Efficient Semi-Supervised Learning on Locally Informative Multiple Graphs

Motoki Shiga*[,1], and Hiroshi Mamitsuka[1]

[1]Bioinformatics Center, Kyoto University, Gokasho, Uji 611-0011, Japan

E-mail:{shiga,mami}@kuicr.kyoto-u.ac.jp

Abstract

We address an issue of semi-supervised learning on multiple graphs, over which informative subgraphs are distributed. One application under this setting can be found in molecular biology, where different types of gene networks are generated depending upon experiments. Here an important problem is to annotate unknown genes by using functionally known genes, which connect to unknown genes in gene networks, in which informative parts vary over networks. We present a powerful, time-efficient approach for this problem by combining soft spectral clustering with label propagation for multiple graphs. We demonstrate the effectiveness and efficiency of our approach using both synthetic and real biological datasets.

Keywords: Semi-supervised learning, graph integration, label propagation, soft spectral clustering, EM (Expectation Maximization) algorithm

1 Introduction

We address the following problem. We have two inputs: 1) a set of labeled and unlabeled examples, and 2) multiple (and different) graphs (or networks), in which each node corresponds to an example and each edge indicates similarity between two linked corresponding nodes. We then attempt to estimate labels of unlabeled examples by using connectivity of given multiple graphs. Fig. 1 (a) and (b) show an example of our input with two different graphs, which share the same set of nodes, some part being labeled (by green or red) and the rest being unlabeled (or no colors). This problem is a kind of semi-supervised learning, which is practically useful because labeled examples are hard to get in real world [1, 2, 3]. Our problem is in graph-based semi-supervised learning, which can be classified into two types [4]: 1) supervised learning with unlabeled examples (semi-supervised classification) [1, 2] and 2) unsupervised learning with constraints (semi-supervised clustering) [5, 6]. Our problem setting is in the first type but our proposed approach is slightly related with the second type. We note that usual graph-based semi-supervised learning has one single graph as an input, whereas multiple graphs are an input of our problem.

We can find this problem setting in a lot of applications. One example is gene annotation, in which some genes are already annotated but most of all genes are still functionally unknown, despite of all experimental efforts in molecular biology [7, 8, 9]. However, the recent development of experimental techniques in biology provides us with a rich number of gene networks, which can be derived from different types of experiments, such as cDNA microarray, protein-protein interactions and ChIP-on-Chip, which are all gene affinity networks but capture different features of gene similarity [10, 11, 12, 13]. Thus the purpose in this application is to assign functions of

*Corresponding author. Current address: Department of Computer Science and Engineering, Toyohashi University of Technology, 1-1 Hibarigaoka Tenpaku-cho, Toyohashi 441-8580, Japan, Phone:+81-532-44-6876

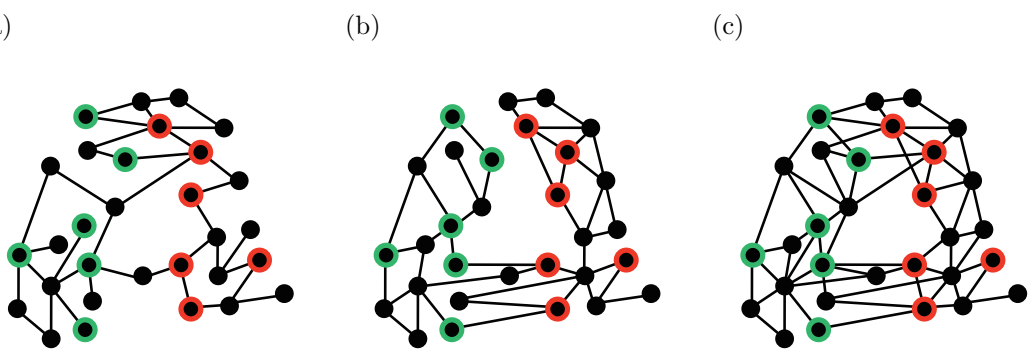

Figure 1: (a) and (b): Two example graphs of our input, and (c) A graph obtained by equally integrating (a) and (b) simply.

unlabeled genes by using known genes and different graphs. Classifying web pages can be also another example [14, 15]. A huge amount of homepages are in the Web, making us hard to assign labels to all of them, but we can have multiple networks over the Web, such as hyperlinks and a content similarity network, all indicating the similarity of web pages. Thus our objective here is to assign class labels to unlabeled Web pages by using a small number of labeled pages and given multiple graphs. Another example is to classify scientific papers, accumulated in academic databases, such as PubMed, CAS (Chemical Abstract Service) and Citeseerx etc [16]. In fact, currently PubMed has over 18 million papers, which far exceeds a manually tractable number to classify them, but we can generate a lot of networks on these publications, such as the co-author network, citation network, co-citation network and content-based network, etc., from the information in PubMed. The purpose of this application is then to assign class labels to unlabeled articles by using these graphs and a small number of labeled documents.

Since our problem setting assumes multiple graphs as an input, we cannot directly apply an usual graph-based semi-supervised learning method to our setting, because such a method accepts only one graph as its input. A simple way to use such an existing method is to merge given graphs into a single graph. However, graphs differ in reliability, meaning that each graph should be weighted, according to its consistency with labeled nodes. This case, the problem is to estimate labels of unlabeled examples as well as weights (which hereafter we call *graph weights*), each showing the significance of the corresponding graph. Graph weights are reasonable but practically it is not sufficient to use graph weights only. For example, Fig. 1 (c) shows one example of simply integrating two graphs of Fig. 1 (a) and (b) equally. Here each of (a) and (b) has densely connected parts (or subgraphs) with high consistency in labels by which unlabeled nodes can be predicted. However (c), a simple integration of (a) and (b), connects nodes almost uniformly over the graph, by which information for labeling unknown nodes is rather lost. This problem cannot be solved even if we use weights for two graphs: (a) and (b). Instead, in order to address this problem, we need to think about that parts of a given graph have different reliability. In fact, given a gene network from a cDNA microarray experiment, it is natural that a certain subgraph (or a particular set of genes) is densely connected with each other and reliable, depending on the experimental condition under which this cDNA microarray was obtained. This indicates that graphs can have more reliable (or informative) subgraphs than other parts. For example, we can easily decompose Fig.1 (a) into three subgraphs shown in Fig. 2 (a-1) to (a-3) in terms of edge connectively. Here we can see that labels in (a-1) are relatively consistent, i.e. that nodes are basically colored by green in (a-1). This means that (a-1) is reliable. Similarly nodes are all colored by red in (a-3), meaning that (a-3) is totally consistent and reliable. On the other

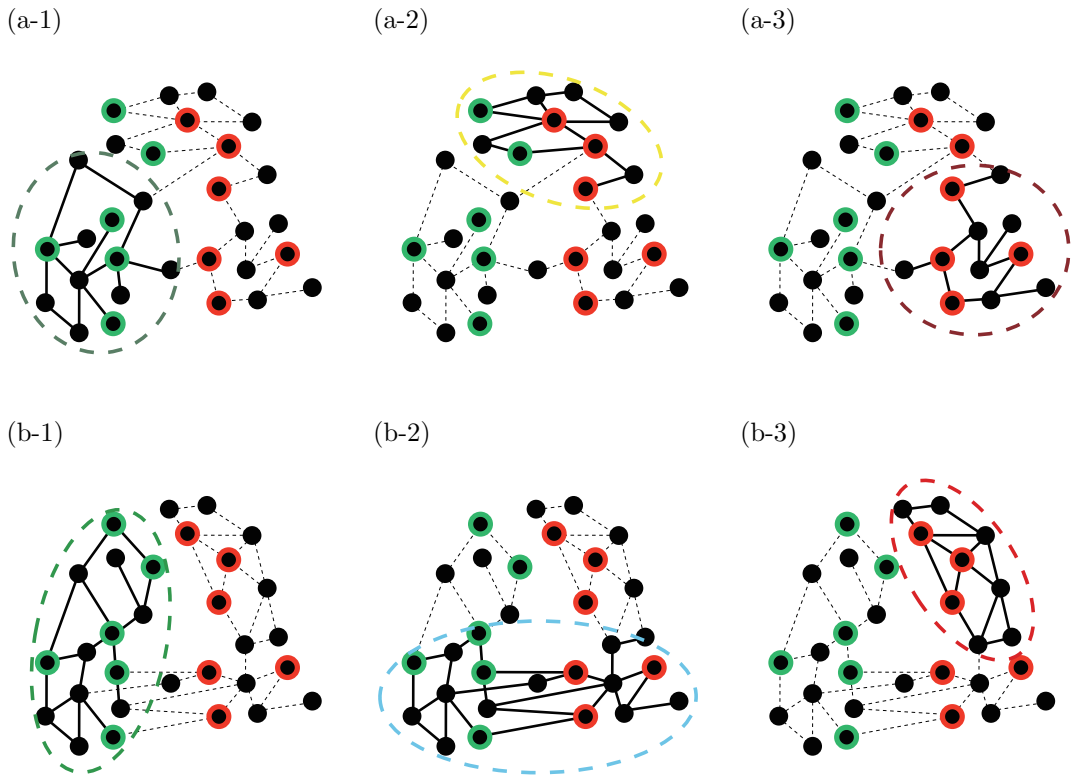

Figure 2: (a-1)-(a-3): Three subgraphs decomposed from Fig. 1 (a), and (b-1)-(b-3): Three subgraphs decomposed from Fig. 1 (b).

hand, labels of nodes in (a-2) are inconsistent, meaning that this is an unreliable subgraph. We hereafter call reliable subgraphs like (a-1) and (a-3), *locally informative subgraphs*, and we further call graphs with locally informative subgraphs *locally informative graphs*. Thus subgraphs in Fig. 2 (b-1) to (b-3), which can be derived from Fig.1 (b) by edge connectively, indicates that (b-1) and (b-3) are locally informative subgraphs because of their high label consistency but (b-2) is not a locally informative subgraph.

We propose an efficient approach for our problem setting, allowing to consider the situation in which locally informative subgraphs can be overlapped with each other in each given graph. Our approach is based on a combination of soft spectral clustering and label propagation. More concretely, we first decompose each given graph into eigenvector spectra by which multiple graphs, representing latent clusters, can be generated. We perform this spectral decomposition over all given graphs and then estimate graph weights over all generated graphs by using efficient label propagation algorithm for multiple graphs [17]. Each graph weight shows the significance of the corresponding clustering component (or subgraph), meaning that our method can handle the situation in which multiple locally informative subgraphs are overlapped with each other in each given graph. In fact, spectral decomposition of our approach corresponds to that the graph of Fig. 1 (a) is decomposed into three subgraphs of Fig. 2 (a-1) to (a-3). Then (a-1) and (a-3) are locally informative subgraphs, while (a-2) is not, and thus we can assign relatively high weights to (a-1) and (a-3) and a low weight to (a-2). We can conduct the same procedure to Fig. 1 (b), which is first decomposed into Fig. 2 (b-1) to (b-3), and we can then assign high weights to (b-1) and (b-3) because they are locally informative subgraphs and a low weight to (b-2). Finally we

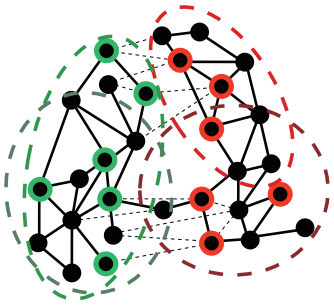

Figure 3: A graph obtained by integrating six graphs in Fig. 2 with graph weights showing label consistency.

can integrate all subgraphs, i.e. (a-1) to (a-3) and (b-1) to (b-3) with their weights by using label propagation for multiple graphs. The integrated graph can be shown in Fig. 3, where each edge reflects upon the total sum of graph weights assigned to graphs with this edge. Note that as shown by colored circles, the integrated graph of Fig. 3 has four subgraphs, corresponding four locally informative subgraphs, which are obtained in (a-1), (a-3), (b-1) and (b-3) of Fig. 2. The final graph given in Fig. 3 can easily assign labels to unknown nodes by their connected edges, while it would be not so easy by using the simply integrated graph in Fig. 1 (c). This example shows the high potentiality and effectiveness of the proposed approach under our problem setting. We stress that our problem setting has not been considered and tackled by any existing method in graph-based semi-supervised learning. Furthermore, despite of the simpleness, our approach, i.e. the combination of spectral decomposition with label propagation over multiple graphs, is reasonable for our purpose in the literature of semi-supervised learning:

1. A disadvantage of current graph-based semi-supervised learning methods (including label propagation) is that they are biased to the connectivity of graphs, meaning that local (or node neighboring) information is emphasized. More concretely, graph-based semi-supervised learning is an optimization problem to minimize a classification error plus a graph cost on the label consistency between neighboring nodes, i.e. neighboring information. On the other hand, spectral clustering pays attention to more global information, which can be a complement to the drawback of graph-based semi-supervised learning.

