Vishwanathan. 2015. Deep graph kernels. In *Proc. of the 21th KDD*. ACM, 1365–1374.
[44] T. Yoshida, I. Takeuchi, and M. Karasuyama. 2018. Safe Triplet Screening for Distance Metric Learning. In *Proc. of the 24th KDD*. 2653–2662.
[45] M. Zhang, Z. Cui, M. Neumann, and Y. Chen. 2018. An end-to-end deep learning architecture for graph classification. In *Proc. of 32nd AAAI*.
[46] Z. Zhang, M. Wang, Y. Xiang, Y. Huang, and A. Nehorai. 2018. RetGK: Graph Kernels based on Return Probabilities of Random Walks. In *Advances in NIPS*. 3968–3978.

Appendix

A DUAL PROBLEM

The primal problem (2) can be re-written as

$$\min_{\boldsymbol{m},\boldsymbol{z}} \sum_{i\in[n]}[\sum_{l\in\mathcal{D}_i}\ell_L(z_{il})+\sum_{j\in\mathcal{S}_i}\ell_{-U}(z_{ij})]+\lambda R(\boldsymbol{m})$$
$$\text{s.t. } \boldsymbol{m}\geq \boldsymbol{0}, z_{il}=\boldsymbol{m}^\top \boldsymbol{c}_{il},\ z_{ij}=-\boldsymbol{m}^\top \boldsymbol{c}_{ij}.$$

The Lagrange function $\mathcal{L}$ is

$$\mathcal{L}(\boldsymbol{m},z,\boldsymbol{\alpha},\boldsymbol{\beta}) := \sum_{i\in[n]}[\sum_{l\in\mathcal{D}_i}\ell_L(z_{il})+\sum_{j\in\mathcal{S}_i}\ell_{-U}(z_{ij})]+\lambda R(\boldsymbol{m})$$
$$+\sum_{i\in[n]}[\sum_{l\in\mathcal{D}_i}\alpha_{il}(z_{il}-\boldsymbol{m}^\top\boldsymbol{c}_{il})+\sum_{j\in\mathcal{S}_i}\alpha_{ij}(z_{ij}+\boldsymbol{m}^\top\boldsymbol{c}_{ij})]-\boldsymbol{\beta}^\top\boldsymbol{m},$$

where $\boldsymbol{\alpha}\in\mathbb{R}^{2nK}$ and $\boldsymbol{\beta}\in\mathbb{R}^p_+$ are Lagrange multipliers. The dual function D_λ is then

$$D_\lambda(\boldsymbol{\alpha},\boldsymbol{\beta}) := \inf_{\boldsymbol{m},\boldsymbol{z}} \mathcal{L}(\boldsymbol{m},\boldsymbol{z},\boldsymbol{\alpha},\boldsymbol{\beta}). \tag{20}$$

In the definition of dual function (20), in order to minimize $\mathcal{L}$ with respect to $\boldsymbol{m}$, by partially differentiating $\mathcal{L}$, we obtain

$$\nabla_{\boldsymbol{m}}\mathcal{L}=\lambda(\boldsymbol{1}+\eta\boldsymbol{m})+\sum_{i\in[n]}[-\sum_{l\in\mathcal{D}_i}\alpha_{il}\boldsymbol{c}_{il}+\sum_{j\in\mathcal{S}_i}\alpha_{ij}\boldsymbol{c}_{ij}]-\boldsymbol{\beta}=\boldsymbol{0}. \tag{21}$$

The convex conjugate function of ℓ_t is

$$\ell_t^*(-\alpha_{ij})=\sup_{z_{ij}}\{(-\alpha_{ij})z_{ij}-\ell_t(z_{ij})\}, \tag{22}$$

which can be written as

$$\ell_t^*(x_*)=\frac{1}{4}x_*^2+tx_*, (x_*\leq 0). \tag{23}$$

From equations (21), (22) and (23), the dual fuction can be written as

$$D_\lambda(\boldsymbol{\alpha},\boldsymbol{\beta})$$
$$=-\sum_{i\in[n]}[\sum_{l\in\mathcal{D}_i}\ell_L^*(-\alpha_{il})+\sum_{j\in\mathcal{S}_i}\ell_{-U}^*(-\alpha_{ij})]-\frac{\lambda\eta}{2}\|\boldsymbol{m}_\lambda(\boldsymbol{\alpha},\boldsymbol{\beta})\|_2^2$$
$$=-\frac{1}{4}\|\boldsymbol{\alpha}\|_2^2+\boldsymbol{t}^\top\boldsymbol{\alpha}-\frac{\lambda\eta}{2}\|\boldsymbol{m}_\lambda(\boldsymbol{\alpha},\boldsymbol{\beta})\|_2^2.$$

