Neurocomputing 397 (2020) 114–126

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Feature reduction based on semantic similarity for graph classification

Zhigang Sun[a], Hongwei Huo[a,*], Jun Huan[b], Jeffrey Scott Vitter[c]

[a] School of Computer Science and Technology, Xidian University, Xi'an 710071, China
[b] StylingAI Inc, Beijing 100094, China
[c] Department of Computer & Information Science, the University of Mississippi, MS 38677-1848, USA

ARTICLE INFO

Article history:
Received 16 March 2019
Revised 18 December 2019
Accepted 7 February 2020
Available online 14 February 2020

Communicated by Prof. Sanguineti Marcello

Keywords:
Graph classification
Feature reduction
Neural language model
Semantic similarity

ABSTRACT

Classification and recognition of graph data are crucial problems in many fields, such as bioinformatics, chemoinformatics and data mining. In graph kernel-based classification methods, the similarity among substructures is not fully considered; in addition, poorly discriminative substructures will affect the graph classification accuracy. To improve the graph classification accuracy, we propose a feature reduction algorithm based on semantic similarity for graph classification in this paper. In the algorithm, we first learn vector representations of subtree patterns using neural language models and then merge semantically similar subtree patterns into a new feature. We then provide a new feature discrimination score to select highly discriminative features. Comprehensive experiments on real datasets demonstrate that the proposed algorithm achieves a significant improvement in classification accuracy over compared graph classification methods.

1. Introduction

Graph is a general data structure that is widely used to model complex objects and dependency relationships among them. Graph classification has been an important task in graph data mining, and it has various applications in areas such as cheminformatics, bioinformatics and society network analysis [19,26,28]. In cheminformatics, predicting the toxicity and bioactivity of compounds is a classic example of graph classification, where compounds are represented as graphs.

In most of the existing graph classification methods [3,4,10,14,21,23,38,39,41], graphs are first decomposed into substructures, such as subtrees and frequent subgraphs [37]; and then are represented as graph feature vectors via graph embedding or calculating the occurrence of substructures; finally, graph feature vectors are used to train conventional classifiers, such as SVM [5,7], ELM [15,16] and DNN [20]. Recently, there are also some neural language models-based algorithms for learning the vector representations of graphs, such as node2vec [13], graph2vec [24], and S2S-N2N-PP [34].

Graph kernels are classic graph similarity measures in graph classification. Graph kernels decompose graphs into atomic substructures, such as graphlets, walks, shortest paths, subtree patterns and k-discs [1,8,17,18,29,31,33], and they define the similarity of two graphs as the number of their common substructure pairs. In the graphlet kernel [29], a graphlet is an induced and non-isomorphic subgraph of five vertices or less in general. In the shortest path kernel [17], graphs are decomposed into a series of triples including the labels of two vertices and the length of the shortest path between the two vertices. The random walk kernel [1] randomly walks on two graphs and counts the number of matching walks. The Weisfeiler-Lehman (WL) subtree kernel [31] decomposes a graph into a series of subtree patterns. The graph wavelet alignment kernel [33] obtains multiscale features of each vertex of a graph using discrete wavelet functions, and defines the similarity of two graphs as the sum of kernels of every pair of aligned vertices from the graphs. Costa et al. [8] proposed neighborhood subgraph pairwise distance kernel (NSPDK). NSPDK defined the similarity of two graphs as the number of common *k*-disc pairs within given distance. Kriege et al. [18] gave a theoretical evaluation to the expressivity of graph kernels based on property testing framework [25]. Kriege et al. found that several established graph kernels cannot distinguish graph properties, such as connectivity, planarity, bipartieness or triangle freeness, and then proposed *k*-disc graph kernel which can distinguish connectivity, planarity and triangle freeness. However, there are some literatures [31,38] that demonstrated the validity of existing graph kernels.

However, graph kernel-based classification methods have two limitations: (1) graph kernels regard substructures of graphs as atomic structures and do not fully consider the similarity among substructures; (2) the feature matrix of a graph dataset is very

* Corresponding author.
E-mail addresses: szung@163.com (Z. Sun), hwhuo@mail.xidian.edu.cn (H. Huo), lukej.huan@yahoo.com (J. Huan), jsv@OleMiss.edu (J.S. Vitter).

https://doi.org/10.1016/j.neucom.2020.02.047
0925-2312/

sparse, and substructures with low discrimination have an effect on the graph classification accuracy. To overcome these limitations, Yanardag et al. [38] proposed Deep graph kernels based on neural language models. In the Deep graph kernels, neural language models are used to learn vector representations of substructures, and the similarity of substructures is defined according to the vector representations of the substructures. subgraph2vec [23] is another method that was proposed to learn vector representations of rooted subgraphs using neural language models. Compared with the Deep graph kernels, for each vertex v in a graph, subgraph2vec viewed subgraphs rooted at neighbors of v as the context information of the subgraph rooted at v. Yu et al. [39] proposed a sparse graph feature selection method for graph classification. Considering the sparsity of the feature matrix of a graph dataset, they used LASSO to select key features. Ma et al. [21] proposed a graph classification method based on graph dataset reconstruction and kernel feature reduction. They first reconstructed the graph dataset by deleting subgraphs with a low discrimination score from the graph dataset, and then they used kernel discriminant analysis to reduce the dimension of features. Smalter et al. [32] proposed a pattern-based highly discriminative patterns mining method for chemical compound classification. To reduce the gap between graph feature selection and classifier training, Pan et al. [27] proposed a regularized loss minimization subgraph selection method. This method integrated feature generation, feature selection and graph classifier training into a unified framework and determined the number of selected features by minimizing the loss. However, the above methods only attempt to solve one of the two limitations.