2. The number of subgraphs (or components in clustering) in each graph is *a priori* unknown and hard to decide. However, by using spectral decomposition, we can easily focus on major cluster structures more by using eigenvectors with lower eigenvalues, which allows to partially solve the problem of determining the number of clusters.

3. Both the soft spectral clustering and the label propagation are time-efficient, making our method applicable to a large-scale network in real world, regardless of our complicated problem setting. In particular, the real amount of computation of our method is linear in the number of examples, which is far more time-efficient than a kernel-based approach of using quadratic programming.

4. Our method outputs not only label scores but also graph weights for all generated graphs, corresponding to latent clusters in given graphs. This allows us to easily find what clusters (or subparts) are the most informative and significant in classification. This is an important aspect in terms of knowledge discovery and data mining, and has not been considered by other graph-based semi-supervised learning methods.

We empirically demonstrate the performance and efficiency of our proposed approach by using a variety of datasets, including synthetic and real datasets. Experimental results clearly showed the significant performance achievements, the time-efficiency over a large network and the output comprehensibility of our method.

This paper is organized as follows: Section 2 summarizes the current status on semi-supervised or supervised learning over multiple graphs. In Section 3, we start reviewing two important components of our proposed method: soft spectral clustering and label propagation over multiple graphs. We then present our efficient approach for multiple graphs with locally informative subgraphs which combines label propagation with spectral clustering. We further examine the computational complexities of our approach, comparing with that of kernel-based approaches. In Section 4, we validate the effectiveness and efficiency of the proposed approach by using synthetic datasets including locally informative subgraphs, comparing with existing methods of semi-supervised learning over multiple graphs. The performance advantage of the proposed approach was further confirmed by using real gene networks which were already used in the literature.

2 Related Work

Real graphs differ in reliability, by which graphs can be weighted when combined. This combination is not only in direct but also possible in a high-dimensional feature space by mapping nodes in a graph onto such a space and running a kernel-based semi-supervised learning method over the combined graph. By transforming each graph into a kernel matrix, this problem, as a classification task, has been considered extensively in the context of multiple kernel learning. Technically methods in the multiple kernel learning have used semidefinite programming (SDP) [18, 19, 20], latent maximum entropy discrimination [21] and hyperkernels [22], etc. However, they have significant drawbacks in computational complexity: 1) A kernel matrix is not necessarily sparse but rather dense, though a given graph is usually very sparse and useful for keeping low practical computation time. 2) More importantly, computationally intensive quadratic programming is included in a part or the main step of the kernel-based procedure. This makes hard to apply them to a real, large-sized graph even if an efficient algorithm such as sequential minimum optimization (SMO) is used. In contrast, estimating graph weights as well as labels of unlabeled examples can be solved by regarding this problem as a process of maximum *a posteriori* (MAP) estimation [17]. The approach in [17] is label propagation by an EM (Expectation-Maximization) algorithm in which labels of unlabeled examples and graph weights are estimated alternately. This algorithm, which hereafter we call *K-LP* standing for the Kato *et al.*'s Label Propagation approach, is very efficient and practically useful, since the amount of computation at each step is very small and usually this algorithm can be converged with only a small number of steps.

The problem of graph classification under locally informative graphs was considered by Gönen and Alpaydin who attempted to estimate node weights, by which the significance of a part of each given graph can be emphasized [23]. This algorithm, which hereafter we call LMK (Localized Multiple Kernels), estimates node weights and node labels alternately at each iteration, where a support vector machine (SVM) solver must be run. In addition, the cost function of LMK is not convex, by which LMK requires a lot of trials on initial values. Thus the algorithm of LMK suffers from a high computational burden, making hard to run this algorithm in a real situation. Furthermore, another serious drawback of this algorithm is that their framework limits the number of locally informative subgraphs per graph to only one.

3 Method

3.1 Preliminaries

Let $\mathcal{X} = \{x_1, \cdots, x_N\}$ be given N examples and $\mathcal{G}_1, \ldots, \mathcal{G}_K$ be given K graphs (or networks). That is, we have a node set $\mathcal{X}$, for which we can define different edge sets: $\mathcal{E}_1, \ldots, \mathcal{E}_K$, by which we can have K graphs: $\mathcal{G}_1, \ldots, \mathcal{G}_K$. A graph (or an edge set $\mathcal{E}$) can be defined by an $N \times N$ node similarity matrix $\boldsymbol{W}$, where the (i,j)-element $\boldsymbol{W}(i,j)$ can be given by positive edge weight $w_{i,j}$ between x_i and x_j (where $w_{i,i} = 0$):

$$\boldsymbol{W}(i,j) = \begin{cases} w_{i,j} & \text{if } (i,j) \in \mathcal{E} \\ 0 & \text{otherwise.} \end{cases}$$

Using $\boldsymbol{W}$, we can define *graph Laplacian* $\boldsymbol{L}$ as follows:

$$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W},$$

where $\boldsymbol{D}$ is a diagonal matrix with the i-th diagonal element d_i $(= \sum_j w_{i,j})$ (or diagonal elements of $\boldsymbol{D}$ can be defined by those of $\boldsymbol{W}\boldsymbol{1}_N$, where $\boldsymbol{1}_N = (1, \ldots, 1)^T$). Similarly we can define *normalized graph Laplacian* $\bar{\boldsymbol{L}}$:

$$\bar{\boldsymbol{L}} = \boldsymbol{I}_N - \boldsymbol{D}^{-\frac{1}{2}} \boldsymbol{W} \boldsymbol{D}^{-\frac{1}{2}},$$

where $\boldsymbol{I}_N$ is the $N \times N$ identity matrix. We assume that given classes are binary and only first L $(\leq N)$ examples (or nodes) are labeled and other $N - L$ examples are unlabeled, due to the nature of semi-supervised learning. That is, let $\boldsymbol{y} = (y_1, \ldots, y_L, 0, \ldots, 0)$ be labels of N examples, where $y_n \in \{+1, -1\}$ $(n = 1, \ldots, L)$. Our purpose is to estimate label score $\boldsymbol{z} = \{z_{L+1}, \ldots, z_N\}$, from which we can predict the label of the n-th unlabeled example. For example, we can place a cut-off value, and if z_n is larger than this value, $y_n = +1$; otherwise $y_n = -1$.

3.2 Soft Spectral Clustering

We here explain a soft spectral clustering method on a given graph which will be used in our proposed approach.

3.2.1 Spectral Decomposition

Given graph Laplacian $\boldsymbol{L}$, let λ_n $(n = 1, \ldots, N)$ be the eigenvalues and $\boldsymbol{v}_n$ $(n = 1, \ldots, N)$ the eigenvectors. Here let C be the number of clusters we try to obtain. A standard manner of C-way spectral clustering is that we first generate a matrix of $C-1$ eigenvectors with smallest eigenvalues [24]:

$$\boldsymbol{V} = (\boldsymbol{v}_1 \ldots \boldsymbol{v}_{C-1})$$

Let $\boldsymbol{V}_{i,j}$ be the (i,j)-element of $\boldsymbol{V}$, and $\psi(x_n)$ be a projection on x_n into the spectral space with $C-1$ dimensions:

$$\psi(x_n) = (\boldsymbol{V}_{n,1}, \ldots, \boldsymbol{V}_{n,C-1}).$$

We then use $\psi(x_n)$ to clustering given examples x_n $(n = 1, \ldots, N)$. Typically, we can run k-means over $\psi(x_n)$ $(n = 1, \ldots, N)$, but k-means is disjoint clustering, which does not suit our purpose of capture locally informative subgraphs that can be overlapped with each other.

3.2.2 Soft Clustering in Spectral Space: Estimating a Mixture of von Mises-Fisher Distributions

To implement our idea of soft clustering over the spectral space, we use a mixture of von Mises-Fisher distributions, which are used for examples distributed over a spherical surface. This distribution is suitable for our purpose, because the most informative feature in the spectral space on eigenvectors is the direction from the origin. We explain our procedure, according to [25]. Assuming that an input example is a real-valued vector $\boldsymbol{x}$, the von Mises-Fisher distribution for $\boldsymbol{x}$ can be written as follows:

$$p(\boldsymbol{x}|\boldsymbol{\mu}, \kappa) = R_h(\kappa) e^{\kappa \boldsymbol{\mu}^T \boldsymbol{x}},$$

where h is the dimension of examples, $\boldsymbol{\mu}$ is the cluster center, κ is the sample density, roughly corresponding to the inverse of variance, and $R_h(\kappa)$ is the normalization term given as follows:

$$R_h(\kappa) = \frac{\kappa^{h/2-1}}{(2\pi)^{h/2} I_{h/2-1}(\kappa)},$$

where $I_h(\kappa)$ is the modified Bessel function of the first kind of order h, which is given as follows:

$$I_h(\kappa) = \frac{1}{\pi} \int_0^{\pi} e^{\kappa \cos t} \cos(ht) dt.$$

A mixture of von Mises-Fisher distributions can then be given as follows:

$$p(\boldsymbol{x}|\boldsymbol{\alpha}, \boldsymbol{\mu}, \boldsymbol{\kappa}) = \sum_{c=1}^{C} \alpha_c R_h(\boldsymbol{\kappa}) e^{\kappa_c \boldsymbol{\mu}_c^T \boldsymbol{x}},$$

where α_c is a mixture coefficient, satisfying that $\alpha_c \geq 0$ and $\sum_{c=1}^{C} \alpha_c = 1$.

We use an EM (Expectation-Maximization) algorithm to estimate α_c and $\boldsymbol{\mu}_c$ for all c, keeping κ_c fixed for all components[1], i.e. $\kappa_c = \kappa$ for all c ($c = 1, \ldots, C$). The EM algorithm iterates the following two steps, i.e. E- and M-steps, alternately until some convergence criterion is satisfied.

(E-step)

$$f_c(\boldsymbol{x}_n|\boldsymbol{\mu}_c) \leftarrow R_h(\kappa) e^{\kappa \boldsymbol{\mu}_c^T \boldsymbol{x}_n}, \tag{1}$$

$$p(c|\boldsymbol{x}_n, \boldsymbol{\mu}_c, \boldsymbol{\alpha}) \leftarrow \frac{\alpha_c f_c(\boldsymbol{x}_n|\boldsymbol{\mu}_c)}{\sum_{c'=1}^{K} \alpha_{c'} f_{c'}(\boldsymbol{x}_n|\boldsymbol{\mu}_{c'})} \tag{2}$$

(M-step)

$$\alpha_c \leftarrow \frac{1}{N} \sum_{n=1}^{N} p(c|\boldsymbol{x}_n, \boldsymbol{\mu}_c, \boldsymbol{\alpha}), \tag{3}$$

$$\boldsymbol{\mu}_c \leftarrow \sum_{n=1}^{N} \boldsymbol{x}_n p(c|\boldsymbol{x}_n, \boldsymbol{\mu}_c, \boldsymbol{\alpha}), \tag{4}$$

$$\boldsymbol{\mu}_c \leftarrow \frac{\boldsymbol{\mu}_c}{||\boldsymbol{\mu}_c||} \tag{5}$$

For our proposed algorithm which we will describe later, we need the probability that example $\boldsymbol{x}_n$ is in cluster c, i.e. $p(c|\boldsymbol{x}_n, \boldsymbol{\mu}_c, \boldsymbol{\alpha})$. For simplicity we write $p(c|\boldsymbol{x}_n, \boldsymbol{\mu}_c, \boldsymbol{\alpha})$ by $p(c|\boldsymbol{x}_n)$ and $p(c|\boldsymbol{x}_n)$ ($c = 1, \ldots, C, n = 1, \ldots, N$) by $p(C|\mathcal{X})$. Fig. 4 is a pseudocode of our soft spectral clustering procedure using a mixture of von Mises-Fisher distributions.

