Model Selection-inspired Coefficients Optimization for Polynomial-Kernel Graph Learning

Cheng Yang*, Fen Wang‡, Minxiang Ye‡, Guangtao Zhai*,
Xiao-Ping Zhang§, Vladimir Stankovic¶, and Lina Stankovic¶
* Shanghai Jiao Tong University, Shanghai, China
E-mails: {bobcy,zhaiguangtao}@sjtu.edu.cn
‡ Zhejiang Lab, Hangzhou, China
E-mail: yemx@zhejianglab.com
§ Ryerson University, Toronto, Canada
E-mail: xzhang@ryerson.ca
¶ University of Strathclyde, Glasgow, UK
E-mail: {vladimir.stankovic,lina.stankovic}@strath.ac.uk

***Abstract*—Graph learning has been extensively investigated for over a decade, in which the graph structure can be learnt from multiple graph signals (*e.g.*, graphical Lasso) or node features (*e.g.*, graph metric learning). Given partial graph signals, existing node feature-based graph learning approaches learn a pair-wise distance metric with gradient descent, where the number of optimization variables dramatically scale with the node feature size. To address this issue, in this paper, we propose a low-complexity model selection-inspired graph learning (MSGL) method with very few optimization variables independent with feature size, via leveraging on recent advances in graph spectral signal processing (GSP). We achieve this by 1) interpreting a finite-degree polynomial function of the graph Laplacian as a positive-definite precision matrix, 2) formulating a convex optimization problem with variables being the polynomial coefficients, 3) replacing the positive-definite cone constraint for the precision matrix with a set of linear constraints, and 4) solving efficiently the objective using the Frank-Wolfe algorithm. Using binary classification as an application example, our simulation results show that our proposed MSGL method achieves competitive performance with significant speed gains against existing node feature-based graph learning methods.**

***Index Terms*—Graph signal processing, graph learning, convex optimization**

I. Introduction

Recent years have witnessed the rise of cross-over research between statistical machine learning [20] and graph spectral signal processing (GSP) [17] with applications to data classification [36], social network management [17], regression-based image quality assessment [33], image enhancement [5], and clustering [24]. GSP focuses on processing the signals defined on a graph with vertices that support the data and weighted edges that represent the observation-pair similarity. The crux of GSP with machine learning is to design a graph that captures well the relationships among data observations, and formulate a signal prior, *e.g.*, a *graph signal smoothness* (GSS) prior, to perform efficient filtering of graph signals, e.g., to facilitate data classification via signal extrapolation or label propagation [2]. The graph structure can be learned either from: 1) collected raw data (e.g., time-series) [7], or 2) features constructed from data samples. Since successful graph filtering depends on how well the underlying graph models the structure of the collected data, graph learning is fundamentally important. Graph learning approaches can be categorized into two tracks: 1) data-driven graph learning [11] that learns the graph based on the collected raw data and 2) feature-enabled graph learning that uses constructed features to learn the graph [30], [31].

In the data-driven graph learning track, the focus is on modelling relationships between *multiple* spatial and/or temporal observations (graph signals) assigned to graph nodes. Authors in [7], [11] learnt the graph structure (i.e., the graph Laplacian matrix $\mathbf{L}$) by modelling M-dimensional observations distributed on N nodes, $\mathbf{X} \in \mathbb{R}^{M \times N}$, as a Gaussian process with a GSS prior, *i.e.*, using the quadratic form $\text{Tr}(\mathbf{X}^\top \mathbf{L} \mathbf{X})$. Recently, [34] proposed a generalized smoothness prior of graph signals defined by a similar quadratic function on a learnable *graph spectral kernel*, which manipulates the spectrum of $\mathbf{L}$ using a polynomial function. On the other hand, in the feature-enabled graph learning track, graph structure is learnt from the constructed features using a pre-defined / optimized metric function that captures importance of each feature [30]. These approaches reduce the complexity by optimising the metric function using a GSS prior defined as $\mathbf{x}^\top \mathbf{L} \mathbf{x}$, where $\mathbf{x} \in \mathbb{R}^N$ is a graph signal representing sample labels, each assigned to one graph node. Similarly, [35] presented a semi-supervised learning framework using a high-order regularizer $\mathbf{x}^\top \mathbf{L}^P \mathbf{x}$ as the graph signal prior.

Note that the existing graph learning methods either update every single graph node by solving systems of linear equations (*e.g.*, graphical Lasso [11], sparse graph learning [8], [10], and low-dimensional space learning [13]) or learn a pair-wise distance metric with gradient descent [28], [29], where the number of optimization variables dramatically scale with the number of data samples or feature dimension. In this paper, motivated by the enriched signal prior used by data-driven graph learning methods in a model selection framework [34], we propose a model selection-inspired graph learning

This is a peer-reviewed, accepted author manuscript of the following paper: Yang, C., Wang, F., Ye, M., Zhai, G., Zhang, X-P., Stankovic, V., & Stankovic, L. (Accepted/In press). Model selection-inspired coefficients optimization for polynomial-kernel graph learning. Paper presented at 13th Asia Pacific Signal and Information Processing Association Annual Summit and Conference , Tokyo, Japan.

(MSGL) method with very few optimization variables. Our main contribution is that, we extend the deterministic quadratic GSS prior to a signal prior defined on a learnable kernel function $\mathcal{L} = \sum_{i=1}^{P} \beta_i \mathbf{L}^i$ with the kernel parameters $\boldsymbol{\beta}$ where $\mathcal{L}$ is a positive-definite, finite-degree polynomial function of Laplacian matrix $\mathbf{L}$ for the graph computed from data features. We optimise $\boldsymbol{\beta}$ by first interpreting $\mathcal{L}$ as a precision matrix (*i.e.*, an inverse covariance matrix [9]) of the single-observation data [30] and formulate a convex optimization problem. We then constrain the positive-definiteness of $\mathcal{L}$ by a set of linear constraints of $\boldsymbol{\beta}$, such that the convex objective function can be solved efficiently using a Frank-Wolfe method [15].