To overcome the two above limitations, we propose a graph feature reduction method based on semantic similarity for graph classification in this paper. The main contributions of this paper include the following:

(1) We propose a new corpus building method for subtree patterns occurring in a graph dataset. Then we learn vector representations for subtree patterns using neural language models so that semantically similar subtree patterns can be mapped to near positions in the vector space.
(2) We divide the subtree patterns into blocks according to the Euclidean distances between vector representations of the subtree patterns and the origin, and we apply AP clustering to each block to merge semantically similar subtree patterns in each cluster. In addition, we design a feature discrimination score based on feature occurrence to select highly discriminative features.
(3) We train an FRS_KELM graph classifier using the highly discriminative features. Comprehensive experiments on real datasets show that the proposed graph classification method achieves a significant improvement in classification accuracy over compared graph classification methods.

The remainder of this paper is organized as follows. In Section 2, we introduce the problem definition and neural language models. In Section 3, we describe the subtree pattern merging method based on semantic similarity and the feature selection method based on feature discrimination score in detail. Then, we present the overall graph classification method. Comprehensive experimental studies are shown in Section 4. We provide concluding remarks in Section 5.

2. Preliminaries

In this section, we provide formal definitions of subtree pattern and corpus, and then we introduce neural language models. Here, we focus on undirected vertex-labeled simple graphs, where each vertex of the graphs has a discrete label. Specifically, a vertex-labeled simple graph g can be denoted as a four-tuple $(V_g, E_g, \Sigma_g, \lambda)$, where V_g is the set of vertices, $E_g \subseteq V_g \times V_g$ is the set of edges, Σ_g is the set of vertex labels, and λ is the mapping function from a vertex to its label.

2.1. Problem definition

The WL subtree graph kernel is a fast graph kernel based on WL test of isomorphism. This graph kernel decomposes graphs into subtree patterns and defines the similarity of two graphs as the number of their common subtree pattern pairs.

Definition 1 (Subtree Pattern)**.** Consider a graph $g = (V_g, E_g, \Sigma_g, \lambda)$ and a vertex $v \in V_g$. Let $N(v)$ be the set of neighbor vertices adjacent to v. The i-hop subtree pattern rooted at v, $i \geq 0$, denoted as $P_i(v)$, is recursively defined as an i-level tree: it is the root node v for $i = 0$; otherwise it is a tree rooted at v, and has $|N(v)|$ subtrees, whose jth subtree is an $(i-1)$-hop subtree pattern rooted at the jth entry of $N(v)$ for $1 \leq j \leq |N(v)|$. The feature multiset of g consisting of all i-hop subtree patterns can be defined as $f_i(g) = \bigcup_{v \in V_g} P_i(v)$.

Each iteration of the WL subtree kernel [31] maps a graph into a label multiset, where each label in the multiset corresponds to a subtree pattern of the graph. Given a graph g, the label multiset generated at the ith iteration of the WL subtree kernel exactly equals $f_i(g)$. Therefore, we use the WL subtree kernel to obtain the subtree pattern multiset of a graph.

Given a graph dataset $G = \{g_1, g_2, \ldots, g_n\}$ and a graph $g \in G$. Let $F_t(g)$ be the feature multiset of g consisting of all subtree patterns from 0-hop to t-hop occurring in g, namely, $F_t(g) = \bigcup_{i=0}^{t} f_i(g)$. We define the set of i-hop feature multisets of G consisting of the i-hop feature multisets of all graphs in G as $\mathbb{F}_i(G) = \{f_i(g_j) \mid 1 \leq j \leq n\}$, and the set of $\mathbb{F}_i(G)$ for $0 \leq i \leq t$ as $\mathcal{F}_t(G) = \{\mathbb{F}_i(G) \mid 0 \leq i \leq t\}$.

We have the following observations on the subtree patterns that occur in a graph dataset G: subtree patterns occurring in one graph rarely occur in other graphs. The number of subtree patterns is proportional to the product of the number of vertices in G and the number of WL iterations. We can transform a subtree pattern into another by changing some vertex labels.

Definition 2 (Feature Occurrence)**.** Given a training graph dataset G and its label set Y, where $Y = \{y_1, y_2, \ldots, y_n\}$ is the set of the class labels of graph $g_i \in G$ with l distinct class labels, and the set of graphs with the ith label is denoted as G_i. Let $\mathcal{F}_t(G_i)$ be the feature set of G_i; then, the occurrence of a feature ft in $\mathcal{F}_t(G_i)$ can be defined as $CNT(ft, \mathcal{F}_t(G_i)) = \sum_{j=0}^{t} \sum_{(g \in G_i) \wedge (f_j(g) \in \mathbb{F}_j(G_i))} cnt(ft, f_j(g))$, where $cnt(ft, f_j(g))$ is the number of occurrence of ft in $f_j(g)$.

Example 1. In Fig. 1, we present an example of labeled graphs and their subtree patterns. Fig. 1(a) and (b) are two labeled graphs g_1 and g_2, respectively, where the number in each vertex is its vertex label. Fig. 1(c) and (d) are the 0-hop and 1-hop subtree patterns of g_1 and g_2, respectively, where the number under a subtree pattern is its occurrence in the graph. As shown in the figures, subtree pattern 6 can be transformed into subtree pattern 8 by only changing its root node's label.

Definition 3 (Graph Classification Problem)**.** Given a training graph dataset G and its label set Y, the goal of graph classification is to learn a graph classifier using G and Y such that given a set of testing graphs without class labels, the graph classifier can predict the class labels of these graphs. We use graph classification accuracy ACC to measure the graph classifier, defined as follows:

$$ACC = \frac{\#\text{predictions that are correct}}{\#\text{graphs in the tested graph dataset}}$$

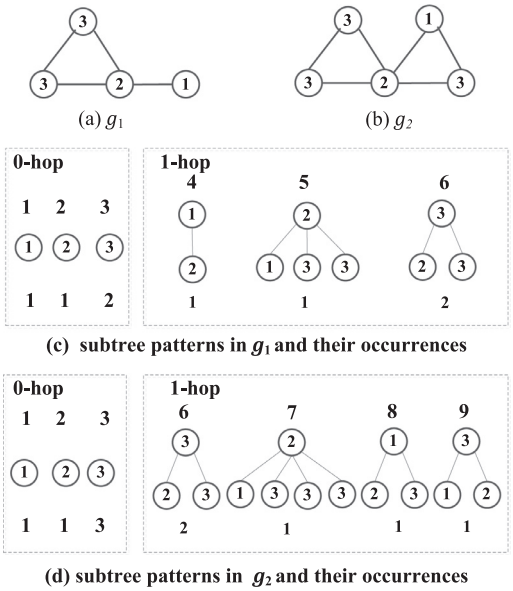


Fig. 1. Labeled graphs g_1 and g_2, and their subtree patterns and occurrences.