where

$$\boldsymbol{m}_\lambda(\boldsymbol{\alpha},\boldsymbol{\beta}) := \frac{1}{\lambda\eta}\left[\boldsymbol{\beta}+\sum_{i\in[n]}(\sum_{l\in\mathcal{D}_i}\alpha_{il}\boldsymbol{c}_{il}-\sum_{j\in\mathcal{S}_i}\alpha_{ij}\boldsymbol{c}_{ij})-\lambda\boldsymbol{1}\right]$$
$$=\frac{1}{\lambda\eta}[\boldsymbol{\beta}+\boldsymbol{C}\boldsymbol{\alpha}-\lambda\boldsymbol{1}].$$

Therefore, although dual problem can be written as

$$\max_{\boldsymbol{\alpha}\geq\boldsymbol{0},\boldsymbol{\beta}\geq\boldsymbol{0}} D(\boldsymbol{\alpha},\boldsymbol{\beta}),$$

by maximizing $D(\boldsymbol{\alpha},\boldsymbol{\beta})$ with respect to $\boldsymbol{\beta}$, we obtain more simple dual problem (3).

We obtain $\alpha_{ij}=-\ell_t'(z_{ij})$, used in (5), from derivative of $\mathcal{L}$ with respect to z_{ij}.

B PROOF OF LEMMA 2.1

From equation (10), the value of $(x_{i,k'}-x_{j,k'})^2$ is bounded as follows:

$$(x_{i,k'}-x_{j,k'})^2\leq \max_{0\leq x_{i,k'}\leq x_{i,k},0\leq x_{j,k'}\leq x_{j,k}}(x_{i,k'}-x_{j,k'})^2$$
$$=\max\{x_{i,k},x_{j,k}\}^2.$$

Using this inequality, the inner product $C_{k',:}\boldsymbol{q}$ is also bounded:

$$C_{k',:}\boldsymbol{q}=\sum_{i\in[n]}\Big[\sum_{l\in\mathcal{D}_i}q_{il}(x_{i,k'}-x_{l,k'})^2-\sum_{j\in\mathcal{S}_i}q_{ij}(x_{i,k'}-x_{j,k'})^2\Big]$$
$$\leq\sum_{i\in[n]}\sum_{l\in\mathcal{D}_i}q_{il}\max\{x_{i,k},x_{l,k}\}^2.$$

Similarly, the norm $\|C_{k',:}\|_2$ is bounded:

$$\|C_{k',:}\|_2=\sqrt{\sum_{i\in[n]}[\sum_{l\in\mathcal{D}_i}(x_{i,k'}-x_{l,k'})^4+\sum_{j\in\mathcal{S}_i}(x_{i,k'}-x_{j,k'})^4]}$$
$$\leq\sqrt{\sum_{i\in[n]}[\sum_{l\in\mathcal{D}_i}\max\{x_{i,k},x_{l,k}\}^4+\sum_{j\in\mathcal{S}_i}\max\{x_{i,k},x_{j,k}\}^4]}.$$

Therefore, $C_{k',:}\boldsymbol{q}+r\|C_{k',:}\|_2$ is bounded by Prune$(k|\boldsymbol{q},r)$.

C PROOF OF THEOREM 2.1 (DGB)

From 1/2-strong convexity of $-D_\lambda(\boldsymbol{\alpha})$, for any $\boldsymbol{\alpha}\geq\boldsymbol{0}$ and $\boldsymbol{\alpha}^\star\geq\boldsymbol{0}$, we see

$$D_\lambda(\boldsymbol{\alpha})\leq D_\lambda(\boldsymbol{\alpha}^\star)+\nabla D_\lambda(\boldsymbol{\alpha}^\star)^\top(\boldsymbol{\alpha}-\boldsymbol{\alpha}^\star)-\frac{1}{4}\|\boldsymbol{\alpha}-\boldsymbol{\alpha}^\star\|_2^2. \tag{24}$$

Applying weak duality $P_\lambda(\boldsymbol{m})\geq D_\lambda(\boldsymbol{\alpha}^\star)$ and the optimality condition of the dual problem $\nabla D_\lambda(\boldsymbol{\alpha}^\star)^\top(\boldsymbol{\alpha}-\boldsymbol{\alpha}^\star)\leq 0$ to (24), we obtain DGB.