Unlike [30] that is restricted to optimizing sample-pair feature distances within a graph kernel $\mathbf{L}$ chosen a priori, our proposed MSGL optimizes the polynomial coefficients $\boldsymbol{\beta}$ of $\mathcal{L}$ given $\mathbf{L}$. Moreover, we differ from [34] in that, our proposed MSGL method makes use of a feature-based graph for *precision matrix tuning* instead of exploiting multi-dimensional raw samples distributed across the graph nodes for *data prediction*, and thus MSGL only focuses on the kernel defined on the features that are in the input space (training), rather than the ones in both the input and output space (training and testing samples) [34]. Such fundamental difference leads to a highly efficient optimization scheme with one-time eigen-decomposition in our MSGL method, while [34] requires repeated eigen-decomposition or Cholesky decomposition iteratively throughout the optimization process. Experimental results for a binary classification problem shows that our proposed MSGL method achieves competitive performance with significant speed gains against existing feature-enabled graph learning methods.

II. Preliminaries

We consider an undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathbf{A})$ with $|\mathcal{V}| = N$ nodes, edges $(i, j) \in \mathcal{E}$ and adjacency matrix $\mathbf{A}$ whose entries A_{ij} are positive edge weights defined by a Gaussian kernel:

$$A_{ij} = \begin{cases} \exp\left\{-(\mathbf{f}_i - \mathbf{f}_j)^\top \mathbf{M}(\mathbf{f}_i - \mathbf{f}_j)\right\}, & \text{if } i \neq j \\ 0, & \text{otherwise,} \end{cases} \tag{1}$$

where $\mathbf{f}_i \in \mathbb{R}^J$ denotes the feature vector corresponding to the i-th data sample and assigned to node i, and $\mathbf{M} \in \mathbb{R}^{J \times J}$ is a symmetric positive-definite metric matrix that could, for example, be used to provide different weights to different features (see, e.g., [31]). A_{ij} captures the Mahalanobis distance [16] between samples in node i and j in the feature space. Note that, finding the optimal 1) $\mathbf{M}$-defined feature distance w.r.t. a graph and 2) A_{ij}-defined sparsity structure of the graph are out of scope of this paper, The reader can refer to [30] and [9] for details.

Next we define a degree matrix $\mathbf{D}$ whose off-diagonals are zeros and diagonals $D_{ii} = \sum_i A_{ij}$. Then *normalized graph Laplacian* matrix is:

$$\mathbf{L} = \mathbf{D}^{-1/2}(\mathbf{D} - \mathbf{A})\mathbf{D}^{-1/2}, \tag{2}$$

with eigen-decomposition $\mathbf{L} = \mathbf{U}\Sigma\mathbf{U}^\top$. We use the normalized Laplacian matrix since common graph kernel functions can be easily approximated using polynomial functions [1], [23], because of its eigenvalue property, that is, the eigenvalues of $\mathbf{L}$ have the property that $0 \leq \lambda_1 < \lambda_2 \leq \cdots \leq \lambda_N \leq 2$, and thus $\mathbf{L} \succeq 0$, *i.e.*, $\mathbf{L}$ is positive semi-definite (PSD) [5], [25].

Graph filters with kernel function $f(x)$ are then defined using eigenvectors $\mathbf{U}$ of $\mathbf{L}$, by

$$\bar{\mathbf{H}} = \mathbf{U} f^\dagger(\Sigma)\mathbf{U}^\top \tag{3}$$

where $f^\dagger(\Sigma) = \text{diag}\{f^{-1}(\lambda_1), \cdots, f^{-1}(\lambda_N)\}$ and $\dagger$ is the pseudo inverse computation [21]. The kernel function can be the diffusion kernel $f(x) = \exp\{\sigma^2\lambda/2\}$, the p-step random walk kernel $f(x) = (a - \lambda)^{-p}$ or Laplacian regularization $f(x) = 1 + \sigma^2\lambda$ [21]. However, $\bar{\mathbf{H}}$ requires explicit eigen-decomposition of $\mathbf{L}$, which is intractable for large graphs. To reduce the computation burden, in the graph kernel and regularization literature [26], one typically adopts polynomial functions $g(x) = \sum_{i=1}^{P} \beta_i x^i$ to approximate the function $f^{-1}(x)$ such that graph filter $\bar{\mathbf{H}}$ can be approximated by another filter with a following finite-degree *polynomial graph kernel* [23], [34]:

$$\mathcal{L} = \mathbf{U}\text{diag}\{\sum_{i=1}^{P} \beta_i \lambda_1^i, \cdots, \sum_{i=1}^{P} \beta_i \lambda_N^i\}\mathbf{U}^\top = \sum_{i=1}^{P} \beta_i \mathbf{L}^i \tag{4}$$

which does not require the explicit eigen-decomposition.