Definition 4 (Corpus). Given a graph dataset $G = \{g_1, g_2, \ldots, g_n\}$, a corpus Cp_i built for $\mathbb{F}_i(G)$ can be defined as a tuple (Ω_i, S_i). Here Ω_i is a vocabulary that is the set of distinct subtree patterns in $f_i(g)$ for $f_i(g) \in \mathbb{F}_i(G)$, and S_j is a set of word (subtree pattern) sequences generated from $\mathbb{F}_i(G)$, where the jth word sequence is a permutation of subtree patterns in $f_i(g_j)$.

2.2. Neural language model

Given a corpus $Cp = (\Omega, S)$, neural language models learn the continuous vector of each word w in Ω using the context information of w, where the context of w is defined as a fixed number of words preceding and following w in word sequences of S. In this section, we introduce the following two neural language models: Continuous Bag-of-Words model and Skip-gram model [22].

Continuous Bag-of-Words (CBOW). CBOW model aims to predict the current word according to the context of the word. Given a word sequence $s \in S$, where $s = [w_1, w_2, \ldots, w_T]$. Let $w_{t-n}, \ldots, w_{t-1}$ and $w_{t+1}, \ldots, w_{t+n}$ be the context of word w_t. The goal of CBOW model is to maximize the following log-likelihood by mapping each word in Ω to a continuous vector:

$$\sum_{t=1}^{T} \log p(w_t | w_{t-n}, \ldots, w_{t-1}, w_{t+1}, \cdots, w_{t+n})$$

where $p(w_t | w_{t-n}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+n}) = \frac{\exp(\overline{\Phi(w_t)} \cdot \Phi'(w_t))}{\sum_{i=1}^{|\Omega|} \exp(\overline{\Phi(w_t)} \cdot \Phi'(w_i))}$. Here $\Phi(w)$ and $\Phi'(w)$ are the input vector and output vector of word w. $\overline{\Phi(w_t)}$ is the average vector of the input vectors of w_t's context words.

Skip-gram model. In contrast to CBOW model, the Skim-gram model aims to predict context words according to the current word. Given a word sequence $s \in S$, where $s = [w_1, w_2, \ldots, w_T]$, the goal of Skip-gram model is to maximize the following log-likelihood by mapping each word in Ω to a continuous vector:

$$\sum_{t=1}^{T} \log p(w_{t-n}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+n} | w_t)$$

where $p(w_{t-n}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+n} | w_t) = \prod_{-n \le j \le n, j \ne 0} \frac{\exp(\Phi(w_t) \cdot \Phi'(w_{t+j}))}{\sum_{i=1}^{|\Omega|} \exp(\Phi(w_t) \cdot \Phi'(w_i))}$.

3. Methods

In this section, we describe the method of learning vector representations for subtree patterns in Section 3.1, and then we introduce two feature reduction methods: subtree pattern merging based on semantic similarity and graph feature selection based on discrimination score in Sections 3.2 and 3.3, respectively. We present the overall graph classification method using reduced graph features in Section 3.4.

3.1. Learning vector representations for subtree patterns

To learn the vector representations of subtree patterns in a graph dataset G, we build corpora for subtree patterns in G, and then we use neural language models to learn continuous vector representations of subtree patterns in this section.

3.1.1. Building corpus for subtree patterns

Given a graph $g \in G$, the i-th WL subtree iteration on g generates a feature multiset $f_i(g)$ that consists of all i-hop subtree patterns occurring in g. If we view a subtree pattern as a word, then subtree patterns in $f_i(g)$ can be viewed as co-occurring words. By concatenating all subtree patterns in $f_i(g)$, we can generate a word sequence in which a word and its context have a co-occurring relationship. For any two subtree pattern multisets $f_i(g)$ and $f_j(g)$, if $i \ne j$, then $f_i(g) \cap f_j(g) = \varnothing$, which means that subtree patterns generated from different WL iterations do not have a co-occurring relationship. Therefore, in contrast to the corpus building method in Deep WL kernel [38], which concatenates all subtree patterns in $F_t(g)$ to form a word sequence and builds only one corpus for G, we build a corpus set for G, where each corpus corresponds to a feature set $\mathbb{F}_i(G)$. For example, if the total number of WL iterations on G is t, then we build a corpus set $Cps = \{Cp_0, Cp_1, \ldots, Cp_t\}$ for G, where Cp_i is the corpus built for subtree patterns generated on the i-th WL subtree iteration on G.

Given a graph dataset G and the number of WL subtree iterations t, the procedure of building a corpus set Cps for G is shown in Algorithm 1. The algorithm iterates $t+1$ times. In the i-th iteration, we obtain the feature set $\mathbb{F}_i(G)$ for G using WL subtree iteration in line 3; we then build a corpus Cp_i by calling procedure *buildCp()* in line 4.

In procedure *buildCp*, we build a corpus $Cp = (\Omega, S)$ for a given feature set $\mathbb{F}_i(G)$. Ω consists of all distinct subtree patterns in $f_i(g)$ for $f_i(g) \in \mathbb{F}_i(G)$, as shown in line 2. In the iteration of lines 3–7,

Algorithm 1: *BuildCorpus*(G, t).

```
1 Cps ← ∅
2 for i ← 0 to t do
3     get F_i(G) of G using WL subtree iteration
4     Cp_i ← buildCp(F_i(G))
5     Cps ← Cps ∪ {Cp_i}
6 return Cps

procedure buildCp(F_i(G))
1 S ← ∅
2 Ω ← the set of distinct subtree patterns occurring in F_i(G)
3 for each feature multiset f_i(g) in F_i(G) do
4     s ← ∅
5     sort st_j ∈ f_i(g) to form the ascending order list L
6     concatenate elements in L into a sentence s
7     S ← S ∪ {s}
8 Cp ← (Ω, S)
9 return Cp
```

we build the word sequence set S. For each $f_i(g) \in \mathbb{F}_i(G)$, we build a word sequence s. In line 5, we sort the elements of $f_i(g)$ to form an ascending order list L according to their Id; then, in line 6, we concatenate the elements of L into s.

