Generalized Sparse Learning of Linear Models over the Complete Subgraph Feature Set

Ichigaku Takigawa, *Member, IEEE,*
and Hiroshi Mamitsuka, *Senior Member, IEEE*

Abstract—Supervised learning over graphs is an intrinsically difficult problem: simultaneous learning of relevant features from the complete subgraph feature set, in which enumerating all subgraph features occurring in given graphs is practically intractable due to combinatorial explosion. We show that 1) existing graph supervised learning studies, such as Adaboost, LPBoost, and LARS/LASSO, can be viewed as variations of a branch-and-bound algorithm with simple bounds, which we call Morishita-Kudo bounds; 2) We present a direct sparse optimization algorithm for generalized problems with arbitrary twice-differentiable loss functions, to which Morishita-Kudo bounds cannot be directly applied; 3) We experimentally showed that i) our direct optimization method improves the convergence rate and stability, and ii) L1-penalized logistic regression (L1-LogReg) by our method identifies a smaller subgraph set, keeping the competitive performance, iii) the learned subgraphs by L1-LogReg are more size-balanced than competing methods, which are biased to small-sized subgraphs.

Index Terms—supervised learning for graphs, graph mining, sparsity-inducing regularization, block coordinate gradient descent, simultaneous feature learning

1 Introduction

We consider the problem of modeling the response $y \in \mathcal{Y}$ to an input graph $g \in \mathcal{G}$ as $y \approx \mu(g)$ with a model function μ from n given observations

$$\{(g_1, y_1), (g_2, y_2), \ldots, (g_n, y_n)\}, \quad g_i \in \mathcal{G}, \quad y_i \in \mathcal{Y}, \qquad (1)$$

where $\mathcal{G}$ is a set of all finite-size, connected, node-and-edge-discretely-labeled, undirected graphs, and $\mathcal{Y}$ is a label space, i.e. a set of real numbers $\mathbb{R}$ for regression and a set of nominal or binary values such as $\{T, F\}$ or $\{-1, 1\}$ for classification as seen in previous work [1], [2], [3], [4]. This problem arises in computer vision for images and 3D shapes [5], [6], [7], [8], in natural language processing [1], and in bioinformatics for QSAR analysis [9], virtual drug screening [10], nucleotide or amino-acid sequences [11], sugar chains or glycans [12], [13], RNA secondary structures [14], protein 3D structures [15], and biological networks [16].

For Problem (1), the most widely-used approach would be *graph kernel method*. So far various types of graph kernels have been developed and used in various applications successfully [17], [18], [19], [20], [21], [22]. However [18] showed that the all subgraph kernel over *the complete subgraph feature set* — all possible subgraph features occurring in the given data — is infeasible in practice. Hence, any practical graph kernel restricts the subgraph features to some limited types such as paths and trees, bounded-size subgraphs, or heuristically inspired subgraph features in individual applications. Another approach for directly generating feature vectors as 'fingerprint' comes from chemoinformatics, such as extended connectivity fingerprints (ECFP) [23], frequent subgraphs [24], and bounded-size graph fingerprint [25], which also limits the subgraph types.

In contrast, a series of inspiring studies have been made for simultaneous learning of relevant features from the complete subgraph feature set [1], [2], [3], [4], [26], [27], [28] , where however enumerating all subgraph features occurring in given graphs is practically intractable due to combinatorial explosion. Triggered by the seminal work [1], it has been shown that we can perform the simultaneous feature learning for Adaboost [1], LARS/LASSO [2], sparse PLS regression [4], sparse PCA [28], and LPBoost [3].

In terms of sparse learning over the complete subgraph feature set, the contributions of this paper are the following three folds:

- We show that existing graph supervised-learning approaches in the literature can be viewed as variations of the branch-and-bound algorithm with a simple bound, which we call Morishita-Kudo bounds, working for any *separable* target function. This can be a simple and useful criterion to define the types of optimization solvable over the the complete subgraph feature set, which is intractably large in practice. **(Section 3)**
- We present a direct optimization algorithm for the following graph version of a generalized problem with the target function with *non-separable* terms; This means that the previous branch-and-bound strategy with Morishita-Kudo bounds cannot be directly used:

$$\min_{\beta, \beta_0} \sum_{i=1}^{n} L\big(y_i, \mu(g_i; \beta, \beta_0)\big) + \lambda_1 \|\beta\|_1 + \frac{\lambda_2}{2} \|\beta\|_2^2, \qquad (2)$$

  for a linear model over all indicators of all possible subgraph x_i

$$\mu(g) := \beta_0 + \sum_{j=1}^{\infty} \beta_j I(x_j \subseteq g), \quad \beta := (\beta_1, \beta_2, \ldots), \quad (3)$$

  where L is a twice differentiable loss function and $\lambda_1 > 0, \lambda_2 \geqslant 0$, and we assume the *sparsity* on coefficient parameters: Most coefficients $\beta_1, \beta_2, \ldots$ are zero and a few of them are nonzero. **(Section 4)**
- We experimentally show that (i) our direct optimization improves the convergence rate and stability; (ii) L1-penalized logistic regression by our method (L1-LogReg) identifies a smaller subgraph set than competing methods (including LPBoost), keeping competitive performance; (iii) the subgraphs learned by L1-LogReg are relatively size-balanced, while those by boosting methods are biased to only small-size subgraphs, implying easier interpretability of obtained subgraphs by L1-LogReg. **(Section 5)**

2 Preliminaries

2.1 Notations

$I(A)$ is a binary indicator function of an event A, meaning that $I(A) = 1$ if A is true; otherwise $I(A) = 0$. The notation $x \subseteq g$ denotes the subgraph isomorphism that g contains a subgraph that is isomorphic to x. Hence the subgraph indicator $I(x \subseteq g) = 1$ if $x \subseteq g$; otherwise 0.

Given a set of n graphs, $\mathcal{G}_n := \{g_i\}_{i=1}^n$, we define the union of all subgraphs of $g \in \mathcal{G}_n$ as

$$\mathcal{X}(\mathcal{G}_n) := \{x \in \mathcal{G} \mid x \subseteq g, g \in \mathcal{G}_n\}.$$

- *I. Takigawa is with the Graduate School of Information Science and Technology, Hokkaido University, and PRESTO, Japan Science and Technology Agency (JST). E-mail: takigawa@ist.hokudai.ac.jp*
- *H. Mamitsuka is with Institute for Chemical Research, Kyoto University, Japan and Department of Computer Science, Aalto University, Finland. E-mail: mami@kuicr.kyoto-u.ac.jp*

It is important to note that $\mathcal{X}(\mathcal{G}_n)$ is a finite set and is equal to *the complete subgraph feature set* of $\mathcal{G}_n$ as

$$\mathcal{X}(\mathcal{G}_n) = \{x \in \mathcal{G} \mid \exists g \in \mathcal{G}_n \text{ such that } x \subseteq g\}.$$

For given $\mathcal{X}(\mathcal{G}_n)$, we can construct an *enumeration tree* $\mathcal{T}(\mathcal{G}_n)$ over $\mathcal{X}(\mathcal{G}_n)$ that is described in Section 2.2. Also we write a subtree of $\mathcal{T}(\mathcal{G}_n)$ rooted at $x \in \mathcal{X}(\mathcal{G}_n)$ as $\mathcal{T}(x)$.

For given $\mathcal{G}_n$ and a subgraph feature x, we define the *characteristic vector* of x over $\mathcal{G}_n$ as

$$I_{\mathcal{G}_n}(x) := (I(x \subseteq g_1), I(x \subseteq g_2), \ldots, I(x \subseteq g_n)). \tag{4}$$

From the definition, $I_{\mathcal{G}_n}(x)$ is an n-dimensional Boolean vector, i.e., $I_{\mathcal{G}_n}(x) \in \{0,1\}^n$.