III. Model Selection-inspired Graph Learning

It is clear that the above polynomial graph kernel in Eq. (4) is linearly defined by parameters $\boldsymbol{\beta}$. Given a dataset $\mathcal{D} = \{(\mathbf{f}_i, y_i)\}_{i=1}^N$ with feature vectors $\mathbf{f}_i$'s and scalars y_i's (*e.g.*, binary class labels), we propose to learn the parameters $\boldsymbol{\beta}$ in Eq. (4) via the following objective:

$$\begin{aligned} \min_{\boldsymbol{\beta}} g(\boldsymbol{\beta}) = \min_{\boldsymbol{\beta}} \; & \mathbf{y}^\top \mathcal{L}(\boldsymbol{\beta})\mathbf{y} + \log\det(\mathcal{L}^{-1}(\boldsymbol{\beta})) \\ \text{s.t.} \; & \begin{cases} \mathcal{L} \succ 0 \\ \beta_i \in [-a, a], a \in \mathbb{R}^+, \forall i, \end{cases} \end{aligned} \tag{5}$$

where the left term denotes a data-fit term [20] that is lower bounded by 0 and is widely used in the machine learning literature, and the right term denotes a model complexity penalty term that is generally adopted for model selection in the graph learning [11] and machine learning literature [20]. The second, box constraint on $\boldsymbol{\beta}$ ensures that the model complexity penalty term does not reach minus infinity. Since $\mathcal{L}$ is PSD, we define $\mathcal{L} \leftarrow \mathcal{L} + \mu\mathbf{I}$ throughout this paper to ensure $\mathcal{L}$ is invertible and the computation of the second term is valid, where μ is a very small constant and $\mathbf{I}$ is an identity matrix with proper dimensions.

We give a short proof of the convexity of the objective function Eq. (5). First, since parameters $\boldsymbol{\beta}$ have linear constraints, $\boldsymbol{\beta}$ lie in a convex set. Second, the data-fit term is both convex and concave due to the fact that its Hessian matrix has all-zero entries. Third, the model complexity penalty term can be

written as:

$$\log\det(\mathcal{L}^{-1}) = \log\prod_{k=1}^{N}\left(\sum_{i=1}^{P}\beta_i\lambda_k^i\right)^{-1} = -\sum_{k=1}^{N}\log\left(\sum_{i=1}^{P}\beta_i\lambda_k^i\right), \tag{6}$$

which consists of an affine function $\sum_{i=1}^{P}(\cdot)$, log function $\log(\cdot)$, and another affine function $\sum_{k=1}^{N}(\cdot)$. Since $\sum_{i=1}^{P}(\cdot)$, $\log(\cdot)$ and $\sum_{k=1}^{N}(\cdot)$ are all convex functions, such function composition preserves the convexity [3], and thus the objective in Eq. (5) is convex.

IV. Algorithm Development

Since $g(\boldsymbol{\beta})$ is convex, we can simply set $\nabla g(\boldsymbol{\beta}) = \mathbf{0}$:

$$\nabla g(\boldsymbol{\beta}) = \begin{bmatrix} \mathbf{y}^\top\mathbf{L}\mathbf{y} - \sum_{k=1}^{N}\frac{\lambda_k}{\sum_{i=1}^{P}\beta_i\lambda_k^i+\mu} \\ \vdots \\ \mathbf{y}^\top\mathbf{L}^P\mathbf{y} - \sum_{k=1}^{N}\frac{\lambda_k^P}{\sum_{i=1}^{P}\beta_i\lambda_k^i+\mu} \end{bmatrix} = \mathbf{0}. \tag{7}$$

Therefore,

$$\nabla g(\boldsymbol{\beta}) \odot \begin{bmatrix}\beta_1\\ \vdots \\ \beta_P\end{bmatrix} = \begin{bmatrix} \mathbf{y}^\top\beta_1\mathbf{L}\mathbf{y} - \sum_{k=1}^{N}\frac{\beta_1\lambda_k}{\sum_{i=1}^{P}\beta_i\lambda_k^i+\mu} \\ \vdots \\ \mathbf{y}^\top\beta_P\mathbf{L}^P\mathbf{y} - \sum_{k=1}^{N}\frac{\beta_P\lambda_k^P}{\sum_{i=1}^{P}\beta_i\lambda_k^i+\mu} \end{bmatrix} = \mathbf{0}, \tag{8}$$

where $\odot$ denotes element-wise product. Taking the summation of all entries of Eq. (8), we observe that

$$\mathbf{y}^\top(\sum_{i=1}^{P}\beta_i\mathbf{L}^i)\mathbf{y} - \sum_{k=1}^{N}\frac{\sum_{i=1}^{P}\beta_i\lambda_k^i}{\sum_{i=1}^{P}\beta_i\lambda_k^i+\mu} = 0. \tag{9}$$

Therefore, the global minimum of Eq. (5) occurs when

$$\mathbf{y}^\top\mathcal{L}\mathbf{y} = \sum_{k=1}^{N}\frac{\sum_{i=1}^{P}\beta_i\lambda_k^i}{\sum_{i=1}^{P}\beta_i\lambda_k^i+\mu} \approx \sum_{k=1}^{N}1 = N. \tag{10}$$

We note that Eq. (10) is a necessary but not sufficient condition of Eq. (7), *i.e.*, we solve the original problem in Eq. (5) via relaxation by Eq. (10) that enlarges the solution search space compared to the optimality condition in Eq. (7). We also observe that the PD-cone constraint $\mathcal{L} \succ 0$ is equivalent to:

$$\begin{cases} \sum_{i=1}^{P}\beta_i\lambda_1^i + \mu > 0 \\ \vdots \\ \sum_{i=1}^{P}\beta_i\lambda_N^i + \mu > 0. \end{cases} \tag{11}$$