[1] We fixed $\boldsymbol{\kappa}$, because a standard estimation method for $\boldsymbol{\kappa}$ gives only an approximation and in fact we empirically found that it is hard to estimate a stable value for κ.

Input : $\mathcal{X}, \boldsymbol{W}, C$
Output : $p(C|\mathcal{X})$
Soft_spectral_clustering($\mathcal{X}, \boldsymbol{W}, C$)

1: Compute normalized graph Laplacian $\bar{\boldsymbol{L}}$ from $\boldsymbol{W}$
2: Compute $\psi(x_n)$ by eigenvalues and eigenvectors of $\bar{\boldsymbol{L}}$
3: $\boldsymbol{x}_n \leftarrow \psi(x_n)$ $(n = 1, \ldots, N)$
4: **repeat**
5: **E-step**:
6: Update $f_c(\boldsymbol{x}_n|\boldsymbol{\mu}_c)$ $(c = 1, \ldots, C, n = 1, \ldots, N)$ by (1)
7: Update $p(c|\boldsymbol{x}_n)$ $(c = 1, \ldots, C, n = 1, \ldots, N)$ by (2)
8: **M-step**:
9: Update α_c $(c = 1, \ldots, C)$ by (3)
10: Update $\boldsymbol{\mu}_c$ $(c = 1, \ldots, C)$ by (4) and (5)
11: **until** some convergence criterion is satisfied.

Figure 4: Pseudocode of the soft spectral clustering algorithm.

3.3 Label Propagation

Label propagation is a time-efficient, powerful algorithm of semi-supervised learning for a graph [26]. We here review this algorithm in two cases: 1) a single graph [26] and 2) multiple graphs [17]. Here the multiple graphs case corresponds to K-LP, which will be used in our proposed approach.

3.3.1 Single Graph

For a single graph (or $K = 1$), the problem is to estimate $\boldsymbol{z}$ from given graph $\mathcal{G}$ (or $\mathcal{W}$) and label $\boldsymbol{y}$. A label propagation algorithm for this problem attempts to minimize the sum of three terms: 1) the least squares errors over labels, 2) a regularization term and 3) a term on the label consistency of neighboring nodes:

$$\begin{aligned} \hat{\boldsymbol{z}} &= \arg\min_{\boldsymbol{z}} J_{\mathrm{SG}}(\boldsymbol{z}), \\ J_{\mathrm{SG}}(\boldsymbol{z}) &= \beta_{\mathrm{Sqr}} \sum_{n=1}^{l} (z_n - y_n)^2 + \beta_{\mathrm{Bias}} \sum_{n=1}^{N} z_n^2 + \beta_{\mathrm{Net}} \sum_{i,j} \boldsymbol{W}(i,j)(z_i - z_j)^2, \end{aligned}$$

where β_{Sqr}, β_{Bias}, β_{Net} are constants for weighting corresponding three terms. This can be rewritten as follows:

$$J_{\mathrm{SG}}(\boldsymbol{z}) = \beta_{\mathrm{Sqr}}(\boldsymbol{z} - \boldsymbol{y})^T \boldsymbol{G}(\boldsymbol{z} - \boldsymbol{y}) + \boldsymbol{z}^T(\beta_{\mathrm{Bias}} \boldsymbol{I}_N + \beta_{\mathrm{Net}} \boldsymbol{L})\boldsymbol{z}, \tag{6}$$

where let $\boldsymbol{G}$ be the matrix with the following (i, j)-element $\boldsymbol{G}(i, j)$:

$$\boldsymbol{G}(i,j) = \begin{cases} 1 & \text{if } i = j \text{ and } i \leq l \\ 0 & \text{otherwise.} \end{cases}$$

The minimal label scores $\hat{\boldsymbol{z}}$ can be easily obtained by setting the first derivative of $J_{SG}(\boldsymbol{z})$ to zero ($\frac{\partial J_{SG}(\boldsymbol{z})}{\partial \boldsymbol{z}} = 0$):

$$\hat{\boldsymbol{z}} = \left(\boldsymbol{G} + \frac{\beta_{\mathrm{Bias}}}{\beta_{\mathrm{Sqr}}} \boldsymbol{I}_N + \frac{\beta_{\mathrm{Net}}}{\beta_{\mathrm{Sqr}}} \boldsymbol{L} \right)^{-1} \boldsymbol{G}\boldsymbol{y}.$$

We note that we can compute $\boldsymbol{z}$ very efficiently, since this is just the solution of a linear equation, keeping the sparseness of a given graph.

3.3.2 Multiple Graphs (K-LP)

For multiple graphs, the simplest way is to integrate given multiple graphs equally, but a more reasonable way is to consider the following weighted sum over K similarity matrices (or graphs) $\boldsymbol{W}^{(1)}, \ldots, \boldsymbol{W}^{(K)}$:

$$\boldsymbol{W}_{\mathrm{MG}} = \sum_{k=1}^{K} u_k \boldsymbol{W}^{(k)},$$

where u_k is a graph weight, i.e. a weight attached to each matrix, satisfying that $u_k \geq 0$. We write $\boldsymbol{u} = \{u_1, \ldots, u_K\}$. Note that if u_k is uniform for all k, given graphs are combined equally. The graph Laplacian of K graphs also can be introduced as follows:

$$\boldsymbol{L}_{\mathrm{MG}} = \sum_{k=1}^{K} u_k \boldsymbol{L}^{(k)},$$

where $\boldsymbol{L}^{(k)}$ is the graph Laplacian of the k-th graph.

As in Eq. (6), the problem can be then defined as follows:

$$\begin{aligned} \hat{\boldsymbol{z}} &= \arg\min_{\boldsymbol{z}} J_{\mathrm{MG}}(\boldsymbol{z}, \boldsymbol{u}) \\ J_{\mathrm{MG}}(\boldsymbol{z}, \boldsymbol{u}) &= \beta_{\mathrm{Sqr}}(\boldsymbol{z} - \boldsymbol{y})^T \boldsymbol{G}(\boldsymbol{z} - \boldsymbol{y}) + \boldsymbol{z}^T(\beta_{\mathrm{Bias}}\boldsymbol{I}_N + \beta_{\mathrm{Net}}\boldsymbol{L}_{\mathrm{MG}})\boldsymbol{z} \end{aligned} \tag{7}$$

We then have to estimate both graph weights $\boldsymbol{u}$ and label scores $\boldsymbol{z}$. For this purpose, K-LP uses an algorithm which repeats estimating the weights and the label scores alternately:

1. We set the initial value of the graph weight: $u_k = \frac{1}{k}$ for all k ($k = 1, \ldots, K$), i.e. a uniform distribution.

2. We repeat the following two steps alternately until the weights and scores are converged.

   (a) We first fix graph weight $\boldsymbol{u}$ and compute $\boldsymbol{L}_{\mathrm{MG}}$. We then update label scores $\boldsymbol{z}$ by the following updating rule, which is similar to the single graph case:

   $$\hat{\boldsymbol{z}} = \left(\boldsymbol{G} + \frac{\beta_{\mathrm{Bias}}}{\beta_{\mathrm{Sqr}}}\boldsymbol{I}_N + \frac{\beta_{\mathrm{Net}}}{\beta_{\mathrm{Sqr}}}\boldsymbol{L}_{\mathrm{MG}}\right)^{-1} \boldsymbol{G}\boldsymbol{y}. \tag{8}$$

   (b) We fix label scores $\boldsymbol{z}$ and update graph weights $\boldsymbol{u}$ by using the following updating rule:

   $$\hat{u}_k = \frac{\gamma + N}{\gamma + \beta_{\mathrm{Net}}\boldsymbol{z}^T\boldsymbol{L}^{(k)}\boldsymbol{z}}, \tag{9}$$

   where γ is a positive constant.

We note that this iterative algorithm can be also computed efficiently, because as in the preceding section, Eq. (8) shows the solution of a linear equation and Eq. (9) is also very simple. Fig. 5 shows a pseudocode of the label propagation algorithm for multiple graphs.

3.4 Proposed Approach

The reliability of a graph is not necessarily uniform over the entire graph, which leads to the idea of locally informative graphs. Furthermore locally informative subgraphs can be unevenly distributed and overlapped with each other in a graph. On the other hand, K-LP uses only a weight attached to each graph and cannot go further, meaning that K-LP cannot check the importance

Input : $\mathcal{X}, \boldsymbol{y}, \boldsymbol{W}^{(1)}, \ldots, \boldsymbol{W}^{(K)}$
Output : $\boldsymbol{z}, \boldsymbol{u}$
Label_propagation($\mathcal{X}, \boldsymbol{y}, \boldsymbol{W}^{(1)}, \ldots, \boldsymbol{W}^{(K)}$)

1: Fix β_{Sqr}, β_{Bias} and β_{Net} at arbitrary values
2: Set $\boldsymbol{u}$ at the uniform distribution, i.e. $u_k = \frac{1}{K}$ for all k
3: **repeat**
4: Compute $\boldsymbol{L}_{\mathrm{MG}}$ with current $\boldsymbol{u}$
5: Update $\boldsymbol{z}$ by using (8) and $\boldsymbol{L}_{\mathrm{MG}}$
6: Update $\boldsymbol{u}$ by using (9) and current $\boldsymbol{z}$
7: **until** some convergence criterion is satisfied.

Figure 5: Pseudocode of K-LP: the label propagation algorithm for semi-supervised learning over K graphs.

of subgraphs in given graphs. This indicates that K-LP would not be good enough in our problem setting in practice. In fact we need to attach a weight to a latent subgraph in each given graph and estimate the weight over all such latent graphs. We thus propose an approach which first captures cluster structures of each given graph, by which multiple graphs, corresponding to clusters, can be generated, and then applies the label propagation to all generated graphs. Our approach allows to find significant clusters (or subgraphs) underlying in given graphs as well as labels of unlabeled examples. We hereafter call our approach *LIG*, standing for the approach for finding Locally Informative subGraphs.

3.4.1 Extracting Cluster Structures as Graphs

We consider the latent cluster structures in each given graph. Let $\eta_{k,c}(x_n)$ be a score (or a probability) that node x_n is in cluster c of graph $\boldsymbol{W}^{(k)}$. We can then define the (i,j)-element of the graph, corresponding to cluster c of graph $\boldsymbol{W}^{(k)}$, by using $\eta_{k,c}(x_n)$:

$$\boldsymbol{W}^{(k,c)}(i,j) := \eta_{k,c}(x_i)\eta_{k,c}(x_j)\boldsymbol{W}^{(k)}(i,j)$$

That is, let $\boldsymbol{E}^{(k,c)}$ be the diagonal matrix with the n-th diagonal element of $\eta_{k,c}(x_n)$, and we can define graph $\boldsymbol{W}^{(k,c)}$:

$$\boldsymbol{W}^{(k,c)} := \boldsymbol{E}^{(k,c)}\boldsymbol{W}^{(k)}(i,j)\boldsymbol{E}^{(k,c)}$$

3.4.2 Proposed Algorithm

The idea behind LIG is to use the cluster structures obtained by soft spectral clustering for label propagation on multiple graphs. More concretely, from the results of soft spectral clustering, we can use probability $p(c|\boldsymbol{x}_n)$ (of the k-th graph) as score $\eta_{k,c}(x_n)$ in generating $\boldsymbol{W}^{(k,c)}$, allowing us to have graphs, each corresponding to a latent cluster in given graphs. We can then run K-LP over all generated $K \cdot C$ graphs. Fig. 6 shows a pseudocode of our proposed algorithm. For each given graph $\boldsymbol{W}^{(k)}$, we first run the soft spectral clustering of Fig 4 (Line 2). We then set $\eta_{k,c}(x_n)$ to $p(c|\boldsymbol{x}_n)$, which was obtained by soft spectral clustering, for all c and n (Line 3). We then compute similarity matrix $\boldsymbol{W}^{(k,c)}$of the graph, corresponding to the c-th cluster of graph $\boldsymbol{W}^{(k)}$ (Line 4). Finally we run K-LP, i.e. the label propagation algorithm of Fig. 5, over all $K \cdot C$ graphs generated in the above procedure and have label scores $\boldsymbol{z}$ and graph weights $\boldsymbol{u}$ (Line 6). We note that graph weights $\boldsymbol{u}$ are obtained over all $K \cdot C$ generated graphs.