D PROOF OF THEOREM 2.2 (RPB)

From the optimality condition on dual problem (3),

$$\nabla_{\boldsymbol{\alpha}}D_{\lambda_0}(\boldsymbol{\alpha}_0^\star)^\top(\frac{\lambda_0}{\lambda_1}\boldsymbol{\alpha}_1^\star-\boldsymbol{\alpha}_0^\star)\leq 0, \tag{25}$$

$$\nabla_{\boldsymbol{\alpha}}D_{\lambda_1}(\boldsymbol{\alpha}_1^\star)^\top(\frac{\lambda_1}{\lambda_0}\boldsymbol{\alpha}_0^\star-\boldsymbol{\alpha}_1^\star)\leq 0. \tag{26}$$

Here, the gradient vector at optimal solution is

$$\nabla D_{\lambda_i}(\boldsymbol{\alpha}_i^\star)=-\frac{1}{2}\boldsymbol{\alpha}_i^\star+\boldsymbol{t}-\boldsymbol{C}^\top\boldsymbol{m}_{\lambda_i}(\boldsymbol{\alpha}_i^\star)$$
$$=-\frac{1}{2}\boldsymbol{\alpha}_i^\star+\boldsymbol{t}-\boldsymbol{C}^\top\boldsymbol{m}_i^\star,$$

thus, by substituting this equation into (25) and (26),

$$(-\frac{1}{2}\boldsymbol{\alpha}_0^\star+\boldsymbol{t}-\boldsymbol{C}^\top\boldsymbol{m}_0^\star)^\top(\frac{\lambda_0}{\lambda_1}\boldsymbol{\alpha}_1^\star-\boldsymbol{\alpha}_0^\star)\leq 0, \tag{27}$$

$$(-\frac{1}{2}\boldsymbol{\alpha}_1^\star+\boldsymbol{t}-\boldsymbol{C}^\top\boldsymbol{m}_1^\star)^\top(\frac{\lambda_1}{\lambda_0}\boldsymbol{\alpha}_0^\star-\boldsymbol{\alpha}_1^\star)\leq 0. \tag{28}$$

From $\lambda_1\times(27)+\lambda_0\times(28)$,

$$(-\frac{1}{2}[\boldsymbol{\alpha}_0^\star-\boldsymbol{\alpha}_1^\star]-\boldsymbol{C}^\top[\boldsymbol{m}_0^\star-\boldsymbol{m}_1^\star])^\top(\lambda_0\boldsymbol{\alpha}_1^\star-\lambda_1\boldsymbol{\alpha}_0^\star)\leq 0. \tag{29}$$

From equation (21),

$$\boldsymbol{C}\boldsymbol{\alpha}_i=\lambda_i\eta\boldsymbol{m}_i+\lambda_i\boldsymbol{1}-\boldsymbol{\beta}_i. \tag{30}$$

By substituting equation (30) into equation (29),

$$-\frac{1}{2}[\boldsymbol{\alpha}_0^\star-\boldsymbol{\alpha}_1^\star]^\top(\lambda_0\boldsymbol{\alpha}_1^\star-\lambda_1\boldsymbol{\alpha}_0^\star)$$
$$-[\boldsymbol{m}_0^\star-\boldsymbol{m}_1^\star]^\top(\lambda_0\lambda_1\eta[\boldsymbol{m}_1-\boldsymbol{m}_0]-\lambda_0\boldsymbol{\beta}_1^\star+\lambda_1\boldsymbol{\beta}_0^\star)\leq 0.$$

Algorithm 4: General Working-Set Method

```
1 initialize $x_0 \in \mathcal{D}$
2 for $t = 1, 2, \dots$ until converged do
3     $\mathcal{W}_t = \{j \mid h_j(x_{t-1}) \geq 0\}$
4     $x_t = \arg\min_{x \in \mathcal{D}} f(x)$ s.t. $h_j(x) \leq 0, \forall j \in \mathcal{W}_t$
```

Transforming this inequality based on completing the square with the complementary condition $\boldsymbol{m}_i^{\star\top}\boldsymbol{\beta}_i^\star = 0$ and $\boldsymbol{m}_1^{\star\top}\boldsymbol{\beta}_0^\star, \boldsymbol{m}_0^{\star\top}\boldsymbol{\beta}_1^\star \geq 0$, we obtain

$$\left\|\boldsymbol{\alpha}_1^\star - \frac{\lambda_0+\lambda_1}{2\lambda_0}\boldsymbol{\alpha}_0^\star\right\|_2^2 + 2\lambda_1\eta\|\boldsymbol{m}_0^\star - \boldsymbol{m}_1^\star\|_2^2 \leq \left\|\frac{\lambda_0-\lambda_1}{2\lambda_0}\boldsymbol{\alpha}_0^\star\right\|_2^2.$$

By using $\|\boldsymbol{m}_0^\star - \boldsymbol{m}_1^\star\|_2^2 \geq 0$ to this inequality, we obtain RPB.