For an n-dimensional Boolean vector $u := (u_1, u_2, \ldots, u_n) \in \{0,1\}^n$, we write the index set of nonzero elements and that of zero elements as

$$1(u) := \{i \mid u_i = 1\} \subseteq \{1, 2, \ldots, n\},$$
$$0(u) := \{i \mid u_i = 0\} \subseteq \{1, 2, \ldots, n\}.$$

From the definition, we have $1(u) \cup 0(u) = \{1, 2, \ldots, n\}$ and $1(u) \cap 0(u) = \varnothing$. For simplicity, we also use the same notation for the characteristic vector of x as

$$1(x) := 1(I_{\mathcal{G}_n}(x)) = \{i \mid x \subseteq g_i, g_i \in \mathcal{G}_n\},$$
$$0(x) := 0(I_{\mathcal{G}_n}(x)) = \{i \mid x \not\subseteq g_i, g_i \in \mathcal{G}_n\},$$

if there is no possibility of confusion.

2.2 Structuring the Search Space

For (2), the model (3) including a countably infinite number of terms can be reduced to a model over intractably large but finite set, i.e., the complete subgraph feature set $\mathcal{X}(\mathcal{G}_n)$:

$$\mu(g) := \beta_0 + \sum_{j=1}^{\infty} \beta_j I(x_j \subseteq g) = \beta_0 + \sum_{x_j \in \mathcal{X}(\mathcal{G}_n)} \beta_j I(x_j \subseteq g).$$

From the definition, $\mathcal{X}(\mathcal{G}_n)$ is equivalent to a set of frequent subgraphs in $\mathcal{G}_n$ having frequency $\geqslant 1$. This fact connects Problem (2) to the problem of enumerating all frequent subgraphs in the given set of graphs—*frequent subgraph mining*—that has been extensively studied in data mining.

To traverse every $x \in \mathcal{X}(\mathcal{G}_n)$, a well-structured search space for $\mathcal{X}(\mathcal{G}_n)$, called *an enumeration tree*, is commonly used in frequent pattern enumeration. Enumeration trees have two nice properties:

1) Isomorphic parent-child relationship: Smaller subgraphs are assigned to levels closer to the root, larger subgraphs to levels closer to leaves. The edge from x_i to x_j implies that x_i and x_j are different by only one edge, and smaller x_i is isomorphic to the subgraph of larger x_j.
2) Spanning tree: Traversal over the entire enumeration tree gives us a set of all subgraphs x_j in $\mathcal{X}(\mathcal{G}_n)$, avoiding any redundancy in checking subgraphs, meaning that the same subgraph is not checked more than once.

This leads to widely-used frequent subgraph mining algorithms such as gSpan [29] and GASTON [30]. Throughout this paper, we use the enumeration tree by the gSpan algorithm as a search space for $\mathcal{X}(\mathcal{G}_n)$.

By using our notation in Section 2.1, the above facts can be formally summarized as follows.

Lemma 1 (Enumeration Tree [31]) *Let $G := (V, E)$ be a graph with a node set $V = \mathcal{X}(\mathcal{G}_n) \cup \{\varnothing\}$ and an edge set $E = \{(x, x') \mid x \subseteq x', x \in V, x' \in V, x$ and x' are different by only one edge$\}$, where $\varnothing$ denotes the empty graph. Then, we can construct a spanning tree $\mathcal{T}(\mathcal{G}_n)$ rooted at $\varnothing$ over G, that is, an enumeration tree for $\mathcal{X}(\mathcal{G}_n)$ that has the following properties.*

1) *The enumeration tree covers all of $x \in \mathcal{X}(\mathcal{G}_n)$, where any of $x \in \mathcal{X}(\mathcal{G}_n)$ is reachable from the root $\varnothing$.*
2) *For a subtree $\mathcal{T}(x)$ rooted at node x, we have $x \subseteq x'$ for any $x' \in \mathcal{T}(x)$.*

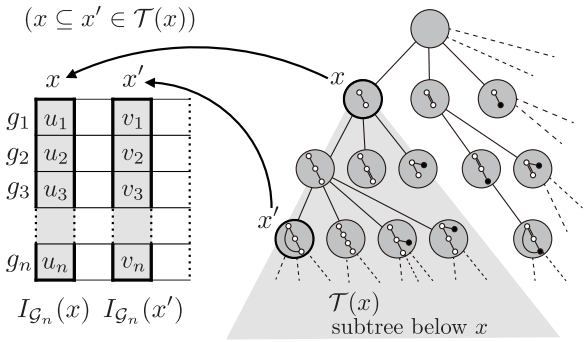

Fig. 1. Boolean vectors $I_{\mathcal{G}_n}(x)$ associated with $x \in \mathcal{T}(\mathcal{G}_n)$. In this example, $x \subseteq x'$. Hence $v_i = 0$ for $u_i = 0$. Only coordinates v_i for $u_i = 1$ can be either 0 or 1, as stated in Lemma 2.

3 Morishita-Kudo Bounds for Separable Functions

Problem (2) is a very typical and general learning problem if we can enumerate all feature subgraphs $\{x_i\} \subset \mathcal{X}(\mathcal{G}_n)$. However $\mathcal{X}(\mathcal{G}_n)$ is intractably large to enumerate all elements in practice. Thus the issue here (in learning from graphs generally) is to efficiently obtain relevant subgraph features $\{x_i\}$ from such intractably large set $\mathcal{X}(\mathcal{G}_n)$. Lemma 1 plays a central role to address this issue.

The existing approaches for this issue are mainly iterative procedures which, at each iteration, search for an "optimal" subgraph from $\mathcal{X}(\mathcal{G}_n)$ in a given criterion. This single optimal-subgraph search is iterated and combined to obtain the final model. In order to perform this search over the complete subgraph feature set in a branch and bound strategy, we need a systematic way to compute the upper and lower pruning bounds for each given criterion. In existing work, these bounds have been derived independently for each specific criterion [1], [2], [3], [4].

Here we show that these specific bounds can be viewed as variations of simple and easy-to-obtain bounds, which we call *Morishita-Kudo bounds*. The original bounds by [1], [32] consider only the specific objective function of Adaboost, but many existing methods for other objectives, such as [2], [3], [4] and the recently proposed gHSIC [33], share the common idea. Thus we can say that most existing approaches are based on a branch-and-bound strategy with Morishita-Kudo bounds. In supplementary materials, we show three examples, in which previously obtained pruning bounds for specific targets can be easily derived by using Morishita-Kudo bounds than the original proofs.

At the same time we note that the objective function of Problem (2), which is thoroughly discussed in the next section, includes non-separable terms, and in particular, the 2-norm penalty term cannot be trivially handled when the complete subgraph feature set is considered.

3.1 Property of Boolean Vectors Associated with $\mathcal{G}_n$

We can associate the characteristic vector $I_{\mathcal{G}_n}(x)$ to each node $x \in \mathcal{T}(\mathcal{G}_n)$ as shown in Figure 1. For these characteristic vectors, we can observe the following result from Lemma 1, widely known as "Apriori property" in frequent pattern mining.

Lemma 2 (Apriori property [34]) $1(I_{\mathcal{G}_n}(x')) \subseteq 1(I_{\mathcal{G}_n}(x))$ *for* $x' \in \mathcal{T}(x)$. *For short,* $1(x') \subseteq 1(x)$ *for* $x' \in \mathcal{T}(x)$.