Example 2. For a graph dataset G consisting of g_1 and g_2 in Fig. 1, when the number of WL iterations is 1, we can build a corpus set $Cps = \{Cp_0, Cp_1\}$ for G, where $Cp_0 = (\Omega_0, S_0)$, $\Omega_0 = \{1, 2, 3\}$, $S_0 = \{\{1, 2, 3, 3\}, \{1, 2, 3, 3, 3\}\}$, and $Cp_1 = (\Omega_1, S_1)$, $\Omega_1 = \{4, 5, 6, 7, 8, 9\}$, $S_1 = \{\{4, 5, 6, 6\}, \{6, 6, 7, 8, 9\}\}$.

3.1.2. Learning vector representations using neural language models

Given the corpus set Cps generated by Algorithm 1. We use the following two neural language models realized in the Gensim library [30]: namely, CBOW model and Skip-gram model to learn d-dimensional vector representations of subtree patterns. For each corpus $Cp_i = (\Omega_i, S_i)$ in Cps, we can obtain a matrix Φ_i of size $|\Omega_i| \times d$ by training CBOW model or Skip-gram model on S_i. The j-th row of Φ_i is the vector representation of the j-th subtree pattern st_j in Ω_i, denoted as $\Phi_i(st_j)$. We can view the vector representation of a subtree pattern as a point in vector space. If two subtree patterns $st_j \in \Omega_i$ and $st_k \in \Omega_i$ are semantically similar, then their vector representations $\Phi_i(st_j)$ and $\Phi_i(st_k)$ will be located at near positions in vector space. Here semantic similarity of two subtree patterns means that they have similar contexts.

3.2. Clustering and merging semantically similar subtree patterns

Because the vector representations of semantically similar subtree patterns are mapped to near positions in vector space, we can use the Euclidean distance between subtree patterns' vector representations to measure the similarity of subtree patterns. Considering the similarity among subtree patterns, we cluster subtree patterns using AP clustering [12], and then we merge the subtree patterns in each cluster into a new feature.

3.2.1. Clustering subtree patterns

To guarantee good time performance, we divide subtree patterns in the vocabulary of a corpus into blocks, and then we apply AP clustering on each block. The time complexity of one AP clustering iteration is proportional to the square of the number of subtree patterns to be clustered. Since the total number of subtree patterns for a corpus is proportional to the number of vertices in the associated graph dataset, if we directly apply AP clustering to all subtree patterns without dividing the blocks, it would be too time consuming for a large graph dataset.

We divide and cluster subtree patterns as follows: given a corpus $Cp_i = (\Omega_i, S_i)$ of graph dataset G, we sort the subtree patterns in Ω_i in ascending order according to the Euclidean distance between their vector representations and the origin; then, we divide the sorted subtree patterns into blocks of size $B = \log^2 N$, where $N = |\Omega_i|$. We then apply AP clustering to each block to make similar subtree patterns in the same clusters. The time complexity of one AP clustering iteration on a subtree pattern block is $O(B^2)$; thus the total time complexity of clustering all subtree patterns in Ω_i is $B^2(N/B) = O(NB) = O(N \log^2 N)$, which is lower than the time complexity $O(N^2)$ of applying AP clustering to Ω_i, where N is the number of subtree patterns in Ω_i and N/B is the number of subtree pattern blocks.

In the following, we introduce the similarity matrix M and preference parameter *pref* used in AP clustering. Given a subtree pattern block of Ω_i, the similarity matrix M can be defined as a matrix of size $B \times B$, where $M_{j,k} = -\|\Phi_i(st_j) - \Phi_i(st_k)\|_2$. Here, st_j is the j-th subtree pattern in the block, $\Phi_i(st_j)$ is the vector representation of st_j, and $\|\Phi_i(st_j) - \Phi_i(st_k)\|_2$ is the Euclidean distance between $\Phi_i(st_j)$ and $\Phi_i(st_k)$.

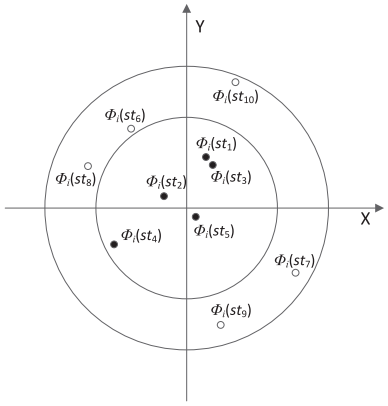


Fig. 2. Illustration of clustering subtree patterns.

In AP clustering, the parameter *pref* for a data point to be clustered is the preference degree that this data point is chosen as an exemplar. This parameter influences the number of clusters in AP clustering. Given the similarity matrix M of a subtree pattern block, to control the number of clusters in the AP clustering result, we define *pref* for all subtree patterns as follows:

$$pref = \min(M) + \beta(av(M) - \min(M))$$

where $0 \le \beta \le 1$, $\min(M)$ is the minimum value of M, and $av(M)$ is the average value of M.

Example 3. Given a graph dataset G, let $\Omega_i = \{st_1, st_2, \ldots, st_{10}\}$ be the vocabulary of the corpus Cp_i built for $\mathbb{F}_i(G)$. The 2-dimensional vector representations $\Phi_i(st_j)$ of st_j for $1 \le j \le 10$ are shown as points in Fig. 2. The clustering subtree patterns for block size $B = 5$ is as follows: We first sort subtree patterns in Ω_i in ascending order according to the Euclidean distance between their vector representations and the origin (0, 0) to obtain the sorted sequence $[st_5, st_2, st_3, st_1, st_4, st_6, st_8, st_9, st_7, st_{10}]$. We then divide the sorted sequence into 2 subtree pattern blocks of size B sequentially and then apply AP clustering to each block. The clustering result on the first block contains 4 clusters: $C_1 = \{st_1, st_3\}$, $C_2 = \{st_2\}$, $C_3 = \{st_4\}$, and $C_4 = \{st_5\}$. The clustering result on the second block contains 5 clusters: $C_5 = \{st_6\}$, $C_6 = \{st_7\}$, $C_7 = \{st_8\}$, $C_8 = \{st_9\}$, $C_9 = \{st_{10}\}$.