We note that K-LP is a special case of LIG, i.e. LIG with $C = 1$.

Input : $\mathcal{X}, \boldsymbol{y}, \boldsymbol{W}^{(1)}, \ldots, \boldsymbol{W}^{(K)}, C$
Output : $\boldsymbol{z}, \boldsymbol{u}$
Proposed_algorithm$(\mathcal{X}, \boldsymbol{y}, \boldsymbol{W}^{(1)}, \ldots, \boldsymbol{W}^{(K)}, C)$
1: **for** $k = 1, \ldots, K$ **do**
2: $p(C|\mathcal{X}) \leftarrow$ Soft_spectral_clustering$(\mathcal{X}, \boldsymbol{W}^{(k)}, C)$
3: $\eta_{k,c}(x_n) = p(c|\boldsymbol{x}_n)$ $(c = 1, \ldots, C, n = 1, \ldots, N)$
4: Compute $\boldsymbol{W}^{(k,c)}$ from $\boldsymbol{E}^{(k,c)}$ and $\boldsymbol{W}^{(k)}$
5: **end for**
6: $(\boldsymbol{z}, \boldsymbol{u}) \leftarrow$ Label_propagation$(\mathcal{X}, \boldsymbol{y}, \boldsymbol{W}^{(1)}, \ldots, \boldsymbol{W}^{(K,C)})$

Figure 6: Pseudocode of LIG: our semi-supervised learning algorithm for capturing locally informative subgraphs from multiple graphs.

3.4.3 Time-Complexity

Our procedure, LIG, can be divided into three parts: spectral decomposition, soft clustering and label propagation. At first, a rough estimation of the time complexity of spectral decomposition is linear in $|\mathcal{E}|$, i.e. the number of edges of a given graph. Thus as we repeat spectral decomposition over K graphs, the time complexity of this step can be roughly $K \cdot |\mathcal{E}|$. The soft clustering and K-LP both contain iterative procedures, which make it hard to compute the total computational amount. Thus we check the computational complexity of each iterative step in the soft clustering and K-LP. Each step of the soft clustering on a graph needs $O(C \cdot N)$, and for K graphs, the complexity can be $O(K \cdot C \cdot N)$. The time complexity of each step of K-LP is $O(K \cdot (|\mathcal{E}| + N))$, and our procedure run K-LP over $K \cdot C$ graphs instead of K graphs, resulting in that the time complexity of K-LP for our case is $O(K \cdot C \cdot (|\mathcal{E}| + N))$. Thus totally the time complexity of LIG can be considered to be $O(K \cdot C \cdot (|\mathcal{E}| + N))$ for each step of K-LP. We note that practically K, the number of given graphs, is not usually large and C is set at a small number (which can be considered as a constant), meaning that the complexity is linear to the number of nodes (or edges in a graph). This indicates that LIG is time-efficient.

On the other hand, a kernel-based approach first needs $O(N^3)$ to generate a graph kernel, such as diffusion kernel [27, 28], from a given graph in a standard manner[2] [8]. Kernel-based approaches then use quadratic programming in most cases, which is processed by a solver, usually with a standard mathematical software, and then the complexity must be high but unclear. Totally the amount of computation time of a kernel-based approach must be much larger than that of LIG, and this generates a sizable difference for large graphs in practical applications.

4 Experimental Results

4.1 Procedure

We first used both synthetic and real data to evaluate the performance of LIG, setting at $C = 3$ and comparing with three other methods: 1) K-LP [17], corresponding to LIG of $C = 1$, 2) an efficient semi-supervised learning method for multiple graphs [8]. This method avoids SDP-based SVM. We call this method TSS, and 3) LMK [23], which is for multiple graphs by using SVM. Once again we here briefly review LMK. LMK has multiple kernel functions, each corresponding to one graph, and each kernel function has a weight, making the kernel function localized and

[2]In addition, a kernel matrix is usually dense (even though a given graph is sparse), which heavily affects the space-complexity.

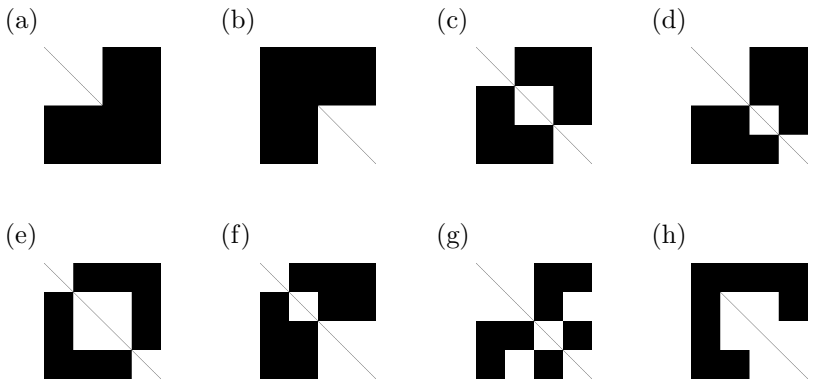

Figure 7: Templates to generate synthetic data (Data 1: (a) and (b), Data 2: (c), Data 3: (d) and (e), Data 4: (d), (e) and (f), Data 5: (g) and (h) and Data 6: (e) and (h)).

being used when combining multiple graphs. Weights correspond to gating functions, each having the central vector and being proportional to the inner product between the feature vector of each node and the center vector. In [23], weights are optimized by alternately updating gate functions with a gradient method and running SVM with SMO. In our experiments, for each node, we used the edge connectivity as the feature vector. Weight optimization was in the same manner as in [23]. Note that LMK needs a heavy computational load, by which we were unable to run LMK for real data and synthetic datasets with 1,000 nodes.

We then checked the time-efficiency of LIG, comparing with the competing methods, by increasing the size of synthetic datasets in a general setting.

Throughout all experiments in this paper, we used $\kappa = 2$ for soft spectral clustering, and $\beta_{\mathrm{Sqr}} = N, \beta_{\mathrm{Bias}} = \frac{1}{N}, \beta_{\mathrm{Net}} = 1$ and $\gamma = 1$ for label propagation[3]. This setting was also applied to K-LP. We optimized regularization parameters of TSS and LMK in an appropriate range for each case. All experiments were run on an IBM x3455 server with Dual-Core AMD Opteron(TM) Processor 2222SE 2.9 GHz and main memory of 48GB for our experiments.

4.2 Performance on Synthetic Data

4.2.1 Parameters for Data Generation

We have four parameters in generating synthetic data, for three inputs, i.e. $\mathcal{X}$ (examples), $\boldsymbol{y}$ (labels) and $\boldsymbol{W}$ (graphs):

1. $\mathcal{X}$: The number of examples, i.e. N, is the parameter, which was fixed at 300 first and then 1,000. LMK is time-consuming, by which we had to skip showing the result of LMK over the graph of 1,000 nodes.

2. $\boldsymbol{y}$: Out of all N examples, we randomly chose examples to be labeled by either of two: +1 or -1. The ratio of labeled examples to all examples, i.e. L/N, is the parameter for $\boldsymbol{y}$. We varied this value in our experiments.

3. $\boldsymbol{W}$: We randomly generated edges of each input graph, i.e. $\boldsymbol{W}$, according to a *template*. Each template is a $N \times N$ binary matrix, i.e. a graph of N nodes $(x_1, \ldots, x_N)$, specifying whether each of N nodes, i.e. examples, can connect to another of N nodes. That is, given

[3]Another item of note is that we used normalized graph Laplacian for soft spectral clustering and graph Laplacian for label propagation, as shown in the Method section.

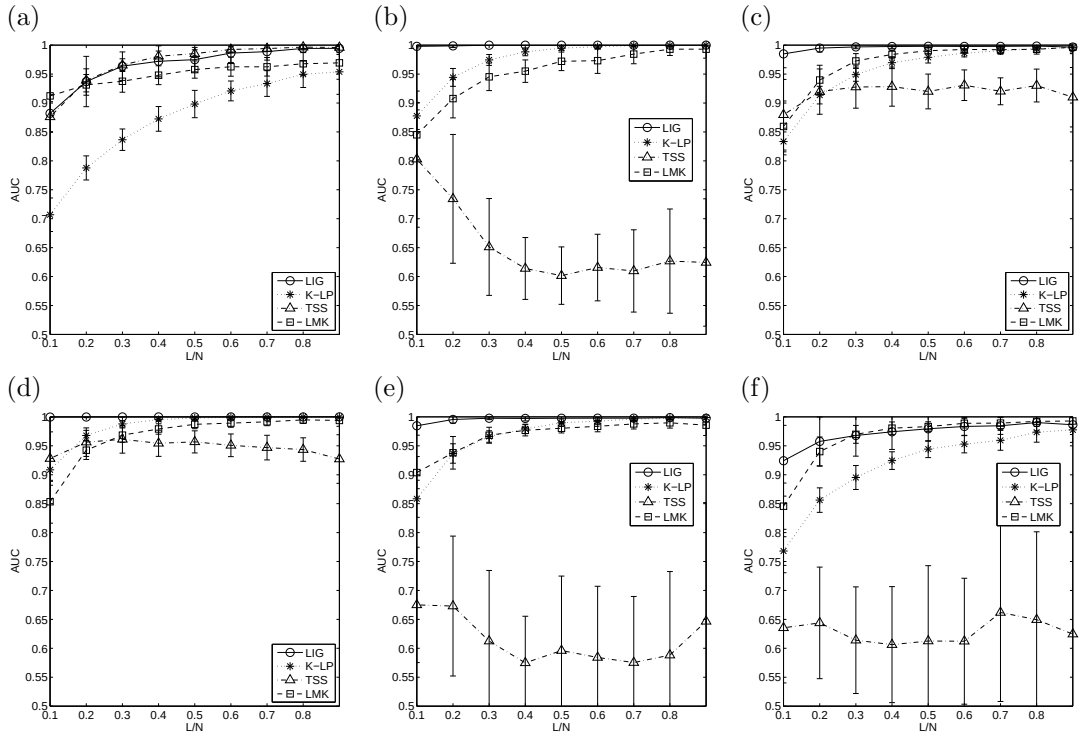

Figure 8: AUC (and standard deviation) of LIG and K-LP at $N = 300$ and $d_{out} = 0.03$ when varying L/N in Data (a) 1, (b) 2, (c) 3, (d) 4, (e) 5 and (f) 6.

a template, if an element is white colored, the corresponding two nodes can be connected; otherwise the two nodes cannot. Fig. 7 shows a list of templates which we used in our experiments. For example, (c) is a template with three white areas (or clusters), meaning that nodes can be connected with each other within each of the three areas. That is, $x_1, \ldots, x_{\frac{N}{3}}$ can be connected with each other, which is shown in the upper-left one, $x_{\frac{N}{3}+1}, \ldots, x_{\frac{2N}{3}}$ can be also connected with each other, shown in the center, and $x_{\frac{2N}{3}+1}, \ldots, x_N$ can be connected with each other, which is shown in the the lower-right one. Our random edge generation has two parameters: d_{IN}, which is the ratio of linked edges to all possible edges in clusters (the white areas in templates), and d_{OUT}, which is the ratio of linked edges to all possible edges which are not in clusters (the black areas in templates). We note that as d_{IN} is increased more, the clusters can be more easily captured[4]. On the other hand, d_{OUT} is like a noise ratio, meaning that capturing clusters is harder with increasing d_{OUT}. We fixed d_{IN} at 0.1 and changed the value of d_{OUT}.

We then have totally two controllable parameters: L/N and d_{OUT}.