Therefore, solving Eq. (5) essentially boils down to iteratively solving the following problem:

$$\begin{aligned} \min_{\boldsymbol{\beta}} h(\boldsymbol{\beta}) = \min_{\boldsymbol{\beta}} \; & -\sum_{k=1}^{N}\log\left(\sum_{i=1}^{P}\beta_i\lambda_k^i\right) \\ \text{s.t. } & \begin{cases} \sum_{i=1}^{P}\beta_i\lambda_1^i + \mu > 0 \\ \vdots \\ \sum_{i=1}^{P}\beta_i\lambda_N^i + \mu > 0 \\ \beta_i \in [-a, a], a \in \mathbb{R}^+, \forall i \\ \mathbf{y}^\top\left(\sum_{i=1}^{P}\beta_i\mathbf{L}^i\right)\mathbf{y} = N. \end{cases} \end{aligned} \tag{12}$$

Since all constraints in Eq. (12) are linear, one can solve it by projected gradient descent [4] with convergence rate $\mathcal{O}(1/t)$ but at the cost of projection after each gradient descent step. Here, we instead solve it via a projection-free Frank-Wolfe method with the same convergence rate as the projected gradient descent [15] as follows. Let the convex set defined by the linear constraints in Eq. (12) be $\mathcal{H}$. At Frank-Wolfe iteration t, we compute the gradient of the objective $\nabla h(\boldsymbol{\beta})$, and then solve the following direction-finding subproblem:

$$\begin{aligned} \min_{\mathbf{s}} \; & \mathbf{s}^\top\nabla h(\boldsymbol{\beta}^{t-1}) \\ \text{s.t. } & \mathbf{s} \in \mathcal{H}, \end{aligned} \tag{13}$$

where $\boldsymbol{\beta}^{t-1}$ denotes the previous solution of (13) and $\mathbf{s}^\top\nabla h(\boldsymbol{\beta})$ is the first-order approximation of $h(\boldsymbol{\beta})$ at $\boldsymbol{\beta}$. Eq. (13) is a linear program that can be efficiently solved using simplex [18] or interior-point method [4]. Next, we choose a step size α^t that solves the following optimization problem:

$$\alpha^t = \arg\min_{\alpha\in[0,1]} h(\alpha), \tag{14}$$

where $h(\alpha) = h(\boldsymbol{\beta}^{t-1} + \alpha(\mathbf{s}^t - \boldsymbol{\beta}^{t-1}))$. Since Eq. (14) is twice differentiable, it can be solved efficiently via a Newton-Raphson method [19]:

$$\alpha^t = \alpha^{t-1} - \frac{\partial h(\alpha)}{\partial\alpha}\Big/\frac{\partial^2 h(\alpha)}{\partial\alpha^2}, \tag{15}$$

where

$$\begin{aligned} \frac{\partial h(\alpha)}{\partial\alpha} &= \mathbf{y}^\top\left(\sum_{i=1}^{P}(s_i^t - \beta_i^{t-1})\mathbf{L}^i\right)\mathbf{y} - \sum_{k=1}^{N}\frac{\sum_{i=1}^{P}(s_i^t-\beta_i^{t-1})\lambda_k^i}{\sum_{i=1}^{P}\beta_i^t\lambda_k^i+\mu} \\ \frac{\partial^2 h(\alpha)}{\partial\alpha^2} &= \sum_{k=1}^{N}\frac{(\sum_{i=1}^{P}(s_i^t-\beta_i^{t-1})\lambda_k^i)^2}{(\sum_{i=1}^{P}\beta_i^t\lambda_k^i+\mu)^2}. \end{aligned} \tag{16}$$

We iteratively solve Eq. (13) and Eq. (14) until convergence. We summarize our proposed MSGL method in Algorithm 1. Together with a one-time eigen-decomposition of $\mathbf{L}$ to calculate $\mathbf{U}$ and $\lambda_1, ..., \lambda_N$, our algorithm has a time complexity $\mathcal{O}(N^3 + lPN)$ with a P-degree polynomial graph spectral kernel and l iterations of Eq. (13).

Algorithm 1 The proposed MSGL

Input: $\mathcal{D} = \{(\mathbf{f}_i, y_i)\}_{i=1}^N$, μ, P.
Output: $\boldsymbol{\beta}^*$.

Construct $\mathbf{L}$ using $\mathbf{f}$ via Eqs. (1) and (2).
Get eigenpairs $\mathbf{U}$ and $\lambda_1, ..., \lambda_N$ of $\mathbf{L}$ via eigen-decomposition.
Construct $\mathcal{L}$ with $\mathbf{U}$, $\lambda_1, ..., \lambda_N$, μ, P, and randomly initialized $\boldsymbol{\beta}$ via Eq. (4).
while *not converged* **do**:
  Solve Eq. (13) for $\mathbf{s}^t$ via interior-point.
  while *not converged* **do**:
  Solve Eq. (14) for α^t via a Newton-Raphson method.
  end while
  $\boldsymbol{\beta}^t = \boldsymbol{\beta}^{t-1} + \alpha^t(\mathbf{s}^t - \boldsymbol{\beta}^{t-1})$.
end while