E PROOF OF THEOREM 2.3 (RRPB)

Considering a hypersphere that expands the RPB radius by $\frac{\lambda_0+\lambda_1}{2\lambda_0}\epsilon$ and replaces the RPB center with $\frac{\lambda_0+\lambda_1}{2\lambda_0}\boldsymbol{\alpha}_0$, we obtain

$$\left\|\boldsymbol{\alpha}_1^\star - \frac{\lambda_0+\lambda_1}{2\lambda_0}\boldsymbol{\alpha}_0\right\|_2 \leq \frac{|\lambda_0-\lambda_1|}{2\lambda_0}\|\boldsymbol{\alpha}_0^\star\|_2 + \frac{\lambda_0+\lambda_1}{2\lambda_0}\epsilon.$$

Since ϵ is defined by $\|\boldsymbol{\alpha}_0^\star - \boldsymbol{\alpha}_0\|_2 \leq \epsilon$, this sphere covers any RPB made by $\boldsymbol{\alpha}_0^\star$ which satisfies $\|\boldsymbol{\alpha}_0^\star - \boldsymbol{\alpha}_0\|_2 \leq \epsilon$. Using the reverse triangle inequality

$$\|\boldsymbol{\alpha}_0^\star\|_2 - \|\boldsymbol{\alpha}_0\|_2 \leq \|\boldsymbol{\alpha}_0^\star - \boldsymbol{\alpha}_0\|_2 \leq \epsilon,$$

the following is obtained.

$$\left\|\boldsymbol{\alpha}_1^\star - \frac{\lambda_0+\lambda_1}{2\lambda_0}\boldsymbol{\alpha}_0\right\|_2 \leq \frac{|\lambda_0-\lambda_1|}{2\lambda_0}(\|\boldsymbol{\alpha}_0\|_2 + \epsilon) + \frac{\lambda_0+\lambda_1}{2\lambda_0}\epsilon.$$

By arranging this, RRPB is obtained.

F SKETCH OF PROOF FOR THEOREM 2.6 (RSS) AND 2.7 (RSP)

When $\lambda_1 = \lambda$ is set in RRPB, the center and the radius of the bound $\mathcal{B} = \{\boldsymbol{\alpha} \mid \|\boldsymbol{\alpha} - \boldsymbol{q}\|_2^2 \leq r^2\}$ are $\boldsymbol{q} = \frac{\lambda_0+\lambda}{2\lambda_0}\boldsymbol{\alpha}_0$ and $r = \left\|\frac{\lambda_0-\lambda}{2\lambda_0}\boldsymbol{\alpha}_0\right\|_2 + \left(\frac{\lambda_0+\lambda}{2\lambda_0} + \frac{|\lambda_0-\lambda|}{2\lambda_0}\right)\epsilon$. Substituting these $\boldsymbol{q}$ and r into (12) and (13), respectively, and arranging them, we can obtain the range in which screening and pruning condition hold.

G PROOF OF THEOREM 2.8 (CONVERGENCE OF WS)

By introducing new variable $\boldsymbol{s}$, the dual problem (3) can be written as

$$\max_{\boldsymbol{\alpha} \geq \boldsymbol{0}, \boldsymbol{s} \geq \boldsymbol{0}} -\frac{1}{4}\|\boldsymbol{\alpha}\|^2 + \boldsymbol{t}^\top\boldsymbol{\alpha} - \frac{1}{2\lambda\eta}\|\boldsymbol{s}\|^2$$
$$\text{s.t. } \boldsymbol{C}\boldsymbol{\alpha} - \lambda\boldsymbol{1} - \boldsymbol{s} \leq \boldsymbol{0}.$$

We prove the convergence of working-set method on more general convex problem as follows:

$$\boldsymbol{x}^\star := \arg\min_{\boldsymbol{x} \in \mathcal{D}} f(\boldsymbol{x}) \text{ s.t. } h_i(\boldsymbol{x}) \leq 0, \forall i \in [n], \quad (31)$$

where $f(\boldsymbol{x})$ is γ-strong convex function ($\gamma > 0$). Here, as shown in Algorithm 4, working set is defined by $\mathcal{W}_t = \{j \mid h_j(\boldsymbol{x}_{t-1}) \geq 0\}$ at every iteration. Then, the updated working set includes all the violated constraints and the constraints on the boundary. We show that Algorithm 4 finishes with finite T-step and returns optimal solution $\boldsymbol{x}_T = \boldsymbol{x}^\star$.