4.2.2 Evaluation Procedure

To evaluate the classification accuracy of each method, we computed AUC (Area under the ROC curve) [29, 30] by using the predicted scores over unlabeled examples. That is, for each parameter

[4]We note that clusters are assigned independent of labels, meaning that clusters are not necessarily consistent with labels.

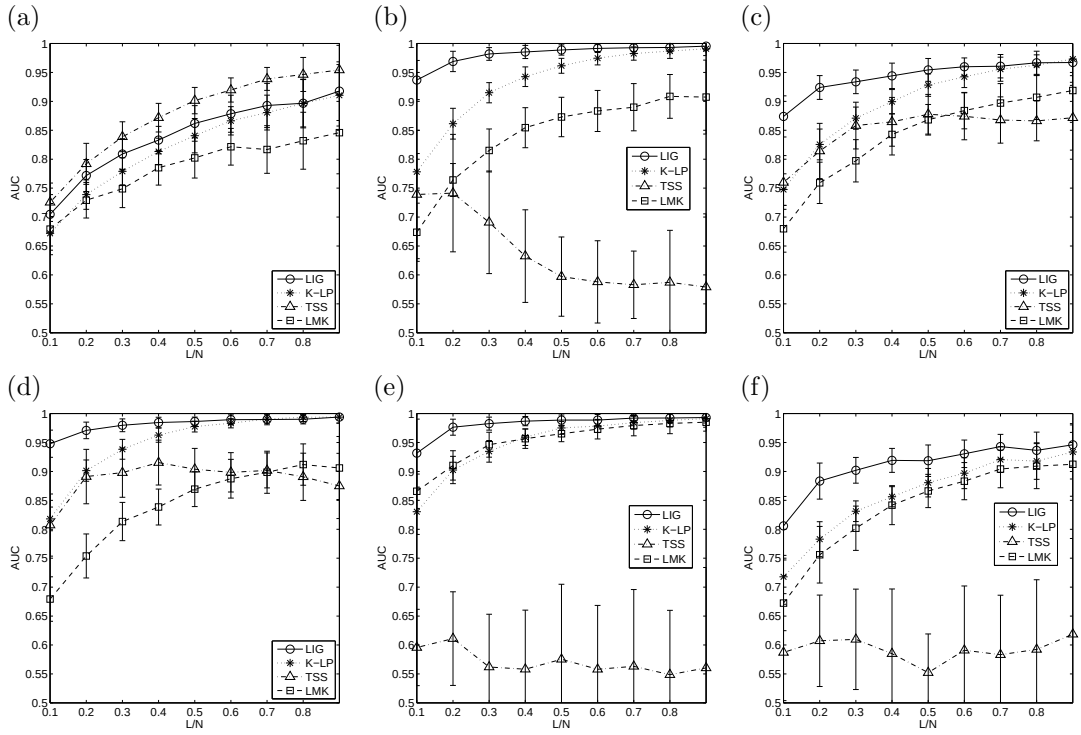

Figure 9: AUC (and standard deviation) of LIG and K-LP at $N = 300$ and $d_{out} = 0.05$ when varying L/N in Data (a) 1, (b) 2, (c) 3, (d) 4, (e) 5 and (f) 6.

setting, we randomly generated datasets fifty times and computed the average AUC (and the standard deviation) over fifty runs from predicted label scores for unlabeled examples.

4.2.3 Results

Data 1: We started with a simple case. True labels were set at $+1$ for $x_1, \ldots, x_{\frac{N}{2}}$ and -1 for $x_{\frac{N}{2}+1}, \ldots, x_N$. We generated $\boldsymbol{W}^{(1)}$ and $\boldsymbol{W}^{(2)}$ from the two templates, shown in Fig. 7 (a) and (b), respectively. As described in the above, we randomly generated edges according to d_{IN} in the white areas and d_{OUT} in the black areas. Simply then the edges of $\boldsymbol{W}^{(1)}$ drawn according to d_{IN} were those between examples with the label of $+1$, and the edges of $\boldsymbol{W}^{(2)}$ drawn according to d_{IN} were those between examples with the label of -1. Thus, even if we combine these two graphs equally, the labels should be predicted well, meaning that K-LP will work. However, Figs. 8 (a), 9 (a), 10 (a) and 11 (a), which are the AUC of competing methods at $N = 300$ for various L/N and d_{OUT} shows that LIG clearly outperformed K-LP and LMK in all cases, while TSS achieved a good performance, being slightly better than LIG in particularly higher d_{OUT}. Figs. 14 (a), 15 (a), 16 (a) and 17 (a) show the AUC of competing methods at $N = 1,000$ for various L/N and d_{OUT}. The feature observed by the case of $N = 300$ was kept for $N = 1,000$, where however the performance difference among the three competing methods, LIG, K-LP and TSS was slight, showing that all these methods achieved a high performance.

Data 2: Data 2 is a little more complicated than Data 1. True labels are set at $+1$ for $x_1, \ldots, x_{\frac{N}{3}}$, -1 for $x_{\frac{N}{3}+1}, \ldots, x_{\frac{2N}{3}}$ and $+1$ for $x_{\frac{2N}{3}+1}, \ldots, x_N$. We used Fig. 7 (c) for the template of Data 2. We randomly generated $\boldsymbol{W}^{(1)}$ and $\boldsymbol{W}^{(2)}$ from this template, but $\boldsymbol{W}^{(3)}$ was randomly

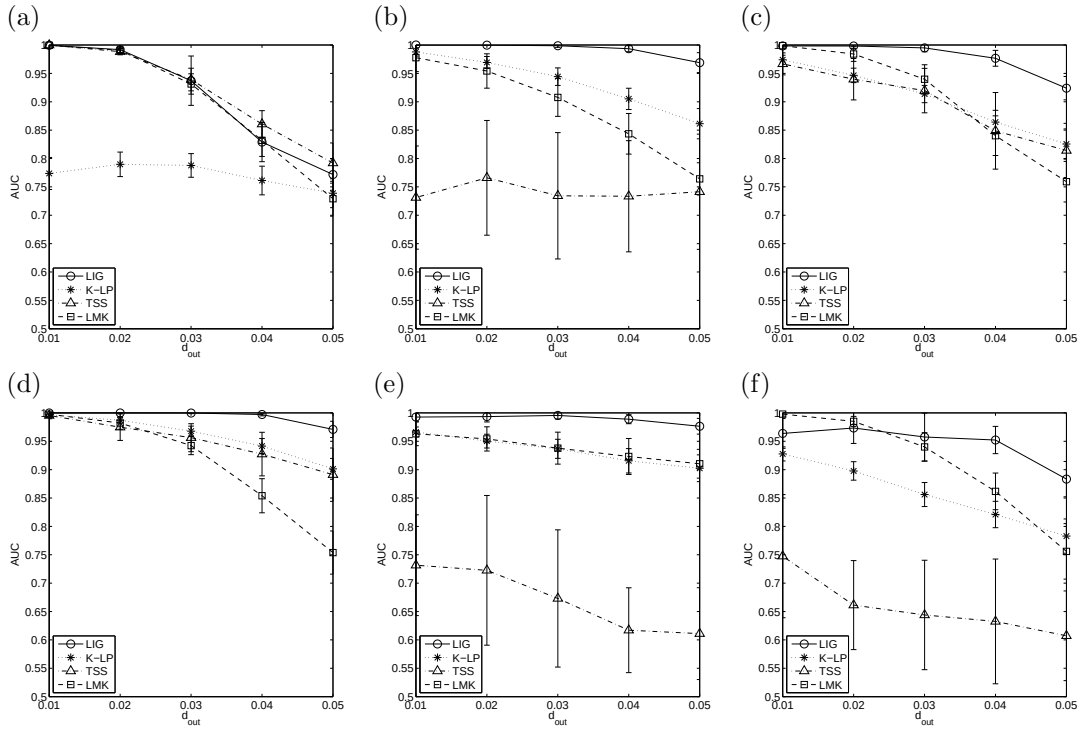

Figure 10: AUC (and standard deviation) of LIG and K-LP at $N = 300$ and $L/N = 0.2$ when varying d_{OUT} in Data (a) 1, (b) 2, (c) 3, (d) 4, (e) 5 and (f) 6.

generated without using any template. Thus, this case, K-LP has to combine these three graphs with more weights on $\boldsymbol{W}^{(1)}$ and $\boldsymbol{W}^{(2)}$, but it would be harder to do so with increasing noise more as well as with decreasing labeled examples. Figs. 8 (b), 9 (b), 10 (b) and 11 (b) show the AUC of four competing methods on Data 2 at $N = 300$ when L/N and d_{OUT} are changed. This time, we can see that LIG clearly outperformed three other methods in all cases, particularly for more noisy cases (d_{OUT} is larger) and a smaller number of labels (L/N is smaller), as we expected. Note that the performance of TSS is significantly worse than other three methods, probably because TSS could not adopt to a more complex dataset like Data 2. This indicates that LIG is more useful in real situations. Fig. 12 shows one example of a result, where the three graphs which are weighted most by LIG (where more weighted edges are colored brighter in each graph). This figure shows that the brightest part in each of the three graphs corresponded to a different cluster in Fig. 7 (c) exactly, meaning that LIG captured each of these three clusters, i.e. locally informative subgraphs. Figs. 14 (b), 15 (b), 16 (b) and 17 (b) show the AUC of LIG, K-LP and TSS, at $N = 1,000$ for various L/N and d_{OUT}. The results of these figures kept the feature obtained by the case of $N = 300$. Again, LIG clearly outperformed K-LP and TSS for the data with larger amount of noise or with a smaller number of labels, where the performance of TSS is far worse than both LIG and K-LP.

Data 3: Data 3 is more complex. True labels were set at $+1$ for $x_1, \ldots, x_{\frac{N}{2}}$ and -1 for $x_{\frac{N}{2}+1}, \ldots, x_N$. We generated two graphs, $\boldsymbol{W}^{(1)}$ and $\boldsymbol{W}^{(2)}$ from the two templates, shown in Fig. 7 (d) and (e), respectively. Fig. 7 (e) had a large cluster, which contained both examples labeled by $+1$ and those by -1, disturbing correct classification on the examples covered by this cluster. Thus, K-LP should weight $\boldsymbol{W}^{(1)}$ more, but it would be more difficult with increasing

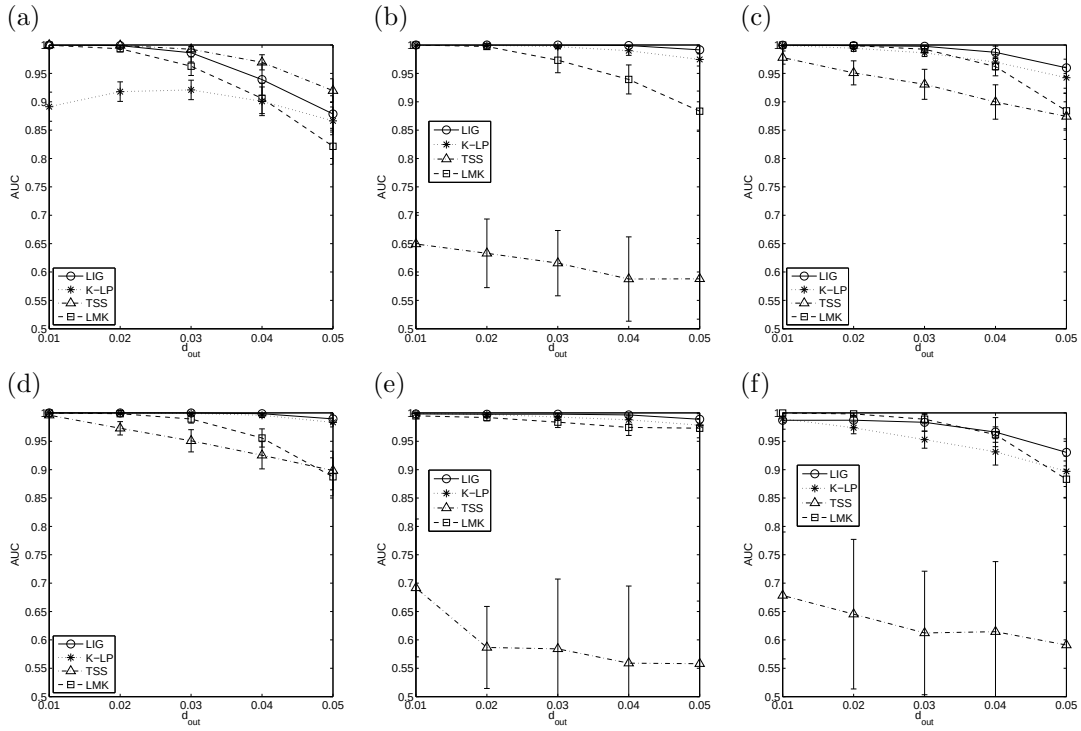

Figure 11: AUC (and standard deviation) of LIG and K-LP at $N = 300$ and $L/N = 0.6$ when varying d_{OUT} in Data (a) 1, (b) 2, (c) 3, (d) 4, (e) 5 and (f) 6.

noise and decreasing labeled examples. Figs. 8 (c), 9 (c), 10 (c) and 11 (c) show the AUC of four competing methods on Data 3 at $N = 300$ when L/N and d_{OUT} are changed. These results clearly show that LIG outperformed other three methods, pronouncing the advantage of LIG for fewer labels and a larger amount of noise, confirming the effectiveness of our approach in a more real situation. Fig. 13 shows the top three most weighted graphs by LIG, as in Fig. 12. This figure shows that the brightest parts shown in these graphs were all correct in classification, and the large noisy cluster was not in the top three. Figs. 14 (c), 15 (c), 16 (c) and 17 (c) show the AUC of LIG, K-LP and TSS, at $N = 1,000$ for various L/N and d_{OUT}. The results had almost a similar feature obtained by the case of $N = 300$, confirming the advantage of LIG over other methods in more real cases.