V. Experimental Results

We implemented our MSGL graph learning scheme in Matlab R2020b[1], and evaluated it in terms of average classification error rate and running time. Let $\mathcal{L} = [\mathcal{L}_{ll}\ \mathcal{L}_{lu}; \mathcal{L}_{ul}\ \mathcal{L}_{uu}]$, where $\mathcal{L}_{ll}$ denotes the submatrix of $\mathcal{L}$ that corresponds to the data samples with known labels, $\mathcal{L}_{uu}$ corresponds to the data samples with predicted labels, and $\mathcal{L}_{ul}$ corresponds to the data samples with predicted labels row-wise and data samples with known labels column-wise. We compared our algorithm against the following graph learning schemes: i) PDcone: standard gradient descent of the convex objective $\hat{\mathbf{y}}^\top \mathcal{L}_{ll}(\mathbf{M})\hat{\mathbf{y}}$ with projection of $\mathbf{M}$ in Eq. (1) onto a PD cone, where $\hat{\mathbf{y}}$ denotes the known labels (the training set), 2) HBNB: a recent convex graph learning scheme of the objective $\hat{\mathbf{y}}^\top \mathcal{L}_{ll}(\mathbf{M})\hat{\mathbf{y}}$ using block coordinate descent with proximal gradient and adopting restricted search spaces that are intersections of half spaces, boxes and norm balls for $\mathbf{M}$ [14], 3) PGML: a convex graph learning scheme of the objective $\hat{\mathbf{y}}^\top \mathcal{L}_{ll}(\mathbf{M})\hat{\mathbf{y}}$ that is based on Gershgorin disc perfect alignment [30] within a positive graph metric space for $\mathbf{M}$ [32], 4) SGML: a convex graph learning scheme of the objective $\hat{\mathbf{y}}^\top \mathcal{L}_{ll}(\mathbf{M})\hat{\mathbf{y}}$ similar to 3) but within a balanced signed graph metric space for $\mathbf{M}$ [30], and 5) Cholesky: a non-convex approach of the objective $\hat{\mathbf{y}}^\top \mathcal{L}_{ll}(\mathbf{Q}\mathbf{Q}^\top)\hat{\mathbf{y}}$ via gradient descent, where $\mathbf{Q}$ denotes a lower-triangular matrix [12] and $\mathbf{Q}\mathbf{Q}^\top$ has the same functionality as $\mathbf{M}$ that defines the pairwise feature distance in Eq. (1). See Table I for the objective functions, optimization variables and time complexities of the competing schemes. Note that the objectives for PDcone, HBNB, PGML, SGML, and Cholesky only consist of the $\mathcal{L}_{ll}$ and $\hat{\mathbf{y}}$ that correspond to the data samples with known labels, which is the standard setting to learn a metric $\mathbf{M}$ or $\mathbf{Q}\mathbf{Q}^\top$ [30]. Our proposed MSGL first requires the full set of labels $\mathbf{y} = [\hat{\mathbf{y}}\ \mathbf{y}_u^*]$ where $\mathbf{y}_u^*$ is predicted with a graph-based classifier [31] given by $\mathbf{y}_u^* = -\mathcal{L}_{uu}^\dagger \mathcal{L}_{ul}\hat{\mathbf{y}}$, then optimizes $\boldsymbol{\beta}$ using the full $\mathcal{L}$ together with $\mathbf{y}$ via Eq. (12), so that the optimized $\boldsymbol{\beta}$ ensures that $\mathcal{L} \succ 0$.

We set the maximum number of main iterations, convergence thresholds and the degree of polynomial function $\mathcal{L}$ for all evaluated schemes to 100, 0.01, and $P = 3$, respectively. Finding the optimal gradient descent step size based on Lipschitz constant for datasets with large per-sample feature size is computationally infeasible on a consumer-level machine. Therefore, as in [27], [30], the step size of gradient descent for PDcone, HBNB and Cholesky was heuristically initialized to $0.1/N$, increased by 1% if GD yielded a better objective value, and decreased by half otherwise. We chose these settings since smaller convergence thresholds and larger number of iterations would cause PDcone, HBNB, PGML, SGML, and Cholesky to be significantly slower to converge. Without loss of generality, we set $\mathbf{M} = \mathbf{I}$ during graph construction using Eq. (1) in our method. We heuristically set $\mu = 10^{-8}, a = 100$ when solving Eq. (12) throughout the experiments. We used default settings for the remaining parameters of all schemes. All computations were carried out in Matlab R2020b on a Windows 10 64bit PC with AMD RyzenThreadripper 3960X 24-core processor 3.80 GHz and 128GB of RAM.

As shown in Table II, we adopted 17 binary datasets that are freely available in UCI[2] and LibSVM[3] databases. We conducted two sets of experiments as follows. For the first set of experiments, we created 10 instances of 60% training-40% test split with random seeds 0-9 [22] for the first 15 out of 17 datasets, and evaluated each graph learning scheme via averaged classification accuracy and runtime. Given a learnt $\mathcal{L}$ (defined by a learnt $\mathbf{M}$ for PDcone, HBNB, PGML and SGML, a learnt $\mathbf{Q}$ for Cholesky, and a learnt $\boldsymbol{\beta}$ for our proposed MSGL), we used the same graph-based classifier [31] $\mathbf{y}_u^* = -\mathcal{L}_{uu}^\dagger \mathcal{L}_{ul}\hat{\mathbf{y}}$ for classification of all evaluated schemes, where $\mathbf{y}_u$ denotes the predicted labels. For the second set of experiments, we focus on the evaluation of runtime. Specifically, we evaluated the fastest competitor PGML and our proposed MSGL with various data sizes using the 16th dataset `Madelon` and evaluated these two methods with various per-sample feature sizes using the 17th dataset `Colon-cancer`. For both sets, we applied the data normalization scheme of [6] for the training/test data, which first subtracts the mean and divides by the feature-wise standard deviation, and then normalizes to unit length sample-wise. We added Gaussian noise with variance 10^{-12} noise to the dataset to avoid NaN's due to data normalization on small samples.

[1]The code is available at https://github.com/bobchengyang/MSGL.

