Key Vertex Pairs Extraction for Graph Classification based on Probability Distribution Learning

1st Jianming Huang
dept. of Computer Science and Communication Engineering
WASEDA University
Tokyo, Japan
koukenmei@toki.waseda.jp

2nd Hiroyuki Kasai
dept. of Computer Science and Communication Engineering
WASEDA University
Tokyo, Japan
hiroyuki.kasai@waseda.jp

***Abstract*—Graph classification is a hot topic of machine learning for graph-structured data, and it is also a very potential and valuable research. However, the difficulty of graph classification is challenging and special, which is quite different from the normal classification problems. One of the most difficult points of graph classification is that the numbers of vertex neighbors in graphs are usually variable, which makes the number of weights uncertain and ambiguous. Recent work such like the graph attention network apply the transformer on the graph neural network. However, the learned attentions cannot strictly reveal the importance of each part of graph, which makes the model less explainable. Moreover, for small datasets, it performs less effectively because of the excessive parameters. In order to overcome these difficulties, we propose a lightweight model with an edge weighting function based on the probability distributions of node pair features learned by the Gaussian mixture model. Although the proposed framework is simple, the experimental results shows its effectiveness on small datasets.**

***Index Terms*—Graph Classification, Probability Distribution Learning**

I. Introduction

Graph-structured data have been used widely in various fields, such as chemoinformatics, bioinformatics, social networks, and computer vision [15]. Therefore, there also exist many classification tasks for the graph-structured data, such as the toxicity analysis of compounds and recognitions of handwrittings. However, there exist some difficulties of the graph classification, most of which quite differ from other classfication problems of computer vision and natural language processing. One of the biggest problems is that, unlike the matrix data of a image or the sequence data of a sentence, the number of neighbors of a vertex in a graph is usually variable and uncertain. As shown in Figure 1, this makes it hard to learn the weights of neighbors because we cannot define a size-fixed parameter such like a convolution kernel in image data, or a sliding window in sequence data. To deal with these size-variable data, a related research named Graph Attention Networks (GAT) [14] propose a Graph Nerual Network (GNN) based on the attention mechanism similar to the well-known transformer [13], where they learn a weighting function to evaluate the attentions of node pairs in the feature aggregation step. The GAT brought great success to the node classification tasks. Nevertheless, for the graph classification tasks, it performs less effectively when node features of a whole graph needs to be aggregated together. It is also hard to prove that the learned attentions strictly reveal the importance of each part of the whole graph, which makes the model less explainable. Furthermore, for small graph datasets, the attention-based GNNs usually perform worse than graph kernel methods. This is because attention-based methods usually have more parameters (from the multi-head framework) than traditional methods and are more prone to over-fitting due to the excessive parameters.

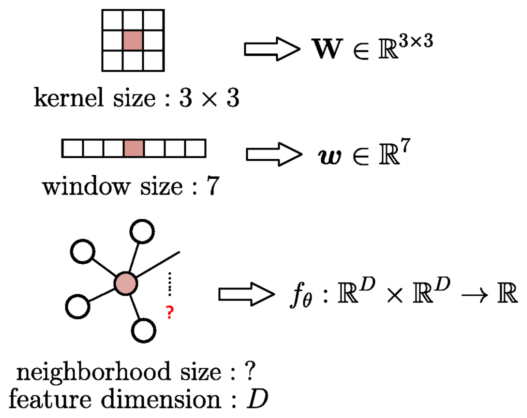
$\Rightarrow \mathbf{W} \in \mathbb{R}^{3\times 3}$
kernel size : 3×3
$\Rightarrow \boldsymbol{w} \in \mathbb{R}^{7}$
window size : 7
?
$\Rightarrow f_\theta : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$
neighborhood size : ?
feature dimension : D

Fig. 1. A vertex tends to have varible number of neighbors, which makes it hard to learn size-fixed parameters. Therefore, a weighting function based on features of vertex pairs might help.