3.2.2. Merging semantically similar subtree patterns

We merge subtree patterns in each cluster into a new feature. Given two graphs g_1 and g_2, suppose that there are two subtree patterns st_1 and st_2 that meet the following conditions: (1) st_1 and st_2 are generated from g_1 and g_2, respectively, and (2) st_1 and st_2 locate in the same cluster. Since st_1 and st_2 are similar, st_1 and st_2 have some contributions to the similarity of g_1 and g_2. If we view subtree patterns as atomic structures, then the contribution of st_1 and st_2 to the similarity of g_1 and g_2 is zero. All subtree patterns in a cluster can be treated as the same feature to some degree; thus we merge subtree patterns in a cluster into a new feature to improve the graph classification accuracy. Suppose that there are N' clusters in AP clustering $\{C_1, C_2, \ldots, C_{N'}\}$; then the total number of merged features is N', where the ith merged feature is a summary of subtree patterns in the ith cluster C_i. The occurrence of the ith merged feature in a graph g is $\sum_{st \in C_i} cnt(st, F_t(g))$.

Algorithm 2 presents the procedure for merging subtree patterns, where G is a graph dataset, $\mathbb{F}_i(G)$ is the i-hop feature set of G, B is the size of subtree pattern blocks, β is a clustering parameter and fn is the initial Id of merged features. The workflow of Algorithm 2 is as follows: we call *buildMR*() in line 1 to obtain the merging rules of subtree patterns in $\mathbb{F}_i(G)$, and we call *merge*() in line 2 to merge subtree patterns in $\mathbb{F}_i(G)$.

Algorithm 2: *FeatureMerge*$(\mathbb{F}_i(G), B, \beta, fn)$.

1 $R \leftarrow$ *buildMR*$(\mathbb{F}_i(G), fn)$
2 $\mathbb{F}_i'(G) \leftarrow$ *merge*$(\mathbb{F}_i(G), R)$
3 **return** $\mathbb{F}_i'(G)$

procedure *buildMR*$(\mathbb{F}_i(G), fn)$
1 $R \leftarrow \varnothing$
2 $Cp_i \leftarrow$ *buildCp*$(\mathbb{F}_i(G))$
3 learn Φ_i for Cp_i using neural language models
4 sort $st_j \in \Omega_i$ to form the ascending orderlist L according to the distance between $\Phi_i(st_j)$ and the origin
5 divide L into blocks of size B
6 **for** each block b **do**
7 compute similarity matrix M of block b
8 $pref \leftarrow \min(M) + \beta \times (av(M) - \min(M))$
9 $Cs \leftarrow APClustering(b, M, pref)$
10 **for** each cluster c in Cs **do**
11 $r \leftarrow (c, fn)$
12 $R \leftarrow R \cup \{r\}$
13 $fn \leftarrow fn + 1$
14 **return** R

procedure *merge*$(\mathbb{F}_i(G), R)$
1 $\mathbb{F}_i'(G) \leftarrow \varnothing$
2 **for** each graph $f_i(g)$ in $\mathbb{F}_i(G)$ **do**
3 $f_i'(g) \leftarrow \varnothing$
4 **for** each merging rule r in R **do**
5 insert $\sum_{st \in r.c} cnt(st, f_i(g))$features with Id $r.fn$ into $f_i'(g)$
6 $\mathbb{F}_i'(G) \leftarrow \mathbb{F}_i'(G) \cup \{f_i'(g)\}$
7 **return** $\mathbb{F}_i'(G)$

	st_1	st_2	st_3	st_4	st_5	st_6	st_7	st_8	st_9	st_{10}
g_1	3	0	2	3	0	0	1	0	4	1
g_2	1	1	3	1	1	3	0	1	1	2
g_3	4	1	3	2	3	3	2	1	2	3

(a) Feature vectors of g_1, g_2, and g_3

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9
g_1	5	0	3	0	0	1	0	4	1
g_2	4	1	1	1	3	0	1	1	2
g_3	7	1	2	3	3	2	1	2	3

(b)Features vectors after subtree patterns merging

Fig. 3. Illustration of merging subtree patterns.

In procedure *buildMR*, we obtain the set of merging rules of subtree patterns for a given feature set $\mathbb{F}_i(G)$. we divide distinct subtree patterns occurring in $\mathbb{F}_i(G)$ into blocks in lines 2–5. For each iteration of lines 6–13, we obtain the merging rules of subtree patterns in each block, where a merging rule (c, fn) means that subtree patterns in the cluster c are merged into a new feature with Id fn.

In procedure *merge*, we convert $\mathbb{F}_i(G)$ into a feature set $\mathbb{F}_i'(G)$ by merging subtree patterns in $f_i(g)$ for $f_i(g) \in \mathbb{F}_i(G)$. In the iteration of lines 4–5, we obtain $f_i'(g)$ by merging subtree patterns in $f_i(g)$ according to merging rules R.

Example 4. Given the subtree pattern clustering result $\{C_1, C_2, \ldots, C_9\}$ shown in Example 3 and the occurrence of subtree patterns in g_1, g_2 and g_3 shown in Fig. 3(a). The merging of subtree patterns is described as follows: the subtree patterns in each cluster are merged into a feature, where subtree patterns in C_i are merged into a feature denoted as f_i for $1 \le i \le 9$; the number of occurrences of f_i in a graph is the sum of the occurrences of subtree patterns in C_i occurring in the graph. The feature vectors of these graphs after merging subtree patterns is shown in Fig. 3(b). For example, since $C_1 = \{st_1, st_3\}$ and the occurrences of st_1 and st_3 in g_1 are 3 and 2, respectively, so the number of occurrence of feature f_1 in g_1 is 5.