[2]https://archive.ics.uci.edu/ml/datasets.php

[3]https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html

TABLE I: The objective functions, optimization variables and time complexities of the proposed and competing schemes. l denotes the number of iterations of any components. J denotes the per-sample feature size. c denotes the number of non-zero entries in $\mathbf{M}$. N denotes the data size. P denotes the degree of the polynomial function.

method	objective	var.	time complexity
PDcone	$\hat{\mathbf{y}}^\top \mathcal{L}_{ll}(\mathbf{M})\hat{\mathbf{y}}$	$\mathbf{M}$	$\mathcal{O}(lJ^3)$
HBNB			$\mathcal{O}(l(lJ+l(lc+J-1)))$
PGML			$\mathcal{O}(l(lc+J(J+1)/2))$
SGML			$\mathcal{O}(lc+(J^2+5J)/2-1)$
Cholesky	$\hat{\mathbf{y}}^\top \mathcal{L}_{ll}(\mathbf{Q}\mathbf{Q}^\top)\hat{\mathbf{y}}$	$\mathbf{Q}$	$\mathcal{O}(lJ(J+1)/2)$
MSGL	Eq. (12)	$\boldsymbol{\beta}$	$\mathcal{O}(N^3+lPN)$

TABLE II: Experimental datasets.

dataset	(N, J)
Australian	(690,14)
Breast-cancer	(683,10)
Diabetes	(768,8)
Fourclass	(862,2)
German	(1000,24)
Haberman	(306,3)
Heart	(270,13)
ILPD	(583,10)
Liver-disorders	(145,5)
Monk1	(556,6)
Pima	(768,8)
Planning	(182,12)
Voting	(435,16)
WDBC	(569,30)
Sonar	(208,60)
Madelon	(2000,500)
Colon-cancer	(62,2000)

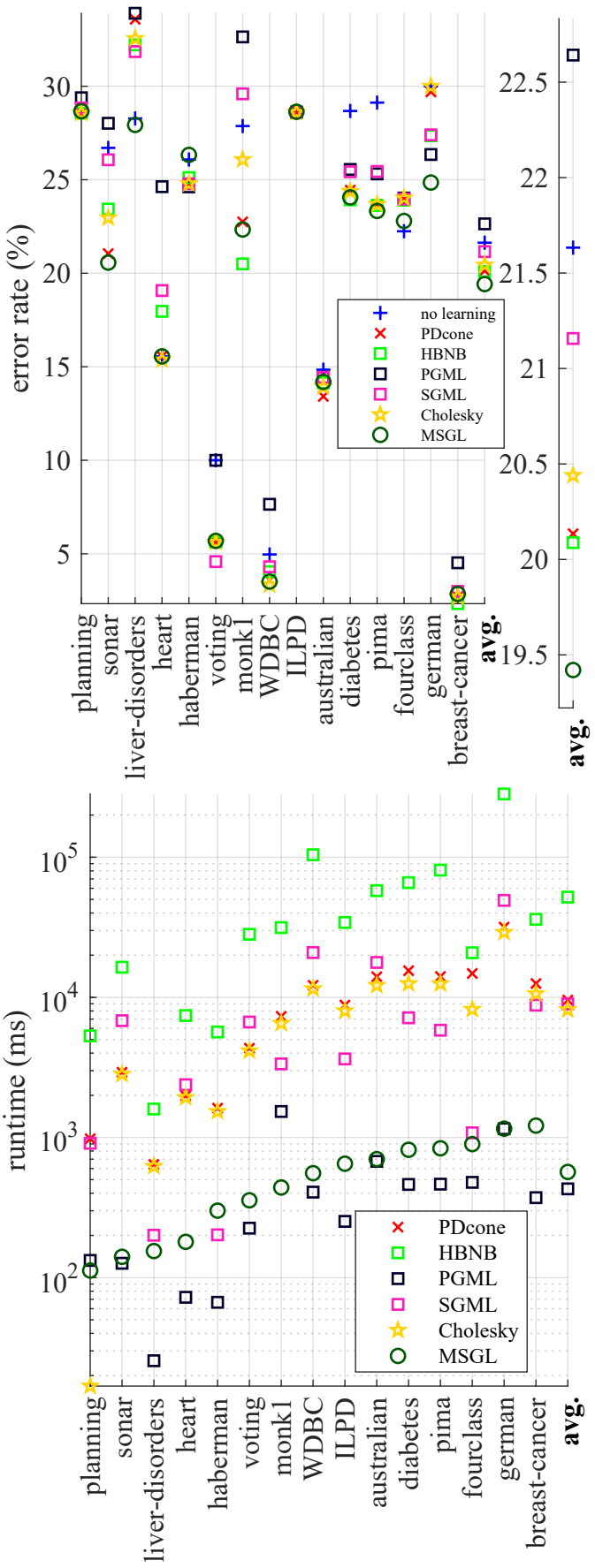

Fig. 1: Error rate (upper) and runtime (lower) on 15 datasets.

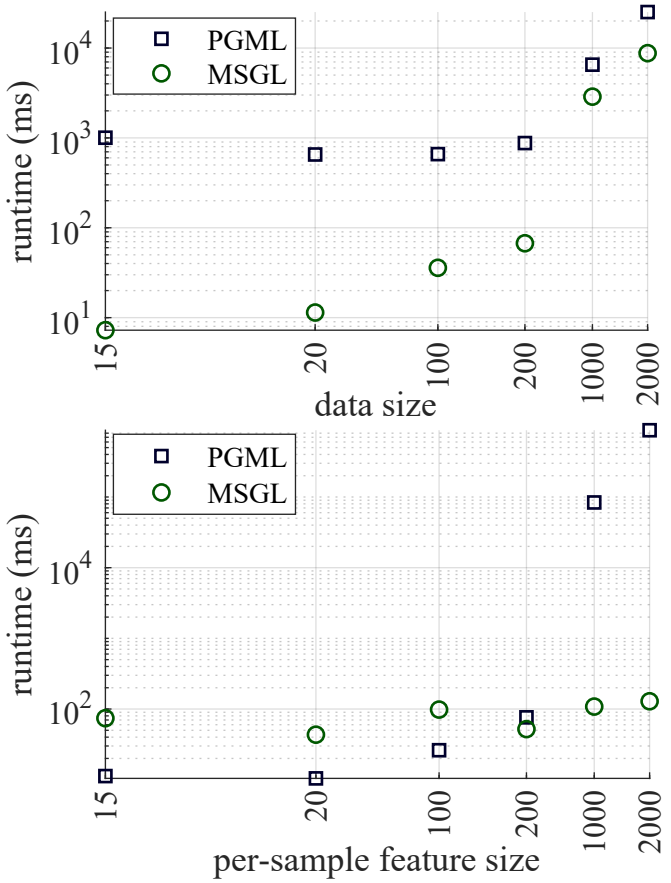

Fig. 2: Runtime for `Madelon` (upper) and `Colon-cancer` (lower) with various data and feature sizes, respectively.