In order to introduce the idea of attention in graph classification tasks on small graph datasets while avoiding the excessive deep learning parameters, we propose a lightweight model with attentions learned by the Gaussian Mixture Model (GMM). We replace the deep attention learning with the GMM, a conventional machine learning method, which greatly reduces the number of deep learning parameters. In more details, we utilize probability distributions of aggregated node pair features learned by GMM to compute scalar attentions for edges instead of adjusting it through deep learning. With these edge weights, we then update node features with a simple sum aggregation message passing framework and finally classify it with a single linear layer. Although we still use deep learning to adjust the parameters of linear layers in our experiments, the number of these parameters is small, and they only function as independent classifiers and do not participate in the message passing process. Fig 5 presents the overall framework of our proposed model. It shows that our proposed model has extremely few learnable parameters, all of which come from the single linear layers for each class. However, although there are very few deep learning parameters in our model, the experimental results show that our proposed method can still outperform many state-of-the-art graph kernel methods and GNNs in graph classification tasks. Our contributions could be summarized as follows:

- We propose a lightweight message-passing model with scalar attentions computed by the GMM-learned probability distribution of node pair features. It avoids the performance degradation of the traditional attention-based GNNs on small datasets. Moreover, it also provides an explainable scheme of edge weighting;
- We extend our model to a multi-layer and multi-class framework, which make it applicable to different classification tasks. The evaluation results show that our proposed model is simple but effective, which has a better performance than many state-of-the-art methods.

II. Preliminaries

This section introduces some notations and prelimiaries. We use lower-case letters in bold typeface to express the vectors such like $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$. For matrices, we express them by upper-case letters in bold typeface such like $\mathbf{A}, \mathbf{B}, \mathbf{C}$. $\mathbb{R}$ denotes the set of real numbers.

A. Graph Attention Network

Let $\boldsymbol{h}_i^{(t)} \in \mathbb{R}^d$ denote the d-dimensional hidden feature of i-th node at t-th iteration, and let $\mathcal{N}(i)$ be the set of indices of the neighbors of the i-th node. The graph attention network, as a message-passing-based GNN, updates node features with the equation below.

$$\boldsymbol{h}_i^{(t+1)} = \sigma\left(\sum_{j\in\mathcal{N}(i)} \alpha_{i,j}\mathbf{W}\boldsymbol{h}_j^{(t)}\right),$$

where $\mathbf{W}$ denotes the learnable parameter, and $\alpha_{i,j}$ denotes the learned attention of i-th and j-th node. The attention is computed by the following equation.

$$\alpha_{i,j} = \frac{\exp\left(\text{LeakyReLU}(\boldsymbol{a}[\mathbf{W}'\boldsymbol{h}_i||\mathbf{W}'\boldsymbol{h}_j])\right)}{\sum_{k\in\mathcal{N}(i)}\exp\left(\text{LeakyReLU}(\boldsymbol{a}[\mathbf{W}'\boldsymbol{h}_i||\mathbf{W}'\boldsymbol{h}_k])\right)},$$

where $\boldsymbol{a}, \mathbf{W}'$ are the learned parameters, the former is a vector and the later is a matrix. $||$ denotes the vector concatenation operation.

B. Multivariate Gaussian Mixture Model

The multivariate GMM is a category of the mixture probabilistic models, which aims to represent the probability distribution of the observed data. The multivariate GMM is a combination of several multivariate normal distributions. Normally, a D-dimensional GMM with K components is expressed as $p(\boldsymbol{x}) = \sum_{i=1}^{K}\alpha_i N(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, where $\boldsymbol{x} \in \mathbb{R}^D$ denotes an observation of the data which we want to model. $N(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ denotes a normal distribution with the mean vector $\boldsymbol{\mu}_i \in \mathbb{R}^D$ and the covariance matrix $\boldsymbol{\Sigma}_i \in \mathbb{R}^{D\times D}$. α_i denotes the weight of each component. The most popular way to solve the optimal parameters $\alpha_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i$ is the expectation-maxmization (EM) algorithm [17] with non-convex clustering initialization.

III. Related Work

For graph classification tasks, Graph Kernel (GK) methods were the mainstream methods and it has been used widely for several decades before Graph Neural Networks (GNN) came out. Theoretically, GKs are kernel functions which could compute similarity for graph pairs. Most of GKs are based on the isomorphism and structural similarity of graph-structured data. In earlier researches, GKs have shown its effectiveness for graph classification tasks with machine learning algorithms including the Support Vector Machine (SVM). A famous work of GK is the Weisfeiler–Lehman (WL) Graph Kernel [10], which brought great success in this domain and it is still inspiring many state-of-the-art reseaches now. In WL graph kernel, They proposed a similarity metric based on the Weisfeiler–Lehman test and implement it as a general framework. To improve the similarity metric of the WL graph kernel and make it more robust for the decomposite graph structure, related work [8] brought an optimal assignment kernel variant from the WL graph kernel, which is called Weisfeiler–Lehman Optimal Assignment (WL-OA) graph kernel. It is based on an optimal bijection between substuctures of graph pairs, which performs better and more robust than the original WL graph kernel. Similar to the WL-OA graph kernel, there is a research [12] proposing a Wasserstein-based Weisfeiler–Lehman (WWL) Graph Kernel, which maps a node embedding based on its neighborhood pattern to a novel feature space. In this feature space, they compute distance of graphs based on the Wasserstein distance of two point clouds where a single point denotes a node of graphs. There are many other impressive works of the Wasserstein-based graph kernels [3], [5] and works beyond the WL test [6].