For each feature set $\mathbb{F}_i(G) \in \mathcal{F}_t(G)$, we convert $\mathbb{F}_i(G)$ into a new feature set $\mathbb{F}_i'(G)$ by merging similar subtree patterns using Algorithm 2. Thus, we can convert $\mathcal{F}_t(G)$ into a new feature set $\mathcal{F}_t'(G)$, where $\mathcal{F}_t'(G) = \{\mathbb{F}_i'(G) \mid 0 \le i \le t\}$, $\mathbb{F}_i'(G) = \{f_i'(g) \mid g \in G\}$ and $f_i'(g)$ is the merged feature multiset of $f_i(g)$.

3.2.3. Influence of merging similar subtree patterns on graph classification

After we apply AP to subtree patterns, each cluster consists of similar subtree patterns. Whether two subtree patterns are in the same cluster depends on the similarity of their vector representations learned from neural language models. We merge subtree patterns in a cluster into a feature. Merging similar subtree patterns could increase the separability of a graph data set.

Under the RBF kernel [6] metrics, the similarity of the two graphs g and q is defined as

$$\mathrm{RBF}(V, Q) = \mathrm{e}^{-\frac{\|V-Q\|^2}{2\sigma^2}}$$

where $V = (V_1, V_2, \ldots, V_d)$ and $Q = (Q_1, Q_2, \ldots, Q_d)$ are d-dimensional feature vectors of g and q, and $\|V - Q\|^2 = \sum_{k=1}^{d} (V_k - Q_k)^2$, and σ is a free parameter.

Suppose there exists a cluster composed of subtree patterns st_i and st_j. We merge st_i and st_j into a feature. V and Q are transformed into $(d-1)$-dimensional feature vectors V' and Q', respectively, where

$$V_k' = \begin{cases} V_k & \text{if } 1 \le k < i \text{ or } i+1 \le k < j; \\ V_i + V_j & \text{if } k = i; \\ V_{k+1} & \text{if } j \le k \le d-1. \end{cases}$$

Similarly, we can define Q_k' as that of V_k'. Then, we have

$$\begin{aligned} \|V' - Q'\|^2 &= \sum_{k=1}^{d-1} (V_k' - Q_k')^2 \\ &= \sum_{k=1 \wedge k \neq i}^{d-1} (V_k' - Q_k')^2 + (V_i' - Q_i')^2 \\ &= \sum_{k=1 \wedge k \neq i}^{d-1} (V_k' - Q_k')^2 + ((V_i + V_j) - (Q_i + Q_j))^2 \\ &= \sum_{k=1 \wedge k \neq i}^{d-1} (V_k' - Q_k')^2 + (V_i - Q_i)^2 + (V_j - Q_j)^2 \\ &\quad + 2(V_i - Q_i)(V_j - Q_j) \\ &= \sum_{k=1}^{d} (V_k - Q_k)^2 + 2(V_i - Q_i)(V_j - Q_j) \\ &= \|V - Q\|^2 + 2(V_i - Q_i)(V_j - Q_j) \end{aligned}$$

After merging, the similarity of graphs g and q is

$$\begin{aligned} \mathrm{RBF}(V', Q') &= \mathrm{e}^{-\frac{\|V'-Q'\|^2}{2\sigma^2}} \\ &= \mathrm{e}^{-\frac{\|V-Q\|^2 + 2(V_i-Q_i)(V_j-Q_j)}{2\sigma^2}} \\ &= \mathrm{e}^{-\frac{\|V-Q\|^2}{2\sigma^2}} \times \mathrm{e}^{-\frac{(V_i-Q_i)(V_j-Q_j)}{\sigma^2}} \\ &= \mathrm{RBF}(V, Q) \times \mathrm{e}^{-\frac{(V_i-Q_i)(V_j-Q_j)}{\sigma^2}} \end{aligned}$$

The change of the similarity of graphs g and q after merging is determined by the occurrences of st_i and st_j in V and Q.

If $(V_i > Q_i \wedge V_j < Q_j)$ or $(V_i < Q_i \wedge V_j > Q_j)$, then $e^{-\frac{(V_i-Q_i)(V_j-Q_j)}{\sigma^2}} > 1$. So the similarity of g and q increases after merging, which is consistent with the fact that subtree patterns merging can reduce the difference of g and q on occurrences of st_i and st_j in this case.

If $V_i = Q_i$ or $V_j = Q_j$, then $e^{-\frac{(V_i-Q_i)(V_j-Q_j)}{\sigma^2}} = 1$. So the similarity of g and q does not change after merging, which is consistent with the fact that subtree patterns merging has no impact on the difference of g and q on occurrences of st_i and st_j in this case.

If $(V_i > Q_i \wedge V_j > Q_j)$ or $(V_i < Q_i \wedge V_j < Q_j)$, then $e^{-\frac{(V_i-Q_i)(V_j-Q_j)}{\sigma^2}} < 1$. So the similarity of g and q decreases after merging, which is consistent with the fact that subtree patterns merging can enlarge the difference of g and q on occurrences of st_i and st_j in this case.

Thus it can be seen that the feature vectors of graphs after merging are more accurate for representing graphs. This could increase the separability among graphs and result in an improvement on graph classification accuracy.

We use an example below to discuss the influence of occurring frequency of st_i and st_j on distance changes among graphs after subtree patterns merging.

Example 5. Fig. 3 gives an example of merging subtree patterns. Fig. 3(a) shows the 10-dimensional feature vectors of graphs g_1, g_2 and g_3. Fig. 3(b) shows the resulting 9-dimensional feature vectors after subtree pattern merging, where st_1 and st_3 are merged into a feature f_1.