Proof. Since f is γ-strong convex from the assumption, the following inequality holds.

$$f(\boldsymbol{x}_{t+1}) \geq f(\boldsymbol{x}_t) + \nabla f(\boldsymbol{x}_t)^\top(\boldsymbol{x}_{t+1} - \boldsymbol{x}_t) + \frac{\gamma}{2}\|\boldsymbol{x}_{t+1} - \boldsymbol{x}_t\|^2. \quad (32)$$

At the step t, the problem can be written as follows using only the active constraint at the optimal solution $\boldsymbol{x}_t$.

$$\begin{aligned}\boldsymbol{x}_t &= \arg\min_{\boldsymbol{x} \in \mathcal{D}} f(\boldsymbol{x}) \text{ s.t. } h_i(\boldsymbol{x}) \leq 0, \forall i \in \mathcal{W}_t \\ &= \arg\min_{\boldsymbol{x} \in \mathcal{D}} f(\boldsymbol{x}) \text{ s.t. } h_i(\boldsymbol{x}) \leq 0, \forall i \in \{j \in \mathcal{W}_t \mid h_j(\boldsymbol{x}_t) = 0\}\end{aligned} \quad (33)$$

From the definition of $\mathcal{W}_t$, the working set $\mathcal{W}_{t+1}$ must contain all the active constraints $\{j \in \mathcal{W}_t \mid h_j(\boldsymbol{x}_t) = 0\}$ at the step t and can contain other constraints which are not included in $\mathcal{W}_t$. This means that $\boldsymbol{x}_{t+1}$ must be in the feasible region of the optimization problem at the step t (33):

$$\mathcal{F} := \left\{\boldsymbol{x} \in \mathcal{D} \mid h_i(\boldsymbol{x}) \leq 0, \forall i \in \{j \in \mathcal{W}_t \mid h_j(\boldsymbol{x}_t) = 0\}\right\}$$

Therefore, from the optimality condition of optimization problem (33),

$$\nabla f(\boldsymbol{x}_t)^\top(\boldsymbol{x}_{t+1} - \boldsymbol{x}_t) \geq 0, \boldsymbol{x}_{t+1} \in \mathcal{F}. \quad (34)$$

From inequality (32) and inequality (34), we obtain

$$f(\boldsymbol{x}_{t+1}) \geq f(\boldsymbol{x}_t) + \frac{\gamma}{2}\|\boldsymbol{x}_{t+1} - \boldsymbol{x}_t\|^2.$$

If $\boldsymbol{x}_t$ is not optimal, there exists at least one violated constraint $h_{j'}(\boldsymbol{x}_t) > 0$ for some j' because otherwise $\boldsymbol{x}_t$ is optimal. Then, we see $\boldsymbol{x}_{t+1} \neq \boldsymbol{x}_t$ because $\boldsymbol{x}_{t+1}$ should satisfy the constraint $h_{j'}(\boldsymbol{x}_t) \leq 0$. If $\boldsymbol{x}_t \neq \boldsymbol{x}_{t+1}$, by using $\|\boldsymbol{x}_{t+1} - \boldsymbol{x}_t\|^2 > 0$,

$$f(\boldsymbol{x}_{t+1}) \geq f(\boldsymbol{x}_t) + \frac{\gamma}{2}\|\boldsymbol{x}_{t+1} - \boldsymbol{x}_t\|^2 > f(\boldsymbol{x}_t).$$

Thus, the objective function always strictly increases ($f(\boldsymbol{x}_t) < f(\boldsymbol{x}_{t+1})$). This indicates that the algorithm never encounters the same working set $\mathcal{W}_t$ as the set of other iterations $t' \neq t$. For any step t, the optimal value $f(\boldsymbol{x}_t)$ with a subset of the original constraints $\mathcal{W}_t$ must be smaller than or equal to the optimal value $f(\boldsymbol{x}^\star)$ of original problem (31) having all constraints. Therefore, $f(\boldsymbol{x}_t) \leq f(\boldsymbol{x}^\star)$ is satisfied, and we obtain $f(\boldsymbol{x}_T) = f(\boldsymbol{x}^\star)$ at some finite step T. □