Remark 3 *Recall that* $I_{\mathcal{G}_n}(x)$, *previously defined as* (4), *is the characteristic vector, an* n*-dimensional Boolean vector, indicating if* x *is contained in each of* $\mathcal{G}_n$. *The number of* 1s *in the vector* $I_{\mathcal{G}_n}(x)$, *that is* $|1(I_{\mathcal{G}_n}(x))|$, *is identical to the "support" of* x *in* $\mathcal{G}_n$ *in the standard data mining terminology.*

Lemma 2 claims that when we traverse an enumeration tree down to the levels closer to leaves, from x to x', the elements taking 1 in $I_{\mathcal{G}_n}(x)$ may change to 0, but the elements taking 0 must remain as 0 in $I_{\mathcal{G}_n}(x')$ (Figure 1). Thus, the number of 1s in the characteristic vector $I_{\mathcal{G}_n}(x)$ at node x monotonically decreases if we proceed to any node $x'(\supseteq x)$ closer to leaves in the enumeration tree. For example, the *anti-monotone property* of the support, which is a fundamental concept in frequent pattern mining, can also be obtained as a corollary of Lemma 2 as $x \subseteq x' \Longrightarrow |1(I_{\mathcal{G}_n}(x))| \geqslant |1(I_{\mathcal{G}_n}(x'))|$.

We also observe the following simple facts for any arbitrary bounded real-valued function on n-dimensional Boolean vector space, $f : \{0,1\}^n \to \mathbb{R}$.

Theorem 4 (Combinatorial bounds) *Assume that* $u \in \{0,1\}^n$ *is fixed. Then, for any* $v \in \{0,1\}^n$ *such that* $1(v) \subseteq 1(u)$, *we have*

$$\overline{f}(u) \geqslant f(v) \geqslant \underline{f}(u),$$

where

$$\overline{f}(u) = \max_{\alpha_i \in \{0,1\}, i \in 1(u)} \{f(\alpha_1, \alpha_2, \ldots, \alpha_n) \mid \alpha_j = 0, j \in 0(u)\},$$
$$\underline{f}(u) = \min_{\alpha_i \in \{0,1\}, i \in 1(u)} \{f(\alpha_1, \alpha_2, \ldots, \alpha_n) \mid \alpha_j = 0, j \in 0(u)\}.$$

Proof. Since $1(v) \subseteq 1(u) \Leftrightarrow 0(u) \subseteq 0(v)$, we have $u_i = 0 \Rightarrow v_i = 0$ for any v such that $1(v) \subseteq 1(u)$. Thus, let be $v = (v_1, v_2, \ldots, v_n)$ and, fixing all $i \in 0(u)$ as $v_i = 0$ and setting all remaining $v_i, i \in 1(u)$ free as

$$\{f(v) \mid 1(v) \subseteq 1(u)\} = \{f(\alpha_1, \alpha_2, \ldots, \alpha_n) \mid \alpha_i = 0, i \in 0(u)\},$$

and taking the maximum and minimum in this set leads to the result in the theorem. Note that the above set is finite since $\alpha_i \in \{0,1\}, i \in 1(u)$. □

Corollary 5 *From Theorem 4, we can have the upper and lower bounds of* $f(I_{\mathcal{G}_n}(x'))$ *at any* $x' \in \mathcal{T}(x)$ *for any function* $f : \{0,1\}^n \to \mathbb{R}$ *as*

$$\overline{f}(I_{\mathcal{G}_n}(x)) \geqslant f(I_{\mathcal{G}_n}(x')) \geqslant \underline{f}(I_{\mathcal{G}_n}(x)),$$

because $1(I_{\mathcal{G}_n}(x')) \subseteq 1(I_{\mathcal{G}_n}(x))$ *for* $x' \in \mathcal{T}(x)$ *from Lemma 2.*

3.2 Morishita-Kudo Bounds

Theorem 4 and Corollary 5 give us a general idea to obtain pruning bounds for arbitrary function f in the depth-first traversal of an enumeration tree. However, in general, it requires a combinatorial search to obtain these bounds. We present computationally tractable and useful bounds that we call *Morishita-Kudo bounds* for *separable* functions. Note that the target functions appeared in the previous studies [1], [2], [3], [4], [32] are all separable. On the other hand, non-separable cases include when the function has a term not controllable by Boolean variables such as the penalty terms of Problem (2) and also when the function has higher-order terms between Boolean variables, like mutual information.

Lemma 6 (Morishita-Kudo bounds) *Assume that a real-valued function of* n*-dimensional Boolean vector* $f : \{0,1\}^n \to \mathbb{R}$ *is separable, meaning that there exists a set of* n *functions* $f_i : \{0,1\} \to \mathbb{R}, i = 1, 2, \ldots, n$ *and*

$$f(u_1, u_2, \ldots, u_n) = \sum_{i=1}^{n} f_i(u_i), \quad u_i \in \{0,1\}.$$

Then, for given $u = (u_1, u_2, \ldots, u_n) \in \{0,1\}^n$, *we have*

$$\overline{f}(u) \geqslant f(v) \geqslant \underline{f}(u)$$

for any $v = (v_1, v_2, \ldots, v_n) \in \{0,1\}^n$ *such that* $1(v) \subseteq 1(u)$, *where*

$$\underline{f}(u) := \sum_{i \in 1(u)} \min\{f_i(0), f_i(1)\} + \sum_{i \in 0(u)} f_i(0)$$
$$\overline{f}(u) := \sum_{i \in 1(u)} \max\{f_i(0), f_i(1)\} + \sum_{i \in 0(u)} f_i(0)$$

Thus, for separable functions, if we have some fixed $u \in \{0,1\}^n$, then we can limit the possible range of $f(v)$ for any v such that $1(v) \subseteq 1(u)$, and the upper and lower bounds for $f(v)$, i.e. $\underline{f}(u)$ and $\overline{f}(u)$, are easy to compute just by comparing $f_i(0)$ and $f_i(1)$ for each $i \in 1(u)$. Since $1(v) \subseteq 1(u)$, we have $v_i = 0$ for $i \in 0(u)$ and the amount of $\sum_{i \in 0(u)} f_i(0)$ is unchanged and cannot be further improved. Only elements that can differ are v_is for $i \in 1(u)$, and therefore we have the maximum or minimum of $f(v)$ for v such that $1(v) \subseteq 1(u)$.

In supplementary materials, we show three examples, in which previously obtained pruning bounds for specific targets can be easily derived using Morishita-Kudo bounds than the original proofs.

4 Learning Sparse Linear Models by Block Coordinate Gradient Descent

We now describe a way to optimize Problem (2), searching necessary subgraph features simultaneously. Our main idea is first to make *block coordinate gradient descent* [35], [36] feasible over all subgraph indicators. Then, by setting a small block size to update, Problem (2) with an intractably large number of variables becomes practically solvable.

Although coordinate-descent-type optimization for solving (2) is known to be quite effective in non-graph cases [37], but totally-corrective boosting based on column generation can be another option [3], [38]. In this framework, adding the top-k optimal variables at each iteration, called *multiple-pricing*, can be also investigated. Note that this approach needs to repeatedly solve internal optimization problems, for example, linear programming for [3], and nonlinear programming for more general loss functions. Multiple pricing leads to more constraints in the dual problem and increases this internal optimization time, and the effect is concluded as not significant when the search space is progressively expanded [3].

4.1 Tseng-Yun Class of Block Coordinate Gradient Descent

We consider block coordinate gradient descent with *small nonzero coordinate blocks* to our simultaneous learning of subgraph features and parameters to be optimized [35], [36]. This algorithm is known to be efficient [37] compared to the other methods, and also the global and linear convergence under a local error bound condition is guaranteed.