Data 4: Data 4 is similar to Data 3. True labels were set at $+1$ for $x_1, \ldots, x_{\frac{N}{2}}$ and -1 for $x_{\frac{N}{2}+1}, \ldots, x_N$. We generated three graphs $\boldsymbol{W}^{(1)}$, $\boldsymbol{W}^{(2)}$ and $\boldsymbol{W}^{(3)}$ from the three templates, shown in Fig. 7 (d), (e) and (f) respectively. This case, Fig. 7 (f) was the reverse of Fig. 7 (d), and this means that the difficulty of this data was also from the large cluster of Fig. 7 (e). Figs. 8 (d), 9 (d), 10 (d) and 11 (d) show the AUC of four competing methods on Data 4 at $N = 300$ when L/N and d_{OUT} are changed. Similarly Figs. 14 (d), 15 (d), 16 (d) and 17 (d) show the AUC at $N = 1,000$ for LIG, K-LP and TSS. These results confirmed the performance advantage of LIG over other methods again, particularly emphasizing on a smaller number of labeled examples and a larger number of noisy edges. Thus this data also revealed the effectiveness of LIG.

Data 5: True labels were set at $+1$ for $x_1, \ldots, x_{\frac{N}{2}}$ and -1 for $x_{\frac{N}{2}+1}, \ldots, x_N$. We generated two graphs $\boldsymbol{W}^{(1)}$ and $\boldsymbol{W}^{(2)}$ from the two templates, shown in Fig. 7 (g) and (h), respectively. These graphs had some clusters overlapped between two different labels, such as the center part

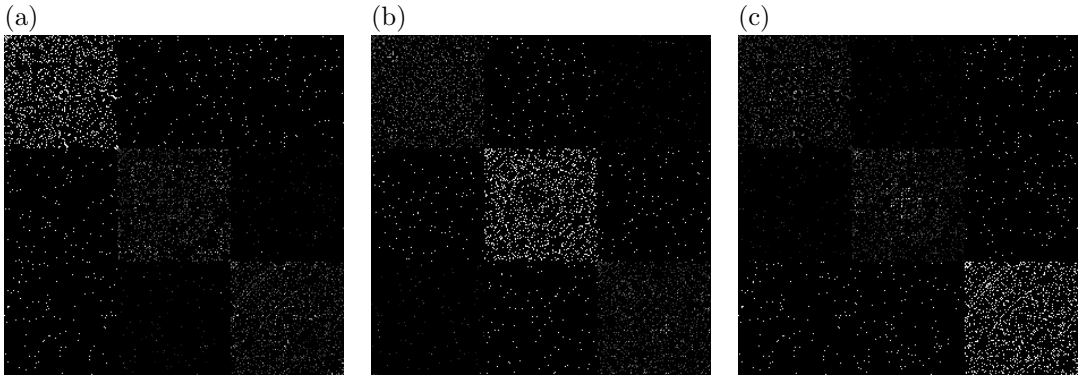

Figure 12: Top three most weighted graphs (where brighter edges are weighted more) in Data 2 by LIG.

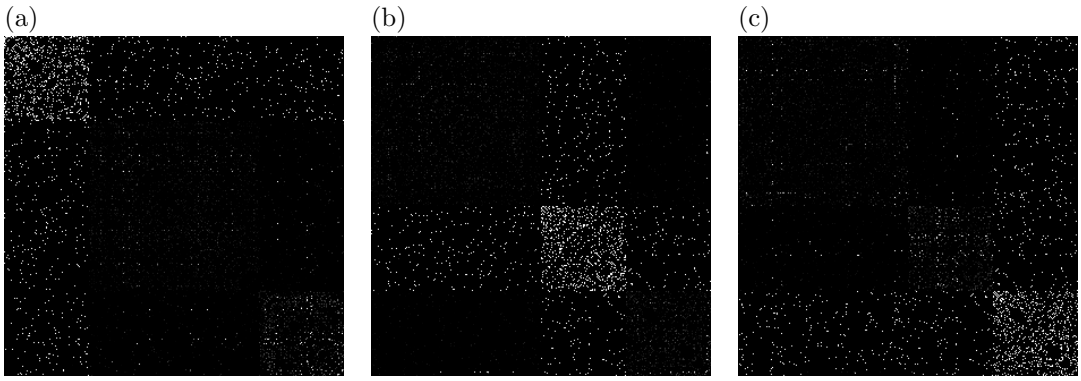

Figure 13: Top three most weighted graphs (where brighter edges are weighted more) in Data 3 by LIG.

of Fig. 7 (h), implying that this was the most difficult case to learn in the six datasets of this experiment. Figs. 8 (e), 9 (e), 10 (e) and 11 (e) show the AUC of four competing methods on Data 5 at $N = 300$, changing L/N and d_{OUT}. Similarly Figs. 14 (e), 15 (e), 16 (e) and 17 (e) show the AUC at $N = 1,000$ for LIG, K-LP and TSS. These results showed the performance advantage of LIG over other methods, being clearer than the previous synthetic datasets particularly for both fewer labels and higher noise ratios, meaning that LIG was more powerful than other methods for this type of harder case, which might be close to real data.

Data 6: True labels were set at $+1$ for $x_1, \ldots, x_{\frac{N}{2}}$ and -1 for $x_{\frac{N}{2}+1}, \ldots, x_N$. We then generated two graphs $\boldsymbol{W}^{(1)}$ and $\boldsymbol{W}^{(2)}$ from the two templates, shown in Fig. 7 (e) and (g), respectively. This was also a hard case to classify examples, because of the clusters covering examples with different labels. Figs. 8 (f), 9 (f), 10 (f) and 11 (f) show the AUC on this data at $N = 300$, and Figs. 14 (f), 15 (f), 16 (f) and 17 (f) show that at $N = 1,000$. Clearly this result also confirmed the performance improvement of LIG over other methods, particularly for the case closer to a real, noisy situation.

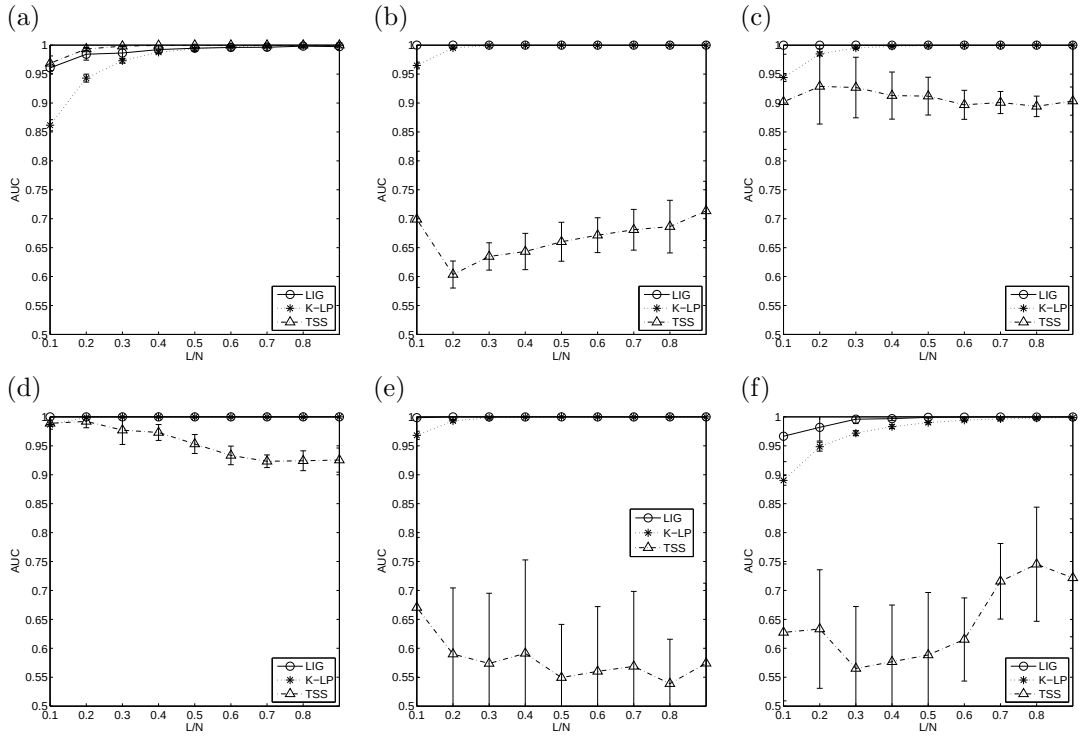

Figure 14: AUC (and standard deviation) of LIG and K-LP at $N = 1,000$ and $d_{out} = 0.03$ when varying L/N in Data (a) 1, (b) 2, (c) 3, (d) 4, (e) 5 and (f) 6.

Table 1: # of genes in each of thirteen classes.