As can be seen in Fig. 3, before merging st_1 and st_3, the similarity of g_1 and g_2 is $e^{-\frac{16}{\sigma^2}}$, the similarity of g_1 and g_3 is $e^{-\frac{16}{\sigma^2}}$, and the similarity of g_2 and g_3 is $e^{-\frac{10}{\sigma^2}}$. After merging st_1 and st_3, the similarity of g_1 and g_2 is $e^{-\frac{14}{\sigma^2}}$, which is larger than that before merging. The similarity of g_1 and g_3 is $e^{-\frac{17}{\sigma^2}}$, which is smaller than that before merging. The similarity of g_2 and g_3 is $e^{-\frac{10}{\sigma^2}}$, which is the same as that before merging.

From the changes of similarity among g_1, g_2, and g_3, we can see that the feature vectors after merging are more accurate for representing these graphs.

3.3. Selecting highly discriminative features

To improve the graph classification accuracy, we need to select highly discriminative features for graph classification. In graph classification, not all the features have equivalent contributions to the graph classification. Some features have similar occurrence in each class of the graphs. Since these features are poorly discriminative, which would affect the graph classification accuracy, we treat them as noises and filter them out.

Definition 5 (Highly Discriminative Feature)**.** Given a training graph dataset G and its label set Y, where Y has l distinct class labels in total and the set of graphs with the ith label is denoted as G_i, a highly discriminative feature of G is a feature occurring disproportionately in different classes of graphs; that is, a highly discriminative feature occurs frequently in one class of graphs and occurs infrequently in the remaining classes of graphs. Given a feature ft, the discrimination of ft can be measured by its discrimination measure $score(ft)$; the greater the discrimination score is, the more discriminative ft will be, where $score(ft)$ can be defined as follows:

$$score(ft) = \max_{1 \le i \le l} \left\{ \text{abs}\left(\frac{CNT(ft,\ \mathcal{F}'_t(G_i))}{|G_i|} - \frac{CNT(ft,\ \mathcal{F}'_t(G - G_i))}{|G - G_i|} \right) \right\}$$

With the discrimination score of each distinct feature occurring in $\mathcal{F}'_t(G)$, we can obtain highly discriminative feature set of G via the following steps. (1) Let U be the set of distinct features occurring in $\mathcal{F}'_t(G)$, and we sort the elements of U in descending order according to their discrimination scores. (2) For a specified parameter α, $0 < \alpha \le 1$, we select the top $\alpha|U|$ features from the sorted U, denoted as D. 3)We update $\mathcal{F}'_t(G)$ by deleting such features ft from $f'_i(g)$ that $ft \in f'_i(g)$ but $ft \notin D$ for $g \in G$ and $0 \le i \le t$. Finally, the updated feature set $\mathcal{F}'_t(G)$ only includes highly discriminative features.

After merging semantically similar subtree patterns and selecting highly discriminative features, we can represent G as a feature matrix W of size $n \times |D|$, where $W_{j,k} = \sum_{i=0}^{t} cnt(D_k, f'_i(g_j))$. Here, n is the number of graphs in G, D is the set of all distinct features occurring in $\mathcal{F}'_t(G)$, and $cnt(D_k, f'_i(g_j))$ is the occurrence of feature D_k in the updated $f'_i(g_j)$. The jth row of W is denoted as W_j, which is the feature vector of graph g_j. The kernel matrix K of G can be defined as a matrix of $n \times n$, where $K_{j,k} = h(W_j, W_k)$ is the kernel of g_j and g_k.

3.4. Whole algorithm

Given a training graph dataset G and its label set Y, we provide the whole algorithm for learning a graph classifier FRS_KELM in this section. FRS_KELM consists of 4 components: W, R, D and $CKelm$, where W is the feature matrix of G, R is the set of subtree pattern merging rules, D is the set of selected highly discriminative features, and $CKelm$ is a KELM classifier. KELM [16] uses a kernel function to substitute the hidden layer of ELM, and KELM has no need for tuning the weights between the input layer and the hidden layer. In $CKelm$, the number of hidden nodes is $|G|$ and the kernel function h can be the linear kernel, RBF kernel or polynomial kernel, among others.

Algorithm 3 presents the procedure for learning FRS_KELM and predicting class labels for graphs, where T is a set of graphs without class labels, B is the block size of subtree patterns, β is a parameter used to compute clustering preference in AP clustering, and α is the ratio of selected highly discriminative features. The workflow of Algorithm 3 is as follows: we call *train* in line 1 to train a graph classifier FRS_KELM; then, for each graph q in T, we call *predict* to predict the class label of q in lines 3–4.

In the *train* procedure, we learn the FRS_KELM graph classifier using training dataset G and its label set Y. In line 2, we obtain feature set $\mathcal{F}_t(G)$ for G using WL subtree iterations. We then convert $\mathcal{F}_t(G)$ into a merged feature set $\mathcal{F}'_t(G)$. In each iteration of the **for** loop in line 3, we convert a feature set $\mathbb{F}_i(G)$ into $\mathbb{F}'_i(G)$ by merging subtree patterns according to merging rules mr. In lines 8–11, we update feature set $\mathcal{F}'_t(G)$ by deleting features that are not highly discriminative features. In lines 12–14, we train a KELM classifier $CKelm$ using the kernel matrix K of G and the label set Y. The final FRS_KELM consists of four components: the feature matrix W, the set of subtree pattern merging rules R, the set of selected highly discriminative features D and the KELM classifier $CKelm$ in line 15.

In the *predict* procedure, we predict the class label of a graph q. We obtain the feature multiset $F_t(q)$ of q using WL subtree iterations in line 1. We then convert $F_t(q)$ into a new feature multiset $F'_t(q)$ by merging subtree patterns according to FRS_KELM.R and deleting features not belonging to FRS_KELM.D, respectively, in lines 2–3. Then, we compute feature vector V and kernel vector KV for q. V is a vector of size $|\text{FRS\_KELM}.D|$, where $V_i = cnt(\text{FRS\_KELM}.D_i, F'_t(q))$; KV is a vector of size $|G|$, where $KV_j = h(V, \text{FRS\_KELM}.W_j)$ is the kernel of q and g_j, in lines 4–5. We obtain the class label of q using FRS_KELM.$CKelm$ and KV in line 6.

Algorithm 3: *ClassifyGraph*$(G, Y, T, t, B, \beta, \alpha)$.