Let $\theta(t)$ be the parameter value of interest at iteration t. The block coordinate gradient descent is based on gradient descent by applying local second-order approximation at the current $\theta(t)$ to only the smooth part $f(\theta)$ of the objective function $F(\theta) = f(\theta) + R(\theta)$ as

$$\begin{aligned}&\min_\theta [f(\theta) + R(\theta)] = \min_\theta [f(\theta) - f(\theta(t)) + R(\theta)]\\ &\approx \min_\theta \Big[\langle \nabla f(\theta(t)), \theta - \theta(t)\rangle \\ &\quad + \frac{1}{2}\langle \theta - \theta(t), H(t)(\theta - \theta(t))\rangle + R(\theta)\Big]\end{aligned}$$

where $H(t) \succ 0$ is a positive-definite matrix approximating the Hessian $\nabla^2 f(\theta(t))$. The main idea is to solve this local minimization by block coordinate descent instead of directly optimizing the original objective function by coordinate descent, which may be viewed as a hybrid of gradient projection and coordinate descent. The coordinate block to be updated at each iteration is chosen in a Gauss-Southwell way, which can be the entire coordinates or a small block of coordinates satisfying a given condition.

More precisely, this algorithm iterates the following steps to update the parameter $\theta(t)$ until convergence:

Step 1. Compute the minimizer $T(\theta(t))$ by coordinate descent.
Step 2. Compute the descent direction by $d(t) = T(\theta(t)) - \theta(t)$.
Step 3. Set Gauss-Southwell-r block by
$d(t)_j = 0$ for $\{j \mid v(t)\|d(t)\|_\infty > |d(t)_j|\}$.
Step 4. Do a line search for $\alpha(t)$ with the modified Armijo rule.
Step 5. Update the parameter by $\theta(t+1) \leftarrow \theta(t) + \alpha(t)d(t)$.

Here the mapping $T(\theta(t))$ is defined as

$$\begin{aligned}T(\theta(t)) := \arg\min_\theta \Big[&\langle \nabla f(\theta(t)), \theta - \theta(t)\rangle \\ &+ \frac{1}{2}\langle \theta - \theta(t), H(t)(\theta - \theta(t))\rangle + R(\theta)\Big]\end{aligned} \tag{5}$$

and the modified Armijo rule for a line search is the following: choose $\alpha_{\text{init}}(t) > 0$ and let $\alpha(t)$ be the largest element of $\{\alpha_{\text{init}}(t)\, s^j\}, j = 0, 1, \dots$, where s is a scaling parameter such that $0 < s < 1$, satisfying

$$F(\theta(t) + \alpha(t)d(t)) \leqslant F(\theta(t)) + \alpha(t)\sigma\Delta(t)$$

where $0 < \sigma < 1, 0 < \gamma < 1$, and

$$\begin{aligned}\Delta(t) := &\langle \nabla f(\theta(t)), d(t)\rangle + \gamma\langle d(t), H(t)d(t)\rangle \\ &+ R(\theta(t) + d(t)) - R(\theta(t)).\end{aligned}$$

4.2 Tracing Nonzero Coefficients of Subgraph Indicators

We show the way to run the Tseng-Yun block coordinate gradient descent during our subgraph search of Problem (2). In our case, the parameter of interest $\theta(t)$ at iteration t is

$$\theta(t) := (\beta_0(t), \beta_1(t), \beta_2(t), \dots)$$

The dimension of $\theta(t)$ is intractably huge and practically uncomputable, due to a combinatorially large number of all possible subgraphs in the given data. Each j-th coordinate of $\theta(t)$ is associated with the corresponding subgraph $x_j \in \mathcal{X}(\mathcal{G}_n)$.

The first problem is that we cannot have even $\theta(t)$ explicitly. Thus our strategy is to keep only the non-zero coordinate block of $\theta(t)$ by avoiding the evaluation of zero coordinates as much as possible. We start with setting all coordinates of initial $\theta(0)$ to zero, and hence it is enough to consider the update rule to obtain the nonzero part of $\theta(t+1)$ from the nonzero part of $\theta(t)$.

Let us assume that we already have all nonzero coordinates of $\theta(t)$. Taking a look at **Step 1** to **Step 5** to get $\theta(t+1)$, the nonzero part of $\theta(t+1)$ is detected in **Step 1**. Indeed, suppose that we have the nonzero part of $T(\theta(t))$, since $d(t) = T(\theta(t)) - \theta(t)$, we have $\{j \mid d(t)_j \neq 0\} \subseteq \{j \mid \theta(t)_j \neq 0\} \cup \{j \mid T(\theta(t))_j \neq 0\}$. Since $\theta(t+1) = \theta(t) + \alpha(t)d(t)$, we can also see $\{j \mid \theta(t+1)_j \neq 0\} \subseteq \{j \mid \theta(t)_j \neq 0\} \cup \{j \mid T(\theta(t))_j \neq 0\}$.

We thus focus on how to identify the nonzero indexes $\{j \mid T(\theta(t))_j \neq 0\}$ from the current $\theta(t)$ (**Step 1**). Note that **Step 1** is based on coordinate descent and each i-th coordinate $T(\theta(t))_j$ is separately computed in a coordinate-wise manner. Also note that **Step 3**, **4** and **5** can be carried out only after **Step 1** and **2** for all nonzero coordinates are obtained because **Step 3** requires $\|d(t)\|_\infty$.

To realize **Step 1**, we use the following lemma, which shows that $T(\theta(t))_j$ for the j-th coordinate has the closed-form solution, which comes from the complementary slackness of primal-dual pairs in convex optimization.

Lemma 7 *In Problem* (2)*, when we solve* ***Step 1*** *by coordinate descent, the following closed-form solution exists for each* $j = 1, 2, \dots$*:*

$$T(\theta(t))_j = \begin{cases} -H(t)_{jj}^{-1}(b_j(t) + \lambda_1) & b_j(t) < -\lambda_1 \\ -H(t)_{jj}^{-1}(b_j(t) - \lambda_1) & b_j(t) > \lambda_1 \\ 0 & |b_j(t)| \leqslant \lambda_1 \end{cases}$$

where

$$b_j := \sum_{i=1}^n \frac{\partial L(y_i, \mu(g_i; \theta(t)))}{\partial \theta(t)_j} + (\lambda_2 - H(t)_{jj})\theta(t)_j.$$

Proof. See the supplementary material. □

Combined with the structured search space of $\mathcal{X}(\mathcal{G}_n)$ which is equal to the enumeration tree $\mathcal{T}(\mathcal{G}_n)$, Lemma 7 provides a way to examine if $T(\theta(t))_k = 0$ for unseen k satisfying $x_k \in \mathcal{T}(x_j)$: if we already know $T(\theta(t))_k = 0$ for all unseen $x_k \in \mathcal{T}(x_j)$, we can skip checking all subgraphs in the subtree below x_j, i.e. $x_k \in \mathcal{T}(x_j)$

As the lemma claims, $T(\theta(t))_k = 0$ is controlled by whether $|b_k(t)| \leqslant \lambda_1$ or not. If we know the largest possible value b_j^* of $|b_k(t)|$ for any k such that $x_k \in \mathcal{T}(x_j)$ and also know that $b_j^* \leqslant \lambda_1$, then we can conclude $T(\theta(t))_k = 0$ for all of such ks. This bound b_j^* can be obtained as follows: let $b_k^{(1)}(t)$ and $b_k^{(2)}(t)$ be the first and second terms of $b_k(t)$, respectively, as

$$\begin{aligned}b_k^{(1)}(t) &:= \sum_{i=1}^n \frac{\partial L(y_i, \mu(g_i; \theta(t)))}{\partial \theta(t)_k}, \\ b_k^{(2)}(t) &:= (\lambda_2 - H(t)_{kk})\theta(t)_k.\end{aligned}$$