1	2	3	4	5	6	7	8	9	10	11	12	13
1048	242	600	753	335	578	479	264	193	411	192	306	81

4.3 Performance on Real Data

4.3.1 Experimental Procedure for Real Gene Function Prediction

We checked the performance of LIG on the problem of gene function prediction by using a real dataset in [8]. This dataset has 3,558 genes, i.e. $N = 3558$, and five graphs (or gene networks), which share the same 3,558 genes (nodes) and have different edges from each other. These five graphs are 1) a protein domain network, 2) a protein complex network, 3) a protein-protein (physical) interactions network, 4) a genetic interactions network and 5) a gene expression network. Thirteen types of labels (which correspond to the most major functional categories, such as metabolism, sorted by CYGD (Comprehensive Yeast Genome Database) [31]) are given, and a gene can be multi-labeled. Table 1 shows the number of genes in each of the thirteen classes. For each label type, we set a binary classification problem, meaning that we generated thirteen different datasets (and binary classifiers). We first conducted 5×5-fold cross-validation on each of these datasets and the performance was evaluated by AUC. That is, all nodes were divided into five folds, one being unlabeled (for testing) and the other four being labeled (for training), and labels of unlabeled nodes were predicted by using labeled nodes. This was done over all five folds,

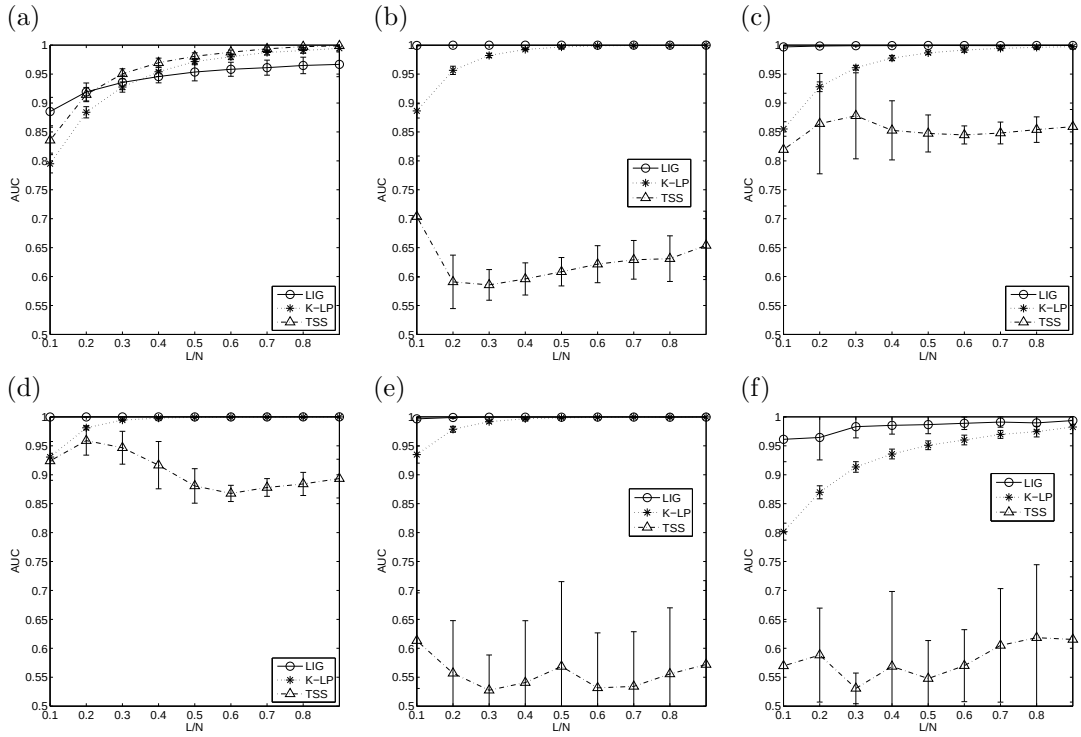

Figure 15: AUC (and standard deviation) of LIG and K-LP at $N = 1,000$ and $d_{out} = 0.05$ when varying L/N in Data (a) 1, (b) 2, (c) 3, (d) 4, (e) 5 and (f) 6.

changing the unlabeled fold, and this division was repeated five times totally. We then checked experimental results, i.e. obtained graph components over the graphs.

4.3.2 Results

Table 2 shows the resultant AUC of LIG, K-LP (C=1) and TSS, where the best AUC is bold-faced for each class. This table shows that LIG achieved the best AUC in eight classes, while TSS achieved that in five classes. One notable feature of this result was that the performance improvement by LIG over K-LP and TSS was clearer for a smaller set of genes. For example, LIG clearly outperformed K-LP and TSS for the 13th class, i.e. the smallest class. This result might be natural, because a smaller set of genes is expected to be related with a particular part (or subgraph) in given gene networks more strongly. Another point is that it might be looked that the difference in AUC between LIG and TSS was very small, but this is not. In fact, in the five classes where TSS achieved the best, you can see the difference of AUC between TSS and the best of LIG was only 1.2 % at maximum. On the other hand, LIG outperformed TSS significantly in most of the eight cases. For example, in the 13th class, LIG achieved AUC of 82% when C=5, while that of TSS was just 70.5%. Table 3 shows the average AUC over 13 classes, indicating this point clearly. This table shows that the average AUC by LIG was 83.5 to 83.8% (Note that $C = 1$ is K-LP) while that by TSS was only 80.9%. Furthermore, we can see that the performance was almost saturated at C of five already, indicating that it was sufficient to set C at five. This result implies that for these networks, the most major five or less clusters were important in the given gene networks to classify genes in terms of their biological functions.

We then checked the graphs (or clusters) obtained by LIG. Since all 3,558 genes were too large

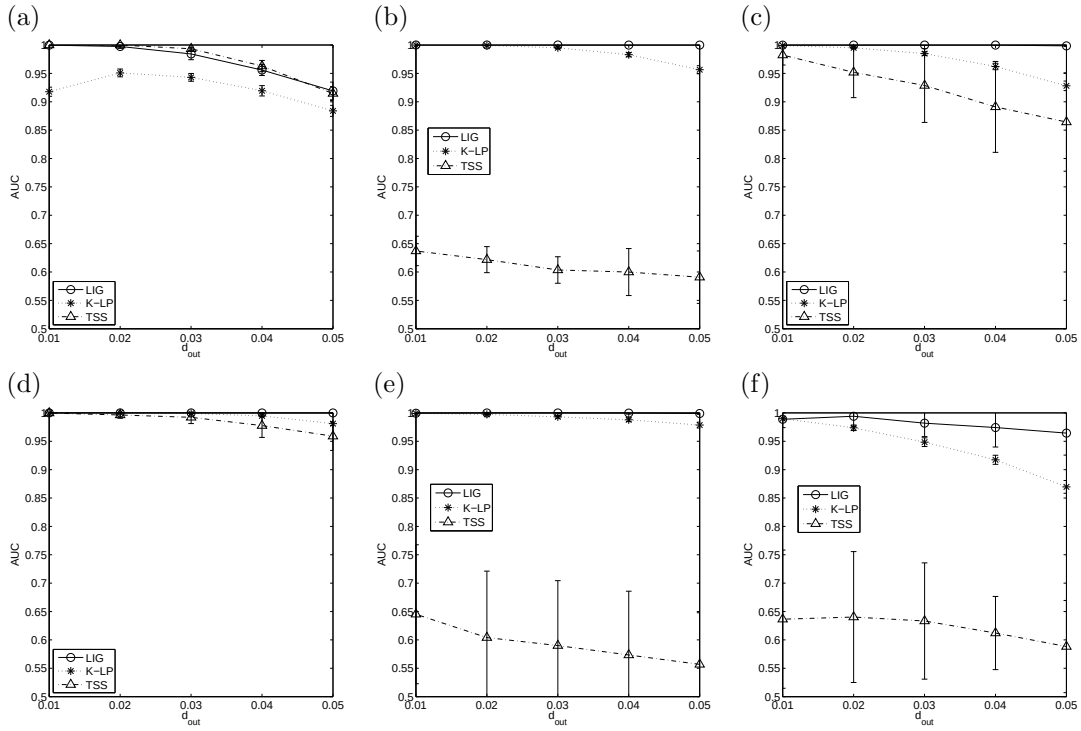

Figure 16: AUC (and standard deviation) of LIG and K-LP at $N = 1,000$ and $L/N = 0.2$ when varying d_{OUT} in Data (a) 1, (b) 2, (c) 3, (d) 4, (e) 5 and (f) 6.

to show in a single graph, we randomly chose 300 genes. Fig. 18 shows the gene networks by these 300 genes with clusters circled by red, for which node weights (η) were high[5]. These graphs were only a part of all 3,558 genes, and so they were very sparse in some cases and contained a lot of unconnected nodes. We first note that red-circled clusters captured by LIG for a network were basically different from those for another network. For example, the nodes of the most dense subgraph in (a) were not connected at all in any of other networks. That is, this cluster is peculiar to (a), i.e. the protein domain network. If all subgraphs were like this type of independent one, K-LP might work well enough. However, on the other hand, some clusters (or subgraphs) were shared by more than one gene networks. For example, the second dense subgraph circled by red in (a) was also a subgraph in (c), and the subgraph colored red in (d) was also found in (c) and (e). This case, it might be not so easy to emphasize (or attach a large weight to) only the network with this type of subgraph by K-LP (or another existing graph-based semi-supervised learning approach), because all graphs with this subgraph might be also emphasized and they might have subgraphs which are irrelevant to classification. However, LIG allows to capture latent clusters like them before running the label propagation, and the above problem can be avoided. This must be a clear advantage of LIG.

4.4 Real Execution Time

Finally Fig. 19 shows the average execution time of LIG and other competing methods over three runs[6] with increasing the number of examples (nodes) of Data 2 in synthetic data, with $L/N = 0.5$,

[5]We removed nodes which were unconnected throughout all five networks.

[6]The computation time was kept almost the same for all three runs.

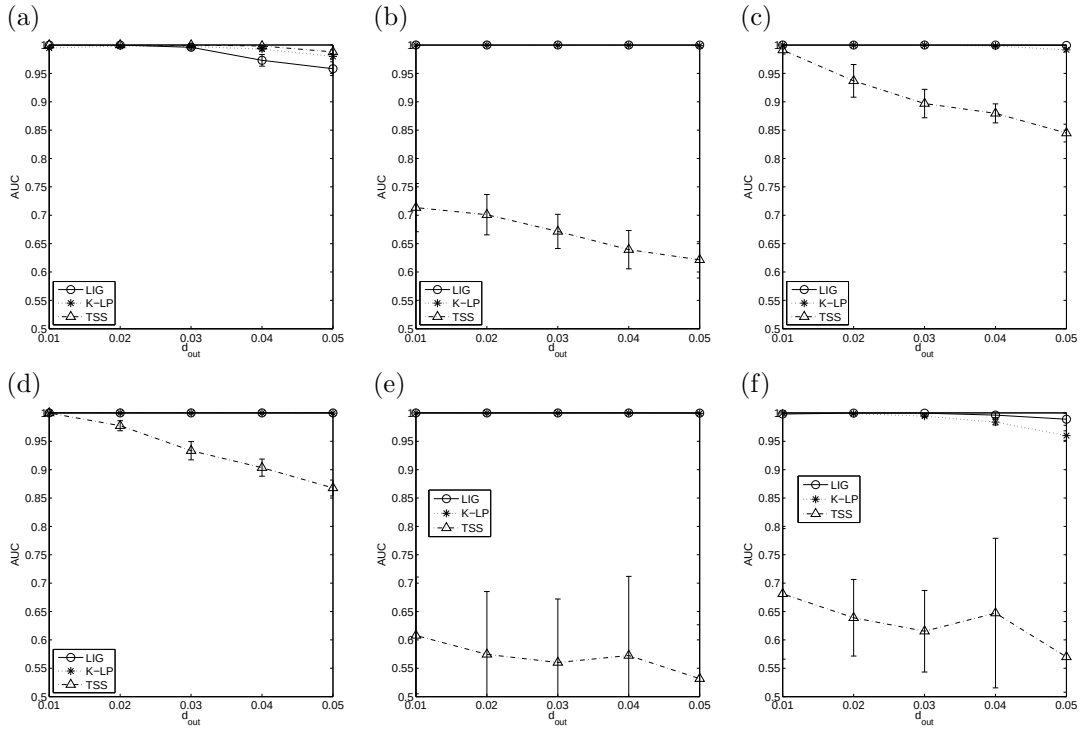

Figure 17: AUC (and standard deviation) of LIG and K-LP at $N = 1,000$ and $L/N = 0.6$ when varying d_{OUT} in Data (a) 1, (b) 2, (c) 3, (d) 4, (e) 5 and (f) 6.

$d_{\mathrm{IN}} = 0.1$ and $d_{\mathrm{OUT}} = 0.025$. We note that the computation time of LMK is just by one trial. This is a log-log plot, showing the linearity of the computation time of LIG with respect to the example (node) size. Precisely LIG is C times slower than K-LP, because the input of the label propagation part of LIG is $K \cdot C$ graphs instead of K graphs in K-LP. However this figure clearly shows that this point did not matter for a larger number of examples, say the case of 10^4 examples (nodes). In fact, this figure indicates that LIG with C of any size, TSS and K-LP needed almost the same amount of computation time for the dataset with 10^4 examples. On the other hand, we note that the computation time of LMK is far larger than that of LIG, TSS and K-LP. This means that the computational cost of LMK is more serious, because this time is just from only one trial, while LMK needs a number of trials with different initial values. By LIG, it was only around 10^3 seconds (roughly 15 minutes) to combine three graphs with 10,000 nodes. We stress that we obtained a similar result on real computation time for each of other synthetic datasets.