Fig. 1 shows classification error rates and runtime (in log scale) for the first 15 out of 17 datasets in Table II, respectively. The horisontal axis of each plot denotes the datasets in ascending order of the runtime of our proposed MSGL. Each point in the plots denotes the average of 10 runs. Fig. 2 shows runtime versus data size N on `Madelon` and runtime versus per-sample feature size J on `Colon-cancer`, respectively. We did not execute PDcone, HBNB, SGML or Cholesky in Fig. 2 since PGML is the fastest competitor to MSGL.

In terms of classification error rate, PGML, SGML, Cholesky, PDcone and HBNB had relatively larger error rates: 22.64%, 21.16%, 20.44%, 20.13% and 20.08%, respectively. PGML is even worse than direct classification without learning of $\mathbf{M}$ or $\boldsymbol{\beta}$ (*i.e.*, $\mathbf{M} = \mathbf{I}$ and $\boldsymbol{\beta} = \mathbf{1}$), which is due to the fact that positive graph search space is much more restricted than the other schemes. SGML occasionally performs worse than direct classification, *e.g.*, `Monk1`, which is due to a restricted, balanced signed graph search space. Due to the non-convexity of Cholesky graph learning scheme, it occasionally converges at a bad local minimum, and thus performs worse than direct classification, *e.g.*, for `liver-disorders` dataset. MSGL achieved the lowest 19.42% classification error rate among all schemes. This shows that, with the polynomial coefficients $\boldsymbol{\beta}$ being the only optimization variables, MSGL successfully extracts the data similarity without learning the pairwise distance metric $\mathbf{M}$ directly.

In terms of runtime, PGML was competitive to MSGL, but MSGL significantly outperformed the other competing schemes on average. As shown in Fig. 1, the speed gains are 91.47x, 16.85x, 15.84x and 14.38x for MSGL vs HBNB, PDcone, SGML and Cholesky, respectively. As shown in the upper sub-figure in Fig. 2, the runtime of both PGML and our proposed MSGL increase dramatically as the data size increases with a 500 per-sample feature size on `Madelon`. This is due to the graph construction with a large data size is generally time-consuming. However, as shown in the lower sub-figure in Fig. 2, the runtime of MSGL remains similar across various per-sample feature sizes with a 62 data size on `Colon-cancer`, whereas the runtime of PGML again increases dramatically as per-sample feature size increases. This means that, with a small data size, the speed gain of our MSGL increased as per-sample feature size increased. The reason for our significant speed gain is the very few optimization variables $\boldsymbol{\beta}$ that correspond to the polynomial function $\mathcal{L}$. For PDcone, HBNB, PGML, SGML and Cholesky, the number of optimization variables in $\mathbf{M}$ or $\mathbf{U}$ scales dramatically with the per-sample feature size.

VI. Conclusion

Inspired by recent advances in GSP, we proposed a model selection-inspired graph learning (MSGL) method that learns the parameters $\boldsymbol{\beta}$ of a polynomial function $\mathcal{L} = \sum_{i=1}^{P} \beta_i \mathbf{L}^i \succ 0$. We first formulate a convex optimization problem that consists of a graph Laplacian regularizer with $\mathcal{L}$ and a log determinant term with the covariance $\mathcal{L}^{-1}$. Then, by replacing the positive-definite cone constraint for $\mathcal{L}$ with a set of linear constraints for $\boldsymbol{\beta}$, we employ the Frank-Wolfe method and iteratively solve a linearized objective efficiently. Our algorithm requires eigen-decomposition of a given graph Laplacian $\mathbf{L}$ only once. Experimental results confirms the competitive performance of the proposed MSGL against state-of-the-art graph learning methods, both in classification accuracy and speed. In practice, it is always desirable to construct a graph with properly defined feature distance and sparsity stricture prior to employing the proposed method. Future work will focus on 1) the combination of the proposed method and finding the optimal $\mathbf{M}$-defined feature distance w.r.t. a graph and A_{ij}-defined sparsity structure of the graph, 2) improvement of the model scalability in terms of data size, and 3) a wider range of applications beyond classification, such as regression, clustering and low-level image processing.

References

[1] A. Anis, A. El Gamal, A. S. Avestimehr, and A. Ortega, "A sampling theory perspective of graph-based semi-supervised learning," *IEEE Transactions on Information Theory*, vol. 65, no. 4, pp. 2322–2342, Apr. 2019.

[2] M. Belkin, I. Matveeva, and P. Niyogi, "Regularization and semisupervised learning on large graphs," in *Shawe-Taylor J., Singer Y. (eds) Learning Theory, COLT 2004, Lecture Notes in Computer Science*, vol. 3120, 2004, pp. 624–638.

[3] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, 2004.