Since for any $\underline{b}^{(1)} \leqslant b^{(1)} \leqslant \overline{b}^{(1)}$, and $\underline{b}^{(2)} \leqslant b^{(2)} \leqslant \overline{b}^{(2)}$,

$$|b^{(1)} + b^{(2)}| \leqslant \max\{\overline{b}^{(1)} + \overline{b}^{(2)}, -\underline{b}^{(1)} - \underline{b}^{(2)}\}. \tag{6}$$

We can obtain the bounds for $|b_k(t)|$ if we have the individual bounds for $b_k^{(1)}$ and $b_k^{(2)}$. Since $b_k^{(1)}(t)$ is separable (See the supplementary material for the proof of Theorem 8), we can have Morishita-Kudo bounds by Lemma 6. On the other hand, Morishita-Kudo bounds cannot be applied to the second term $b_k^{(2)}(t)$. However, since we already have all $\theta(t)_j$ for the nonzero indexes $\{j \mid \theta(t)_j \neq 0\}$, we can have $H(t)_{kk}$ and $\theta(t)_k$ for $\theta(t)_k \neq 0$. Then even if we might not have the value of $H(t)_{kk}$ for $\theta(t)_k = 0$, we can have $b_k^{(2)} = 0$ regardless of the value of $H(t)_{kk}$. Then, provided that we have the index-to-set mapping

$$j \mapsto \{k \mid x_k \in \mathcal{T}(x_j), \theta(t)_k \neq 0\} \tag{7}$$

at each j, we can also have the exact upper and lower bounds for $|b_k^{(2)}(t)|$ such that $x_k \in \mathcal{T}(x_j)$. By Lemma 7 and *depth-first dictionary passing* (See the supplementary material), we have

Theorem 8. Note that this result also confirms that we can control the sparsity of $\theta(t)$ by parameter λ_1.

Theorem 8 *Suppose we have $x_j \in \mathcal{T}(\mathcal{G}_n)$. Then, for any $x_k \in \mathcal{T}(x_j)$, there exist upper and lower bounds*

$$\underline{L}_j(t) \leqslant \sum_{i=1}^{n} \frac{\partial L(y_i, \mu(g_i; \theta(t)))}{\partial \theta(t)_k} \leqslant \overline{L}_j(t)$$
$$\underline{B}_j(t) \leqslant (\lambda_2 - H(t)_{kk})\theta(t)_k \leqslant \overline{B}_j(t),$$

that are only dependent on x_j, and $T(\theta(t))_k = 0$ if $\max\{\overline{L}_j(t) + \overline{B}_j(t), -\underline{L}_j(t) - \underline{B}_j(t)\} \leqslant \lambda_1$.

Proof. See the supplementary material. □

Remark 9 *If we observe $\max\{\overline{L}_j(t)+\overline{B}_j(t), -\underline{L}_j(t)-\underline{B}_j(t)\} \leqslant \lambda_1$ at x_j, we can conclude that there are no $x_k \in \mathcal{T}(x_j)$ such that $T(\theta)_k \neq 0$, and therefore prune this entire subtree $\mathcal{T}(x_j)$ in search for $T(\theta)_k \neq 0$.*

4.3 Algorithm

Figure 2 shows the pseudocode for the entire procedure. Note that h and h' are data structures to access the set of mappings shown in (7), and also h and h' are processed through the *depth-first dictionary passing*. See the supplementary material for details.

5 Numerical Case Study

As a case study of our proposed framework, we take an example of L1-penalized logistic regression. Logistic regression is a fundamental model for classification, and gives a baseline understanding about the linear separability of the data. The proposed algorithm is, to our knowledge, the first exact method to directly learn logistic regression over all subgraph features under elastic-net regularization (a possibility of generalizing a boosting-based approach was already discussed [38]).

We numerically examine the properties and performance of L1-penalized logistic regression (L1-LogReg) by using our algorithm shown in Figure 2, for Problem 2, $\lambda_2 = 0$ and

$$L(y, \mu) = y \log(1 + \exp(-\mu)) + (1 - y) \log(1 + \exp(\mu))$$

for $y \in \{0, 1\}$. Our results are compared to those of two existing methods mentioned in Sections 1 and 2: Adaboost [1] and LPBoost [3]. According to the example of [36], we set

$$H(t)_{jj} := \min\left\{\max\{\nabla^2 f(\theta(t))_{jj}, 10^{-10}\}, 10^{10}\right\}$$

and

$$\sigma = 0.1, c = 0.5, \gamma = 0, \alpha_{\text{init}}(0) = 1,$$
$$\alpha_{\text{init}}(t) = \min\left\{\frac{\alpha(t-1)}{c^5}, 1\right\}, v(t) = 0.9.$$

5.1 Datasets

For systematic evaluation, we use two datasets for binary classification: a controlled random graph dataset (RAND) and a real dataset (CPDB). The full details and the results for other datasets are described in the supplementary materials.

RAND dataset consists of 100,000 graphs, each of which is generated by probabilistically combining small graphs from a random-graph pool. Thus we can know "ground truth" for discriminative subgraph features embedded in given (observed) graphs, and we can compare the selected subgraph features by each learning algorithm to the ground truth.

CPDB dataset is the mutagenicity data from carcinogenic potency database (CPDB) [39], consisting of 684 graphs (mutagens: 341, nonmutagens: 343) for a binary classification task.

Algorithm:

$\theta(0) \leftarrow 0$;
Build empty h, h';
for $t = 0, 1, 2, \ldots$ **do**
  Build an empty hes, h_{tmp};
  foreach $i \in \{i \mid \theta(t)_i \neq 0\}$ **do** $hes[i] \leftarrow H(t)_{ii}$;
  foreach $x_j \in \mathcal{T}(\mathcal{G}_n)$ *in the depth-first traversal* **do**
    begin *pre-order operation*
      Compute $T(\theta(t))_j$;
      if $T(\theta(t))_j \neq 0$ **then**
        foreach $i \in$KEYS(h_{tmp}) **do**
        $h_{\text{tmp}}[i] \leftarrow h_{\text{tmp}}[i] \cup \{j\}$;
      end
      $h_{\text{tmp}}[j] \leftarrow \{\}$;
      $h[j] \leftarrow (h[j] \cup h'[j]) \cap \{i \mid \theta(t)_i \neq 0\}$ **if** $h[j] = \{\}$ **then**
        $\underline{B}_j \leftarrow 0, \overline{B}_j \leftarrow 0$;
      else
        $\overline{B}_j \leftarrow \max\left\{\max_{k \in h[j]}\{(\lambda_2 - hes[k])\theta(t)_k\}, 0\right\}$;
        $\underline{B}_j \leftarrow \min\left\{\min_{k \in h[j]}\{(\lambda_2 - hes[k])\theta(t)_k\}, 0\right\}$;
      end
      Compute $\overline{L}_j, \underline{L}_j$;
      if $\max\{\overline{L}_j + \overline{B}_j, -\underline{L}_j - \underline{B}_j\} \leqslant \lambda_1$ **then**
        Prune $\mathcal{T}(x_j)$;
      else
        Visit children of x_j;
      end
    end
    begin *post-order operation*
      if $h_{\text{tmp}}[j] \neq \{\}$ **then**
        $h'[j] \leftarrow h_{\text{tmp}}[j]$;
      end
      DELETEKEY(h_{tmp}, j);
    end
  end
  $d(t)_i \leftarrow T(\theta(t))_i - \theta(t)_i$ for
  $i \in \{i \mid T(\theta(t))_i \neq 0 \vee \theta(t)_i \neq 0\}$;
  Gauss-Southwell-r: $d(t)_i \leftarrow 0$ *for i such that* $|d(t)|_i \leqslant v(t) \cdot \|d(t)\|_\infty$;
  Armijo: $\theta(t+1) \leftarrow \theta(t) + \alpha d(t)$;
  Convergence test: **if** $\|H(t)d(t)\|_\infty \leqslant \epsilon$ **then** quit;
end

Fig. 2. The proposed algorithm for solving Problem (2).