In recent years, as the large-scale datasets become more important, the shortcomings of GKs become more obvious: the high cost of both the computational complexity and the memory. Thus, GNNs become hot topics in graph domain including the graph classification, the node prediction and the link prediction. A representative method of GNN in recent years is the Graph Convolution Network (GCN) [7], which is inspired by the Convolution Neural Network (CNN) in computer vision domain and applies the same way to deal with graph-structured data. Another representative GNN is the Graph Isomorphism Network (GIN) [16]. The GIN is inspired by the WL graph kernel which specifically examines graph isomorphism. They prove that the SUM aggregation generates better aggregation than other schemes including MEAN and MAX, in message passing process of graphs. They apply this

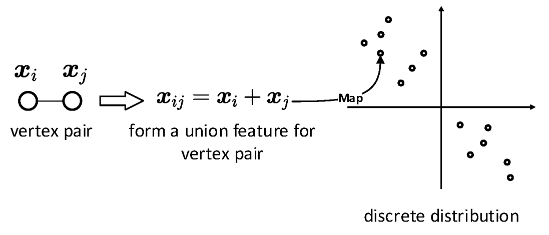

Fig. 2. Overview of step 1, where connected vertex pairs will be mapped to a discrete distribution.

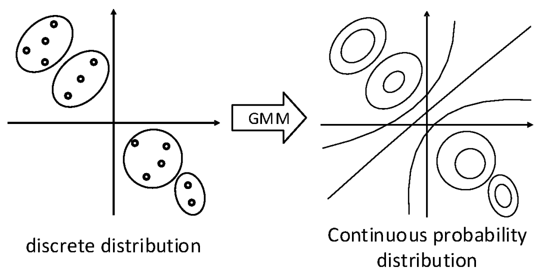

Fig. 3. Overview of step 2, where we use the GMM to learn a continuous probability distribution.

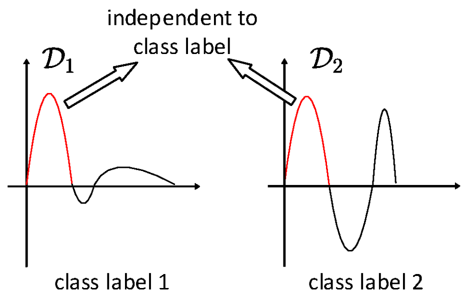

Fig. 4. Illustration of the independent parts in distributions.

scheme in GIN, which make it more powerful than other GNNs in graph classification tasks.

IV. Probability Distribution Learning for Binary Classification

In this section, to make it easy to understand, we first propose a probability distribution learning scheme of vertex pairs under a simple task: the binay classification. We will then expand it for the multi-class classfication in the next section.

A. Step 1: Sample the vertex pairs and map them to a discrete distribution

As described in the introduction section, in order to compute the scalar attentions, we suppose to learn the probability distribution of the node pair features. We start from a vertex pairs sampling operation as our Step 1. Firstly, to represent the vertex pairs, we form a union feature vector for a vertex pair by combining the single vertex features. As shown in Figure 2, before the process of sampling, we compute the feature vector of vertex pair which will form a discrete distribution. We suppose that the representation of vertex pair should be invaried regardless of different input orders, because they are structurally isomorphic. By considering this situation, we apply the summation of the vertex features as our vertex pair representation. Let $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^d$ be the feature of two connected vertices and let $f_{\text{pair}} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ be the function aggregating features of these two vertices, the aggregated feature of vertex pair is

$$f_{\text{pair}}(\boldsymbol{x}_1, \boldsymbol{x}_2) = \boldsymbol{x}_1 + \boldsymbol{x}_2.$$

After the aggregation, we can obtain a d-dimensional vector for each connected vertex pairs, which can be mapped to a d-dimensional Euclidean metric space.