5 Concluding Remarks

We have proposed a time-efficient and powerful approach for semi-supervised learning using multiple graphs. Our approach is a reasonable combination of soft spectral clustering with time-efficient label propagation, which allows to consider latent cluster structures in each graph directly. An interesting point is that if we use some large number for the number of clusters, soft spectral clustering will capture all informative clusters, which can be ordered by using eigenvalues and will be further weighted by label propagation. This implies that our approach is released from the problem of deciding the optimal number of clusters. We have demonstrated the effectiveness and

Table 2: AUC (%) of LIG, with changing C (C=1 corresponding to K-LP), and TSS

Class	1	2	3	4	5	6	7	8	9	10	11	12	13
LIG													
C= 1	84.4	83.1	76.6	82.6	87.3	78.6	85.0	74.2	77.4	70.8	56.2	91.9	64.9
C= 5	85.4	84.6	**83.9**	86.2	91.3	83.7	**88.1**	79.0	83.3	79.5	64.3	93.7	**82.0**
C=10	85.2	85.4	83.7	87.2	91.5	84.2	87.6	80.0	**83.8**	**79.6**	63.9	93.6	81.8
C=15	85.8	85.1	83.4	87.3	91.8	83.7	87.9	80.1	**83.8**	78.9	64.4	94.2	80.9
C=20	85.8	85.2	83.8	87.1	**92.0**	84.1	88.0	**80.6**	**83.8**	79.1	**64.6**	94.1	81.7
TSS	**86.0**	**85.7**	81.5	**88.3**	89.3	**85.4**	85.1	75.3	77.0	75.8	57.7	**94.7**	70.5

Table 3: Average AUC (%) of LIG, changing C (C=1 corresponding to K-LP) and TSS

C	1	5	10	15	20	TSS
AUC	77.9	83.5	83.6	83.6	83.8	80.9

efficiency of our approach by using both synthetic and real datasets. Experimental results clearly revealed the significant performance achievements, the time-efficiency over a large network and the output comprehensibility of our method.

Locally informative subgraphs are highly connected nodes, which are consistent with label information. This means that LIG is advantageous as long as edge connectivity is consistent with label information, while it would be hard for LIG to work under the following two situations: 1) labels are inconsistent with graph structures and 2) graphs are not structured well in terms of clusters, such as highly dense graphs.

In Section 4.3, we have thirteen classes, for which we set thirteen binary classification problems. One gene can have multiple functions, implying that this setting is reasonable, while it would be more efficient if we can assign multiple labels for each node at once. Thus one possible future work would be to develop a method which allows to deal with multi-class problems directly. It would be also interesting to build a method which can assign a score to a node showing whether it belongs to a class and can further rank nodes by using the attached scores. This would be useful in real settings where labels are assigned by performing experiments with high costs, such as those in molecular biology, because we can design experiments to be conducted in an order, according to the predicted scores. Another possibility is to consider using given labels in the first (clustering) step of our approach, by which the performance of the final label propagation might be improved.

Acknowledgment

M. S. and H. M. have been supported in part by BIRD, JST of Japan, and M. S. has been supported in part by KAKENHI, MEXT of Japan, 20700269.

References

[1] J. Weston, C. Leslie, E. Ie, D. Zhou, A. Elisseeff, W. S. Noble, Semi-supervised protein classification using cluster kernels, Bioinformatics 21 (15) (2005) 3241–7.

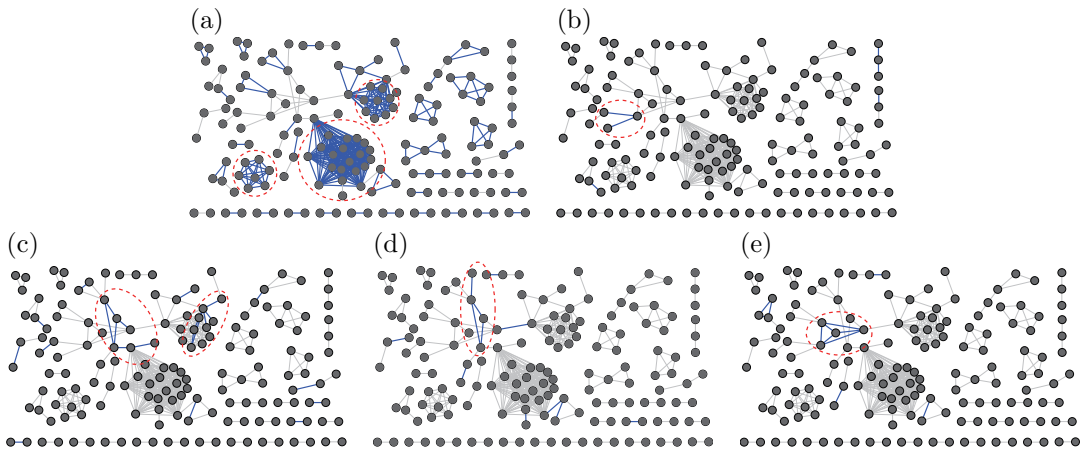

Figure 18: Randomly chosen 300 genes out of all 3,558 genes, with captured clusters colored red: (a) protein domain network, (b) protein complex network, (c) protein-protein interactions network, (d) genetic interactions network and (e) gene expression network.

[2] A. Fujino, N. Ueda, K. Saito, Semisupervised learning for a hybrid generative/discriminative classifier based on the maximum entropy principle, IEEE Transactions on Pattern Analysis and Machine Intelligence 30 (3) (2008) 424–437.

[3] B.-K. Bao, B. Nia, Y. Mua, S. Yana, Efficient region-aware large graph construction towards scalable multi-label propagation, Pattern Recognition (2010) (in press).

[4] O. Chapelle, B. Schölkopf, A. Zien (Eds.), Introduction to Semi-Supervised Learning, MIT Press, Cambridge, MA, 2006, Ch. 1, pp. 1–12.

[5] B. Kulis, S. Basu, I. S. Dhillon, R. J. Mooney, Semi-supervised graph clustering: a kernel approach, Machine Learning 74 (1) (2009) 1–22.

[6] X. Yin, S. Chen, E. Hu, D. Zhang, Semi-supervised clustering with metric learning: An adaptive kernel method, Pattern Recognition 43 (4) (2010) 1320–1333.

[7] I. Lee, S. Date, A. Adai, E. Marcotte, A probabilistic functional network of yeast genes, Science 306 (5701) (2004) 1555.

[8] K. Tsuda, H. Shin, B. Schölkopf, Fast protein classification with multiple networks, Bioinformatics 21 (2) (2005) 59–65.

[9] M. Shiga, I. Takigawa, H. Mamitsuka, Annotating gene function by combining expression data with a modular gene network, Bioinformatics 23 (13) (2007) i468–i478.

[10] U. Karaoz, T. Murali, S. Letovsky, Y. Zheng, C. Ding, C. Cantor, S. Kasif, Whole-genome annotation by using evidence integration in functional-linkage networks, Proceedings of the National Academy of Sciences 101 (9) (2004) 2888.

[11] K. Maciag, S. J. Altschuler, M. D. Slack, N. J. Krogan, A. Emili, J. F. Greenblatt, T. Maniatis, L. F. Wu, Systems-level analyses identify extensive coupling among gene expression machines, Mol Syst Biol 2 (2006) 2006.0003.

[12] R. Sharan, I. Ulitsky, R. Shamir, Network-based prediction of protein function, Molecular Systems Biology 3 (88) (2007) Epub.

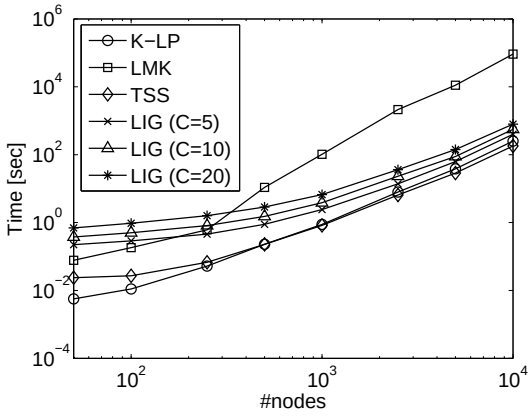

Figure 19: Real computation time with changing the example size of Data 2 in synthetic datasets, with $L/N = 0.5$, $d_{\mathrm{IN}} = 0.1$ and $d_{\mathrm{OUT}} = 0.025$.

[13] S. Mostafavi, D. Ray, D. Warde-Farley, C. Grouios, Q. Morris, GeneMANIA: a real-time multiple association network integration algorithm for predicting gene function, Genome Biology 9 (Suppl 1) (2008) S4.

[14] W. Xi, B. Zhang, Z. Chen, Y. Lu, S. Yan, W.-Y. Ma, E. A. Fox, Link fusion: a unified link analysis framework for multi-type interrelated data objects, in: Proceedings of the 13th International Conference on World Wide Web, ACM, New York, NY, USA, 2004, pp. 319–327.

[15] T. Zhang, A. Popescul, B. Dom, Linear prediction models with graph regularization for web-page categorization, in: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, USA, 2006, pp. 821–826.

[16] X. Ling, W. Dai, G.-R. Xue, Q. Yang, Y. Yu, Spectral domain-transfer learning, in: Proceeding of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, USA, 2008, pp. 488–496.

[17] T. Kato, H. Kashima, M. Sugiyama, Robust label propagation on multiple networks, IEEE Transactions on Neural Networks 20 (1) (2009) 35–44.

[18] G. R. G. Lanckriet, N. Cristianini, P. Bartlett, L. E. Ghaoui, M. I. Jordan, Learning the kernel matrix with semidefinite programming, Journal of Machine Learning Research 5 (2004) 27–72.

[19] X. Zhu, J. Kandola, Z. Ghahramani, J. Lafferty, Nonparametric transforms of graph kernels for semi-supervised learning, in: Advances in Neural Information Processing Systems 17, MIT Press, 2005, pp. 1641–1648.

[20] A. Argyriou, M. Herbster, M. Pontil, Combining graph laplacians for semi–supervised learning, in: Advances in Neural Information Processing Systems 18, MIT Press, 2006, pp. 67–74.

[21] D. P. Lewis, T. Jebara, W. S. Noble, Nonstationary kernel combination, in: Proceedings of the 23rd International Conference on Machine Learning, ACM, New York, NY, USA, 2006, pp. 553–560.

[22] C. S. Ong, A. J. Smola, T. C. Williamson, Learning the kernel with hyperkernels, Journal of Machine Learning Research 6 (2005) 1043–1071.

[23] M. Gönen, E. Alpaydin, Localized multiple kernel learning, in: Proceedings of the 25th International Conference on Machine Learning, ACM, New York, NY, USA, 2008, pp. 352–359.

[24] M. Filippone, F. Camastra, F. Masulli, S. Rovetta, A survey of kernel and spectral methods for clustering, Pattern Recognition 41 (1) (2008) 176–190.

[25] A. Banerjee, I. S. Dhillon, J. Ghosh, S. Sra, Clustering on the unit hypersphere using von Mises-Fisher distributions, Journal of Machine Learning Research 6 (2005) 1345–1382.

[26] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, B. Schölkopf, Learning with local and global consistency, in: Advances in Neural Information Processing Systems 16, MIT Press, 2004, pp. 321–328.

[27] R. I. Kondor, J. D. Lafferty, Diffusion kernels on graphs and other discrete input spaces, in: Proceedings of the 19th International Conference on Machine Learning, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002, pp. 315–322.

[28] A. J. Smola, R. I. Kondor, Kernels and regularization on graphs, in: Proceedings of the 16th Annual Conference on Computational Learning Theory, 2003, pp. 144–158.

[29] D. J. Hand, R. J. Till, A simple generalization of the area under the ROC curve for multiple class classification problems, Machine Learning 45 (2001) 171–186.

[30] H. Mamitsuka, Selecting features in microarray classification using ROC curves, Pattern Recognition 39 (12) (2006) 2393 – 2404. doi:DOI: 10.1016/j.patcog.2006.07.010.

[31] U. Güldener, et al., CYGD: the comprehensive yeast genome database, Nucleic Acids Research 33 (Database issue) (2005) D364–D368.