5.2 Evaluating learning curves on RAND

We first investigated the convergence property by the learning curves of the three methods on RAND. First we divided the dataset into 100 sets, each containing 1,000 graphs (500 positives and 500 negatives). We estimated the expected *training error* by computing each training error of model i with data set i that was used to train the model, and averaging over those 100 values obtained from 100 sets. Since all 100 sets share the same probabilistic rule behind their generation, we also estimated the expected *test error* by first randomly choosing 100 pairs of set i and model j ($i \neq j$), and computing the test error of the model i with data set j that was not used to train the model, and averaging over those 100 values obtained from 100 pairs. We used the same fixed 100 pairs for evaluating all three methods of Adaboost, LPBoost, and L1-LogReg.

Figures 3 shows the learning curves of three methods. We can see that the convergence rate of L1-LogReg was much better and stable than Adaboost and LPBoost. For example, even after only around 20 iterations, the error was almost the same as the last converged value, while Adaboost and LPBoost needed around 100 and 50, respectively, iterations for that condition. Also we can see that LPBoost accelerated the convergence rate of Adaboost. The convergence behavior of LPBoost was

unstable at the beginning of iterations, which was already pointed out in the literature [40], [41], whereas Adaboost and L1-LogReg were more stable. LPBoost however often achieved slightly lower error than L1-LogReg and Adaboost, implying that the hinge loss function (LPBoost) fits better to the task compared to the logistic loss (L1-LogReg) or the exponential loss (Adaboost).

5.3 Evaluating selected subgraph features on RAND

We compared the subgraph features selected by the three methods after convergence, i.e. the features having non-zero coefficients, keeping the number of features at the same. To do so, we carefully chose the parameter values of the three methods, i.e. 325 for Adaboost, 0.335 for LPBoost and 0.008 for L1-LogReg under RAND, resulting in around 240 features by each method. Table 1 shows the statistics of this result. Note that the test errors of the three models were comparable, i.e. around 0.17. Note that the original dataset of RAND was generated by combining 100 small graphs in a seed-graph pool, but the number of learned features in Table 1 was around 240, which is more than 100.

In Figure 4, we show the size distribution of subgraph features in the seed-graph pool in the left-most panel that was used to generate the data, and those of Adaboost, LPBoost, and L1-LogReg in the right three panels. Interestingly, even though the number of features ($\approx$ 240) and the performance of the three methods ($\approx$ 0.17) were almost similar, the selected sets of subgraph features were quite different in their sizes. We can see that compared to the original subgraphs stored in the seed-graph pool (up to size 7), all three methods chose much smaller subgraphs and tried to represent the data by combining those small subgraphs. In particular, Adaboost and LPBoost focused on selecting the subgraph features, where their size was less than and equal to three, mostly two (graphs with only two edges). By contrast, L1-LogReg had comparatively balanced size distribution.

Figure 5 shows the number of overlapped subgraph features between different methods (averaged over 100 sets). Figure 4 might give a misleading impression that the selected features of Adaboost and LPBoost would be similar since the obtained number of features is similar. Yet they are remarkably different as we see in Figure 5. Figure 5 shows that by looking more closely, LPBoost and Adaboost shared a much larger number of features than those between either of them and L1-LogReg.

5.4 Evaluating predictive performance on CPDB

Table 2 shows the comparison result of the classification accuracy (ACC) for CPDB and the number of selected subgraph features and iterations. We included a standard method in chemoinformatics as a baseline, shown by glmnet in Table 2: We first computed the fingerprint bit vector and then applied standard 1-norm penalized logistic regression optimized by glmnet [42]. For fingerprints, we used four different fingerprints, FP1, FP2, FP3, and MACCS, generated by Open Babel[1].

L1-LogReg achieved accuracy comparable with the best one by Adaboost and LPBoost with a much smaller number of subgraph features. Table 3 shows the detailed statistics behind Table 2, where "time (sec)" denotes the CPU time[2] in seconds.

1. Open Babel v2.3.0 documentation: Molecular fingerprints and similarity searching. http://openbabel.org/docs/dev/Features/Fingerprints.html

2. The CPU time is measured by a workstation with 2 $\times$ 2.93 GHz 6-Core Intel Xeon CPUs and 64 GB Memory.

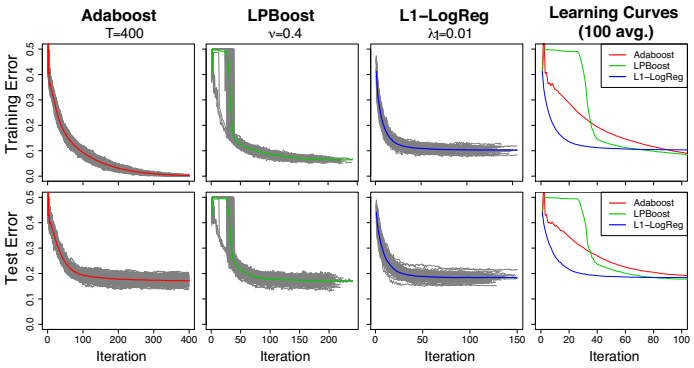

Fig. 3. Learning curves for RAND (average over 100 trials).

TABLE 1
Statistics of carefully chosen parameters for the same number of selected features.

method	param	#feat.	#ite.	error (train)	error (test)
Adaboost	325	239.94 $\pm$ 8.50	325	0.0068	0.1736
LPBoost	0.335	239.69 $\pm$ 21.80	275.91	0.0405	0.1704
L1-LogReg	0.008	239.36 $\pm$ 29.16	122.52	0.0931	0.1758
		$\approx$ 240			$\approx$ 0.17

We also included an elastic-net penalized logistic regression (Els-LogReg) with fixed λ_1 and changing parameter λ_2. Also in this table we show the results of inexact and fast algorithms of Figure 2 with pruning the case in which the same characteristic vector $I_{\mathcal{G}_n}(x)$ is already evaluated and the size of x is larger than 3. However, in this task, the test accuracy of Els-LogReg with $\lambda_2 > 0$ is worse than that with $\lambda_2 = 0$ (L1-LogReg), and decreasing as λ_2 increases. We see that the inexact version achieved fast results compared to the exact version, with keeping or even increasing the accuracy performance.

6 Discussion

The redundancy and high correlation of subgraph features primarily come from the subgraph isomorphism. When subgraph features x_i and x_j are very similar, the corresponding subgraph indicators $I(x_i \subseteq g)$ and $I(x_j \subseteq g)$ would take very similar values. Moreover, the number of samples is generally far less than the number of possible subgraph features. Therefore, we can have many exactly identical column vectors that correspond to different subgraph features, which causes *perfect multicollinearity*.

In most practical cases, we have a particular set of subgraph features, say X, such that any $x \in X$ has the same characteristic vector $I_{\mathcal{G}_n}(x)$. These subgraph features form an equivalence class

$$[x] := \{x' \in \mathcal{X}(\mathcal{G}_n) \mid I_{\mathcal{G}_n}(x') = I_{\mathcal{G}_n}(x)\},$$

where any representative subgraph feature x has the same $I_{\mathcal{G}_n}(x)$. Note that two graphs with very different structures can be in the same equivalence class if their subgraph features perfectly co-occur (For example, disconnected-subgraph patterns).