For the process of sampling, let us consider a binary classification task, we first divide the graphs into 2 subsets corresponding to their class labels. Next, all connected vertex pairs of graphs in the subset of i-th class label will be mapped to a distribution $\hat{\mathcal{D}}_i$. Thereafter, we can get 2 discrete distrbutions $\hat{\mathcal{D}}_1, \hat{\mathcal{D}}_2$ for a binary classification task, which are two discrete point clouds including all the vertex pair samples.

B. Step 2: Use the GMM to learn the probability distribution

One could use the sampled discrete distribution to compute an attention-like weight for a newly inputted vertex pair. However, it might bring great difficulties for computation, which might lead to high computational costs and memory costs because of too many point comparisons. Therefore, to simplify the computational process, we use distribution learning methods such as the GMM to learn a continuous probability distribution of vertex pairs as shown in Figure 3, where we use a C-component GMM to approximate the probability distribution. Thereafter, for each point cloud $\hat{\mathcal{D}}_i$, we can obtain a probality function $\mathcal{D}_i : \mathbb{R}^d \to \mathbb{R}$, which takes a vertex pair feature vector as input and outputs its probability density.

C. Step 3: Compute a combined distribution

Through analyzing the obtained continuous distributions, we found that there usually exist some parts of these distributions which are relatively independent to different class labels. As shown in Figure 4, the red part does not change too much w.r.t class label 1 and class label 2. These parts should be considered as features less valuable for classification because they have a similar probability model w.r.t both the class label 1 and 2. In order to filter out these independent parts, we apply a simple way to combine two distribution as:

$$\mathcal{D}' = |\mathcal{D}_1 - \mathcal{D}_2|,$$

where $\mathcal{D}_1, \mathcal{D}_2$ are two distributions and $\mathcal{D}'$ is their combined distribution.

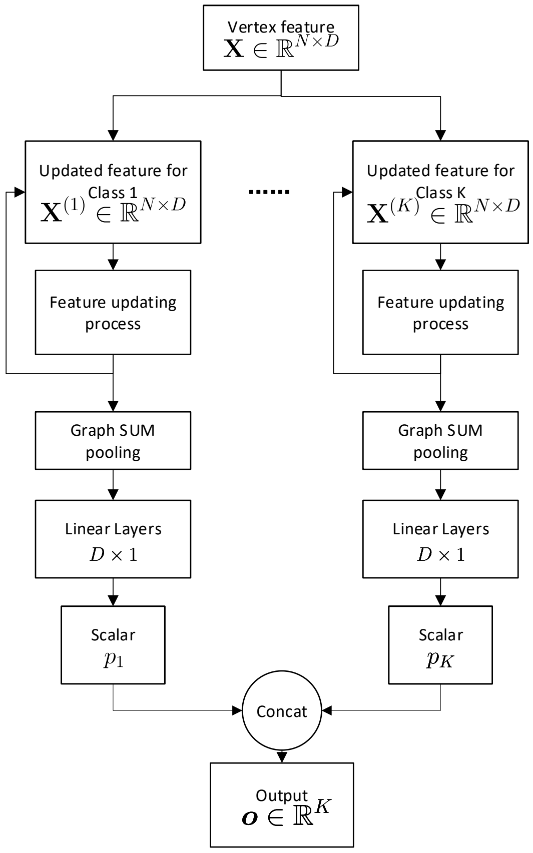
Vertex feature
$\mathbf{X} \in \mathbb{R}^{N\times D}$
Updated feature for Class 1
$\mathbf{X}^{(1)} \in \mathbb{R}^{N\times D}$
Updated feature for Class K
$\mathbf{X}^{(K)} \in \mathbb{R}^{N\times D}$
Feature updating process
Graph SUM pooling
Linear Layers
$D \times 1$
Scalar
p_1
p_K
Concat
Output
$\boldsymbol{o} \in \mathbb{R}^K$

Fig. 5. Illustration of the framework for the multi-class classification task, where the vertex features of a single graph will be updated separately w.r.t each class label and finally they will be concatenated into an ouput vector.