[4] ——, *Convex Optimization*. Cambridge University Press, 2009.
[5] G. Cheung, E. Magli, Y. Tanaka, and M. Ng, "Graph spectral image processing," in *Proceedings of the IEEE*, vol. 106, no.5, May 2018, pp. 907–930.
[6] M. Dong, Y. Wang, X. Yang, and J. Xue, "Learning local metrics and influential regions for classification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 6, pp. 1522–1529, June 2020.
[7] X. Dong, D. Thanou, M. Rabbat, and P. Frossard, "Learning graphs from data: A signal representation perspective," *IEEE Signal Processing Magazine*, vol. 36, no. 3, pp. 44–63, May 2019.
[8] F. Dornaika and Y. El Traboulsi, "Joint sparse graph and flexible embedding for graph-based semi-supervised learning," *Neural Networks*, vol. 114, pp. 91–95, 2019.
[9] H. Egilmez, E. Pavez, and A. Ortega, "Graph learning from data under laplacian and structural constraints," in *IEEE Journal of Selected Topics in Signal Processing*, vol. 11, no.6, July 2017, pp. 825–841.
[10] X. Fang, Y. Xu, X. Li, Z. Lai, and W. K. Wong, "Learning a nonnegative sparse graph for linear regression," *IEEE Transactions on Image Processing*, vol. 24, no. 9, pp. 2760–2771, 2015.
[11] J. Friedman, T. Hastie, and R. Tibshirani, "Sparse inverse covariance estimation with the graphical lasso," in *Biostatistics*, vol. 9, no.3, 2008, pp. 432–441.
[12] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed. The Johns Hopkins University Press, 1996.
[13] X. Han, P. Liu, L. Wang, and D. Li, "Unsupervised feature selection via graph matrix learning and the low-dimensional space learning for classification," *Engineering Applications of Artificial Intelligence*, vol. 87, p. 103283, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0952197619302453
[14] W. Hu, X. Gao, G. Cheung, and Z. Guo, "Feature graph learning for 3D point cloud denoising," *IEEE Transactions on Signal Processing*, vol. 68, pp. 2841–2856, 2020.
[15] M. Jaggi, "Revisiting Frank-Wolfe: Projection-free sparse convex optimization," in *International Conference on Machine Learning*, Jun 2013, pp. 427–435.
[16] P. C. Mahalanobis, "On the generalized distance in statistics," *Proceedings of the National Institute of Sciences of India*, vol. 2, no. 1, pp. 49–55, April 1936.
[17] A. Ortega, P. Frossard, J. Kovacevic, J. M. F. Moura, and P. Vandergheynst, "Graph signal processing: Overview, challenges, and applications," in *Proceedings of the IEEE*, vol. 106, no.5, May 2018, pp. 808–828.
[18] C. Papadimitriou and K. Steiglitz, *Combinatorial Optimization*. Dover Publications, Inc, 1998.
[19] J. Raphson, *Analysis aequationum universalis*, London, 1690.
[20] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.
[21] D. Romero, M. Ma, and G. B. Giannakis, "Kernel-based reconstruction of graph signals," *IEEE Transactions on Signal Processing*, vol. 65, no. 3, pp. 764–778, Feb. 2017.
[22] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. USA: Prentice Hall Press, 2009.
[23] A. Sakiyama, Y. Tanaka, T. Tanaka, and A. Ortega, "Eigendecomposition-free sampling set selection for graph signals," *IEEE Transactions on Signal Processing*, vol. 67, no. 10, pp. 2679–2692, May 2019.
[24] S. E. Schaeffer, "Survey: Graph clustering," *Computer Science Review*, vol. 1, no. 1, p. 27–64, Aug. 2007.
[25] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," in *IEEE Signal Processing Magazine*, vol. 30, no.3, May 2013, pp. 83–98.
[26] A. J. Smola and R. Kondor, "Kernels and regularization on graphs," in *Learning Theory and Kernel Machines*, B. Schölkopf and M. K. Warmuth, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 144–158.
[27] K. Q. Weinberger and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification," *Journal of Machine Learning Research*, vol. 10, no. 2, pp. 207–244, Feb. 2009.
[28] X. Wu, L. Zhao, and L. Akoglu, "A quest for structure: Jointly learning the graph structure and semi-supervised classification," *ACM International Conference on Information and Knowledge Management*, 2018.
[29] E. P. Xing, M. I. Jordan, S. J. Russell, and A. Y. Ng, "Distance metric learning with application to clustering with side-information," in *Annual Conference on Neural Information Processing Systems*, Dec. 2003, pp. 521–528.
[30] C. Yang, G. Cheung, and W. Hu, "Signed graph metric learning via gershgorin disc perfect alignment," *arXiv*, 2021.
[31] C. Yang, G. Cheung, and V. Stankovic, "Alternating binary classifier and graph learning from partial labels," in *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference*, Nov. 2018, pp. 1137–1140.
[32] C. Yang, G. Cheung, and W. Hu, "Graph metric learning via Gershgorin disc alignment," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2020.
[33] G. Zhai and X. Min, "Perceptual image quality assessment: a survey," *Science China Information Sciences*, vol. 63, no. 211301, pp. 1–52, Nov. 2020.
[34] Y.-C. Zhi, Y. Ng, and X. Dong, "Gaussian processes on graphs via spectral kernel learning," *arXiv*, 2020.
[35] X. Zhou and M. Belkin, "Semi-supervised learning by higher order regularization," in *International Conference on Artificial Intelligence and Statistics, JMLR Workshop and Conference Proceedings*, Apr. 2011, pp. 892–900.
[36] X. Zhu, Z. Ghahramani, and J. Lafferty, "Semi-supervised learning using Gaussian fields and harmonic functions," in *International Conference on Machine Learning*, Aug. 2003, pp. 912–919.