Therefore, for given $x_i, x_j \in [x]$, we cannot distinguish if either of the two subgraph indicators is better than the other, just by using $\mathcal{G}_n$. A heuristics in terms of predictive performance is that the smallest subgraph in $[x]$ might be good as the representative subgraph, because smaller subgraphs are expected to occur in unseen graphs with a higher probability than that of larger ones. This point should be carefully treated

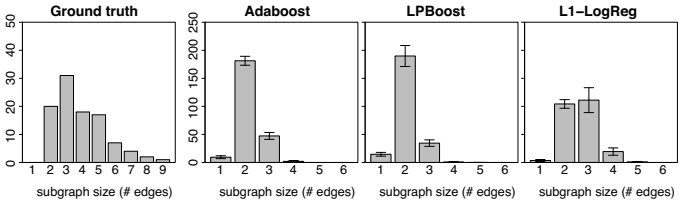

Fig. 4. Distribution of the size of selected subgraph features (average over 100 trials).

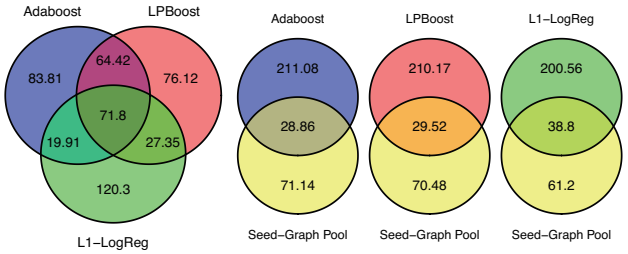

Fig. 5. The number of isomorphic subgraph features (average over 100 trials).

when we are interested in interpreting the selected set of subgraph features. We can always have many other candidates in the equivalence class that has exactly the same effect as a subgraph feature.

The existing methods such as Adaboost and LPBoost add a single best subgraph feature at each iteration with branch and bound, and thus usually ignore this problem, because they just take the first found best subgraph in $[x]$. On the other hand, the proposed method allows to add multiple subgraph features at each iteration. Hence we need to handle this point explicitly. In this paper, for the sake of comparison, we just take the first found subgraph x (following the other methods) and ignore the subsequently found subgraph in the same equivalence class $[x]$ by hashing based on $I_{\mathcal{G}_n}(x)$. However if we consider this problem carefully, the predictive performance of our algorithm might be improved. If needed, we can provide the entire set of equivalent class for each output.

7 Conclusions

In terms of sparse learning over the complete subgraph feature set, our contributions were the three points as mentioned in Introduction. We once again summarize them with the consequences and implications, as follows:

- We have showed that the many existing methods can be viewed as variations of a branch-and-bound algorithm based on Morishita-Kudo bounds. We have formulated this point as a property of pseudo Boolean functions. To our knowledge, this is the first attempt to explicitly formulate the general idea that underlies all relevant work. The presented formulation in terms of Boolean variables is independent of any pattern discovery problems, and also widely applicable to other contexts.
- We have presented a direct optimization algorithm for solving Problem (2) over graphs (Figure 2). This formulation includes a wide variety of important statistical models including generalized linear models, etc. We emphasize that unlike many other relevant work, Problem (2) cannot be directly solved by Morishita-Kudo bounds.
- We experimentally analyzed the convergence property, the predictive performance, and the obtained subgraph features using a case study with L1-penalized logistic regression (L1-LogReg), which was optimized by our proposed algorithm. Note that L1-LogReg has not been examined by any existing work yet. Our results showed the pros and cons of our approach in detail. Furthermore, the problem of multicollinearity and the detailed difference in selected features could be found but have not been pointed out in the literature yet. Our results would warn that we need carefully "interpret" the selected features obtained by these methods.

TABLE 2
Classification accuracy for the CPDB dataset (10-fold CV).

method		param	ACC (train)	ACC (test)	#feat.	#ite.
L1-LogReg (exact)		0.005	0.813	0.774	80.3	62.3
L1-LogReg (inexact)		0.002	0.862	0.783	92.4	72.7
Adaboost		500	0.945	0.772	180.3	500.0
LPBoost		0.4	0.895	0.784	101.1	126.4
glmnet (L1-LogReg)	FP2	0.02	0.828	0.739	1024	
	FP3	0.03	0.676	0.628	64	
	FP4	0.02	0.785	0.721	512	
	MACCS	0.01	0.839	0.771	256	

TABLE 3
Performance and search-space size for the CPDB dataset (10-fold CV).

method	param	ACC (train)	ACC (test)	#feat.	#ite.	time (sec)
L1-LogReg	0.004	0.825	0.755	95.0	57.5	8918.94
(exact,λ_1)	0.005	0.813	**0.774**	80.3	62.3	2078.88
	0.006	0.803	0.762	67.6	66.6	1012.93
	0.008	0.779	0.750	50.9	50.6	277.69
	0.010	0.756	0.733	39.1	46.4	97.29
L1-LogReg	0.001	0.905	0.781	138.0	85.0	2328.28
(inexact,λ_1)	0.002	0.862	**0.783**	92.4	72.7	695.39
	0.004	0.822	0.775	63.9	69.4	202.00
	0.006	0.802	0.762	50.7	61.0	68.91
	0.008	0.774	0.748	40.0	55.6	31.58
	0.010	0.757	0.733	31.8	55.1	21.21
Adaboost	600	0.950	0.769	190.0	600.0	40.67
	500	0.945	**0.772**	180.3	500.0	36.21
	400	0.938	0.769	168.8	400.0	29.38
LPBoost	0.3	0.939	0.748	141.4	185.9	16.02
	0.4	0.895	**0.784**	101.1	126.4	7.76
	0.5	0.858	0.767	67.1	80.2	4.39
Els-LogReg	0.001	0.807	0.759	154.7	82.7	380.12
(inexact,λ_2)	0.002	0.796	0.755	200.0	84.8	516.70
$\lambda_1 = 0.004$	0.004	0.785	0.745	257.1	86.8	720.97
Els-LogReg	0.001	0.793	0.756	120.2	70.7	178.04
(inexact,λ_2)	0.002	0.782	0.752	152.3	73.5	244.02
$\lambda_1 = 0.005$	0.004	0.775	0.749	200.8	81.6	377.32

Possible future work would be to build a faster algorithm under practical situations, which would be always mandatory for supervised learning over complex data, particularly graphs. We believe that our results and findings contribute to the advance of understanding in the field of general supervised learning from graphs considering all possible subgraph features, and also restimulate the current study of this research direction.

Acknowledgments

This work was supported in part by JSPS/MEXT KAKENHI Grant Number 26120503, 26330242, 24300054, 16H02868; the

Collaborative Research Program of Institute for Chemical Research, Kyoto University (grant #2014-27 and #2015-33); JST PRESTO; and FiDiPro, Tekes.

REFERENCES

[1] T. Kudo, E. Maeda, and Y. Matsumoto, "An application of boosting to graph classification," in *Advances in Neural Information Processing Systems 17*, L. K. Saul, Y. Weiss, and L. Bottou, Eds. Cambridge, MA: MIT Press, 2005, pp. 729–736.

[2] K. Tsuda, "Entire regularization paths for graph data," in *Proceedings of the 24th International Conference on Machine learning (ICML)*, Banff, Alberta, Canada, 2007, pp. 919–926.

[3] H. Saigo, S. Nowozin, T. Kadowaki, T. Kudo, and K. Tsuda, "gBoost: a mathematical programming approach to graph classification and regression," *Machine Learning*, vol. 75, pp. 69–89, 2009.

[4] H. Saigo, N. Krämer, and K. Tsuda, "Partial least squares regression for graph mining," in *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, Las Vegas, Nevada, USA, 2008, pp. 578–586.

[5] Z. Harchaoui and F. Bach, "Image classification with segmentation graph kernels," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, Minneapolis, Minnesota, USA, 2007, pp. 1–8.