D. Step 4: Update each vertex feature

Similar to most graph classification methods, we apply a message-passing-based framework to update and aggregate vertex features in graphs. In this step, feature of each vertex will be updated by aggregating its and its neighbors' features. Let $\boldsymbol{x}_i \in \mathbb{R}^d$ be the feature vector of i-th vertex in a graph, then it will be updated as

$$\boldsymbol{x}'_i = \sum_{j\in\mathcal{N}(i)} f_\theta(\boldsymbol{x}_i, \boldsymbol{x}_j) \cdot \boldsymbol{x}_j, \tag{1}$$

where $f_\theta : \mathbb{R}^D \times \mathbb{R}^\theta \to \mathbb{R}$ is:

$$f_\theta(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{\exp\left(\mathcal{D}'(\boldsymbol{x}_i + \boldsymbol{x}_j)\right)}{\sum_{k\in\mathcal{N}(i)} \exp\left(\mathcal{D}'(\boldsymbol{x}_i + \boldsymbol{x}_k)\right)},$$

where $\mathcal{N}(i)$ denotes the set of neighbors of i-th vertex.

The step 1 to step 4 will repeat H iterations for updating. If the process of updating ends, all vertex features will be inputted into a global mean pooling layer to get a graph representation, which will be used for classification.

V. Class-wise Feature Updating for Multi-class Classification

So far, the feature updating process w.r.t a single class-label is described in the previous section, which can be directly used in the binary classification task. However, there are many classification tasks that have more than 2 class labels. In order to extend our method so that it can work in the multi-class classification task, we conduct a separate-and-concatenate framework which is as shown in the Fig. 5. Let G be the graph that we are going to transform and classify, $\mathbf{X} \in \mathbb{R}^{N\times D}$ denotes its vertex features, which has N vertices and each vertex is embedded into a D-dimensional vector. Assume that we have K class labels. In this case, we first copy the vertex features and repeat them into K channels, which are denoted as $\mathbf{X}^{(1)}...\mathbf{X}^{(K)}$. The vertex features in K channels will be updated separately and independently for a proper number of iterations, corresponding to the class label that they belong to. This means that, in the probability distribution learning step of the i-th channel, graphs will be divided into two subsets: belonging to class label i and not belonging to class label i, which correspond to the $\hat{\mathcal{D}}_1, \hat{\mathcal{D}}_2$ in Section IV-A. Then the probability distribution is learned as described in the previous section. After finishing the feature updating process, we will conduct a graph SUM pooling on the vertex features, where the vertex features will be added together to form a new embedding for the graph G. Then we will apply several linear layers to $\mathbf{X}^{(1)}...\mathbf{X}^{(K)}$ separately, which take an input of D dimension and give an output of 1 dimension. Then we can obtain K scalars $p_1..p_K$, each of them denotes the probability of belong to their class labels. Finally, these scalars will be concatenated together to form an output $\boldsymbol{o} \in \mathbb{R}^K$, which will be inputted into a log-softmax classifier. Overall, our proposed model does not require any deep learning process except a linear classifier.

VI. Complexity Analysis

In this section, we present the computational comlexity analysis of our proposed framework. Let n, m denote the number of nodes and the number of edges in a single graph, respectively. Let K be the number of components of GMM, and let l be the number of message passing layers, and let D be the number of feature dimensions. Our proposed method needs $O(lmKD^2)$ to compute the edge weight $f_\theta(\boldsymbol{x}_i, \boldsymbol{x}_j)$ in Eq. (1), because the commputational complexity of GMM is $O(KD^2)$. For the GAT, its complexity is $O(l(anFD+amD))$ [14], where a denotes the number of multi-heads, D, F denote the number of input and output dimension, respectively.

VII. Numerical Evaluation

We conduct evaluation experiments on several widely-used real world benchmark datasets, which are the MUATG [2], the PTC-MR [4], the PROTEINS [1], the COX2 [11], the AIDS [9] datasets. For each dataset, we conduct one time of 10-fold nested cross validation to get the average accuracy and standard deviation. For the parameter setting our proposed method, we use the GMM with 100 clusters and update vertex features for $H = 4$ iterations.

For the comparing methods, we choose 4 state-of-the-art works, which are the Weisfeiler-Leman graph kernel (WL) [10], the Wasserstein Weisfeiler-Leman graph kernel (WWL)

TABLE I
AVERAGE CLASSIFICATION ACCURACY ON GRAPH DATASETS

METHOD	MUTAG	PTC-MR	PROTEINS	COX2	AIDS
WL [10]	85.61±8.02	62.51±4.11	74.24±3.75	81.36±3.21	97.90±0.95
WWL [12]	**85.90±7.39**	**65.31±7.06**	74.13±3.47	81.75±3.71	98.23±0.96
WL-OA [8]	82.72±7.09	63.45±8.63	73.83±3.61	80.47±4.44	**99.02±0.61**
GIN [16]	**88.59±6.89**	64.76±7.67	73.72±4.27	**82.59±4.42**	97.87±1.08
GAT [14]	72.83±9.80	54.98±8.55	70.69±3.53	81.74±3.83	91.00±1.89
unweighted	74.03±8.79	60.16±9.03	**76.82±3.82**	78.15±0.80	98.12±0.44
weighted	80.87±10.53	**65.42±7.40**	**75.74±4.31**	**83.07±4.94**	**99.14±0.55**