[6] S. Nowozin, K. Tsuda, T. Uno, T. Kudo, and G. Bakır, "Weighted substructure mining for image analysis," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, Minneapolis, Minnesota, USA, 2007, pp. 1–8.

[7] V. Barra and S. Biasotti, "3D shape retrieval using kernels on extended reeb graphs," *Pattern Recognition*, vol. 46, no. 11, pp. 2985–2999, 2013.

[8] L. Bai, L. Rossi, H. Bunke, and E. R. Hancock, "Attributed graph kernels using the jensen-tsallis q-differences," in *Proceedings of the 2014 European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD 2014)*, Nancy, France, 2014, pp. 99–114.

[9] I. Takigawa and H. Mamitsuka, "Graph mining: procedure, application to drug discovery and recent advances," *Drug Discovery Today*, vol. 18, no. 1-2, pp. 50–57, 2013.

[10] I. Takigawa, K. Tsuda, and H. Mamitsuka, "Mining significant substructure pairs for interpreting polypharmacology in drug-target network," *PLoS ONE*, vol. 6, no. 2, p. e16999, 2011.

[11] J.-P. Vert, "Classification of biological sequences with kernel methods," in *Proceedings of The 8th International Colloquium on Grammatical Inference (ICGI)*, Tokyo, Japan, 2006, pp. 7–18.

[12] Y. Yamanishi, F. Bach, and J.-P. Vert, "Glycan classification with tree kernels," *Bioinformatics*, vol. 23, no. 10, pp. 1211–1216, 2007.

[13] K. Hashimoto, I. Takigawa, M. Shiga, M. Kanehisa, and H. Mamitsuka, "Mining significant tree patterns in carbohydrate sugar chains," *Bioinformatics*, vol. 24, no. 16, pp. i167–i173, 2008.

[14] Y. Karklin, R. F. Meraz, and S. R. Holbrook, "Classification of non-coding RNA using graph representations of secondary structure," in *Proceedings of the Pacific Symposium on Biocomputing (PSB)*, Hawaii, USA, 2005, pp. 4–15.

[15] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. V. N. Vishwanathan, A. J. Smola, and H.-P. Kriegel, "Protein function prediction via graph kernels," *Bioinformatics*, vol. 21, no. 1, pp. i47–i56, 2005.

[16] J.-P. Vert, J. Qiu, and W. S. Noble, "A new pairwise kernel for biological network inference with support vector machines," *BMC Bioinformatics*, vol. 8, no. Suppl 10, p. S8, 2007.

[17] H. Kashima, K. Tsuda, and A. Inokuchi, "Marginalized kernels between labeled graphs," in *Proceedings of the 20th International Conference on Machine Learning (ICML)*, Washington, DC, USA, 2003, pp. 321–328.

[18] T. Gärtner, P. A. Flach, and S. Wrobel, "On graph kernels: Hardness results and efficient alternatives," in *Proceedings of the 16th Annual Conference on Computational Learning Theory (COLT) and 7th Kernel Workshop*, 2003, pp. 129–143.

[19] P. Mahé and J.-P. Vert, "Graph kernels based on tree patterns for molecules," *Machine Learning*, vol. 75, no. 1, pp. 3–35, 2009.

[20] R. Kondor and K. M. Borgwardt, "The skew spectrum of graphs," in *Proceedings of the 25th International Conference on Machine Learning (ICML)*, Helsinki, Finland, 2008, pp. 496–503.

[21] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph kernels," *Journal of Machine Learning Research*, vol. 11, pp. 1201–1242, 2010.

[22] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research*, vol. 12, no. Sep, pp. 2539–2561, 2011.

[23] D. Rogers and M. Hahn, "Extended-connectivity fingerprints," *Journal of Chemical Information and Modeling*, vol. 50, no. 5, pp. 742–754, 2010.

[24] M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis, "Frequent substructure-based approaches for classifying chemical compounds," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 8, pp. 1036–1050, 2005.

[25] N. Wale, I. A. Watson, and G. Karypis, "Comparison of descriptor spaces for chemical compound retrieval and classification," *Knowledge and Information Systems*, vol. 14, no. 3, pp. 347–375, 2008.

[26] K. Tsuda and T. Kudo, "Clustering graphs by weighted substructure mining," in *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, Pittsburgh, Pennsylvania, USA, 2006, pp. 953–960.

[27] K. Tsuda and K. Kurihara, "Graph mining with variational dirichlet process mixture models," in *Proceedings of the SIAM International Conference on Data Mining (SDM)*, Atlanta, Georgia, USA, 2008, pp. 432–442.

[28] H. Saigo and K. Tsuda, "Iterative subgraph mining for principal component analysis," in *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining (ICDM)*, Pisa, Italy, 2008, pp. 1007–1012.

[29] X. Yan and J. Han, "gSpan: Graph-based substructure pattern mining," in *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM)*, Washington, DC, USA, 2002, pp. 721–724.

[30] S. Nijssen and J. N. Kok, "A quickstart in frequent structure mining can make a difference," in *Proceeding of the 10th ACM SIGKDD international conference on Knowledge discovery and data mining*, Seattle, Washington, USA, 2004, pp. 647–652.

[31] D. Avis and K. Fukuda, "Reserve search for enumeration," *Discrete Applied Mathematics*, vol. 65, no. 1–3, pp. 21–46, 1996.

[32] S. Morishita, "Computing optimal hypotheses efficiently for boosting," in *Progress in Discovery Science, Final Report of the Japanese Discovery Science Project*, ser. Lecture Notes in Computer Science, S. Arikawa and A. Shinohara, Eds. Springer, 2002, vol. 2281, pp. 471–481.

[33] X. Kong and P. S. Yu, "gMLC: a multi-label feature selection framework for graph classification," *Knowledge and Information Systems*, vol. 31, no. 2, pp. 281–305, 2012.

[34] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*, Santiago, Chile, 1994, pp. 487–499.

[35] P. Tseng and S. Yun, "A coordinate gradient descent method for nonsmooth separable minimization," *Mathematical Programming*, vol. 117, pp. 387–423, 2009.

[36] S. Yun and K.-C. Toh, "A coordinate gradient descent method for ℓ_1-regularized convex minimization," *Computational Optimization and Applications*, vol. 48, pp. 273–307, 2011.

[37] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski, "Optimization with sparsity-inducing penalties," *Foundations and Trends in Machine Learning*, vol. 4, no. 1, pp. 1–106, 2011.

[38] S. Nowozin, "Learning with structured data: Applications to computer vision," Ph.D. dissertation, Technical University of Berlin, 2009.

[39] C. Helma, T. Cramer, S. Kramer, and L. D. Raedt, "Data mining and machine learning techniques for the identification of mutagenicity inducing substructures and structure activity relationships of noncongeneric compounds," *Journal of Chemical Information and Modeling*, vol. 44, no. 4, pp. 1402–1411, 2004.

[40] M. Warmuth, K. Glocer, and G. Rätsch, "Boosting algorithms for maximizing the soft margin," in *Advances in Neural Information Processing Systems 20*, J. Platt, D. Koller, Y. Singer, and S. Roweis, Eds. Cambridge, Massachusetts, USA: MIT Press, 2008, pp. 1585–1592.

[41] M. K. Warmuth, K. A. Glocer, and S. V. N. Vishwanathan, "Entropy regularized LPBoost," in *Proceedings of the 19th International Conference on Algorithmic Learning Theory (ALT)*, Budapest, Hungary, 2008, pp. 256–271.

[42] J. H. Friedman, T. Hastie, and R. Tibshirani, "Regularization paths for generalized linear models via coordinate descent," *Journal of Statistical Software*, vol. 33, no. 1, pp. 1–22, 2010.