[12], the Weisfeiler-Leman graph kernel with Optimal Assignment (WL-OA) [8], the Graph Isomorphism Nerual Network (GIN) [16] and the Graph Attention Network (GAT) [14]. For graph kernel methods WL, WWL and WL-OA, we apply a grid search with $H \in \{1,2,3,4,5,6,7,8\}$ and utilize the Support Vector Machine (SVM) for classification. For GIN, we also apply grid searchs with $H \in \{1,2,3,4,5,6,7,8\}$ and the hidden dimensions within $\{32, 64, 128\}$. For GAT, we apply grid searchs with $H \in \{1,2,3,4,5,6,7,8\}$ and the number of heads within $\{1,2,3,4\}$. We set the hidden dimensions for each head as 32. In order to show the difference between weighted strategy and unweighted strategy, we also add an unweighted variant of our proposed method, where we update vertex feature by the following equation:

$$\boldsymbol{x}'_i = \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} \boldsymbol{x}_j.$$

The experimental results are shown in Table I, where the top-2 are in bold typeface. From the results we can see that our proposed method outperforms all comparing methods in the PTC-MR, the BZR and the COX2 datasets. For the MUTAG dataset, although our method is not the best, it obtain a nice accuracy which is not far from other methods.

VIII. CONCLUSION

In this paper, we propose a lightweight message passing method based on a weighting function learned by applying the GMM to the probabiliry distributions of vertex pairs. The experimental results show its effectiveness in classification performance. For future works, we have several direction: 1) Apply sampling methods such as the Gibbs sampling to reduce computational costs; 2) Use deep distribution learning methods to learn continuous distributions for better outcomes.

REFERENCES

[1] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel, "Protein function prediction via graph kernels," *Bioinformatics*, vol. 21, no. suppl_1, pp. i47–i56, 2005.

[2] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch, "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity," *Journal of medicinal chemistry*, vol. 34, no. 2, pp. 786–797, 1991.

[3] Z. Fang, J. Huang, X. Su, and H. Kasai, "Wasserstein graph distance based on l1–approximated tree edit distance between weisfeiler–lehman subtrees," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2023.

[4] C. Helma, R. D. King, S. Kramer, and A. Srinivasan, "The predictive toxicology challenge 2000–2001," *Bioinformatics*, vol. 17, no. 1, pp. 107–108, 2001.

[5] J. Huang, Z. Fang, and H. Kasai, "Lcs graph kernel based on wasserstein distance in longest common subsequence metric space," *Signal Processing*, vol. 189, p. 108281, 2021.

[6] J. Huang and H. Kasai, "Graph embedding using multi-layer adjacent point merging model," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021.

[7] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *International Conference on Learning Representations (ICLR)*, 2016.

[8] N. M. Kriege, P.-L. Giscard, and R. C. Wilson, "On valid optimal assignment kernels and applications to graph classification," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[9] K. Riesen and H. Bunke, "Iam graph database repository for graph based pattern recognition and machine learning," in *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. 2008, pp. 287–297.

[10] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *The Journal of Machine Learning Research*, vol. 12, pp. 2539–2561, 2011.

[11] J. J. Sutherland, L. A. O'brien, and D. F. Weaver, "Spline-fitting with a genetic algorithm: A method for developing classification structure-activity relationships," *Journal of chemical information and computer sciences*, vol. 43, no. 6, pp. 1906–1915, 2003.

[12] M. Togninalli, E. Ghisu, F. Llinares-López, B. Rieck, and K. Borgwardt, "Wasserstein weisfeiler-lehman graph kernels," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[14] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *International Conference on Learning Representations (ICLR)*, 2017.

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

[16] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *International Conference on Learning Representations (ICLR)*, 2019.

[17] G. Yu, G. Sapiro, and S. Mallat, "Solving inverse problems with piecewise linear estimators: From gaussian mixture models to structured sparsity," *IEEE Transactions on Image Processing*, vol. 21, no. 5, pp. 2481–2499, 2011.