FRS_KELM $\leftarrow$ $train(G, Y, t, B, \beta)$
$labels \leftarrow \varnothing$
for each graph q in T **do**
  $lb \leftarrow predict(\text{FRS\_KELM}, q, t)$
  $labels \leftarrow labels \cup \{lb\}$
return *labels*

procedure $train(G, Y, t, B, \beta)$
$\mathcal{F}'_t(G) \leftarrow \varnothing$, $R \leftarrow \varnothing$
$\mathcal{F}_t(G) \leftarrow$ the feature set generated by t WL subtree iterations on G
for each feature set $\mathbb{F}_i(G)$ in $\mathcal{F}_t(G)$ **do**
  $mr \leftarrow buildMR(\mathbb{F}_i(G), |R|)$
  $\mathbb{F}'_i(G) \leftarrow merge(\mathbb{F}_i(G), mr)$
  $\mathcal{F}'_t(G) \leftarrow \mathcal{F}'_t(G) \cup \{\mathbb{F}'_i(G)\}$
  $R \leftarrow R \cup mr$
$U \leftarrow$ all distinct features in $\mathcal{F}'_t(G)$
sort u_i $(u_i \in U)$ in descending order according to $score(u_i)$
$D \leftarrow$ the top $\alpha|U|$ features of the ordered u_i
update $\mathcal{F}'_t(G)$ by deleting features not in D from $\mathcal{F}'_t(G)$
generate feature matrix W of G
generate kernel matrix K of G
train a KELM *CKelm* using K and Y
FRS_KELM $\leftarrow$ $(W, R, D, CKelm)$
return FRS_KELM

procedure $predict(\text{FRS\_KELM}, q, t)$
$F_t(q) \leftarrow$ the feature multiset generated by t WL subtree iterations on q
$F'_t(q) \leftarrow$ merge subtree patterns in $F_t(q)$ according to FRS_KELM.R
update $F'_t(q)$ by deleting features not in FRS_KELM.D from $F'_t(q)$
generate feature vector V for q
compute kernel vector KV for q
use FRS_KELM.*CKelm* and KV to predict the label lb of q
return lb

4. Experiments

4.1. Dataset

To test the efficacy of the proposed graph classification method, in the experiments, we use 4 benchmark datasets: NCI1, Mutag, Enzymes and PTC, which are described in detail in the following.

NCI1 [36] is a dataset of chemical compounds screened for activity against non-small-cell lung cancer;

The Mutag [9] dataset includes 188 aromatic and heteroaromatic nitro molecular structures; they are classified according to whether they have a mutagenic effect on salmonella typhimurium.

Enzymes [2] is a dataset of tertiary structures of 600 enzymes chosen from the BRENDA database, and they are categorized into 6 classes according to their function: EC1, EC2, ..., EC6.

PTC [35] is a dataset recording the carcinogenicity of compounds; according to object species, PTC is further categorized into 4 datasets: PTC_FR, PTC_MR, PTC_FM and PTC_MM.

The general statistical information of each dataset is shown in Table 1, where $|G|$ denotes the number of graphs in a dataset; $|AV|$ and $|AE|$ denote the average numbers of vertices and edges in a dataset, respectively; $|C|$ denotes the number of graph class labels in a dataset; $|P|$ and $|N|$ denote the numbers of positive and negative graphs in a dataset, respectively; and '–' indicates that the dataset does not have the statistical characteristic.

Table 1
Statistics information of graph datasets.

Dataset	$\|G\|$	$\|AV\|$	$\|AE\|$	$\|C\|$	$\|P\|$	$\|N\|$
NCI1	4110	29.9	32.3	2	2057	2053
Mutag	188	17.7	38.9	2	125	63
Enzymes	600	32.6	61.1	6	–	–
PTC_MM	192	25.8	26.2	2	69	123
PTC_MR	196	26.6	27.1	2	70	126
PTC_FM	204	26.0	26.5	2	80	124
PTC_FR	204	26.4	26.9	2	63	141

All the experiments are conducted on an HP Z400 PC, with a 2.39 GHz CPU and 12 GB memory.

4.2. Parameter evaluation

In the experiments, when generating feature set $\mathcal{F}_t(G)$ for G, we set the total number of WL subtree iterations $t = 5$. We use CBOW or Skip-gram models to learn vector representations of subtree patterns, where the context length is 5 or 10 and the dimension of the vector representation is $d = 10$. During training of the KELM classifier, the kernel function adopts the RBF kernel, parameter C is chosen from $\{2^i\}$ for $2 \le i \le 8$, and parameter σ is chosen from $\{2^i\}$ for $-4 \le i \le 8$.

In the following work, we study the effect of subtree pattern block size B, clustering parameter β and discriminative feature selection ratio α on the graph classification accuracy, where B is chosen from {1000, 2000, 3000}, β is chosen from $\{0, 0.1, 0.2, \ldots, 1\}$, and α is chosen from {0.1, 0.2, 0.3, 0.4}. For each parameter combination, we perform 10-fold cross-validation on each dataset. We repeat the experiments 10 times and report the average accuracy.

Figs. 4–7 present the graph classification accuracy of the proposed method on PTC datasets with varying parameters B, β and α. The experiments show the following. (1) In general, for different combinations of B and β, the classification accuracy for $\alpha = 0.2$ or 0.3 is higher than that for $\alpha = 0.1$ or 0.4. This result occurs because an α that is too small in discriminative feature selection would lose some highly discriminative features; in contrast, an α that is too large would select some poorly discriminative features. (2) For different combinations of β and α, with the increase in block size B, the classification accuracy would have a significant improvement. This result occurs because a large B would be capable of eliminating locality in clustering subtree patterns.

Fig. 8 presents the classification accuracy of the proposed method on Mutag. Because the number of distinct subtree patterns occurring in $\mathbb{F}_i(G)$ for $0 \le i \le 5$ is less than 2000 for Mutag, we only set $B = 1000$ or 2000. This figure shows that the classification accuracy for $B = 2000$ is higher than that for $B = 1000$.

Fig. 9 presents the classification accuracy of the proposed method on NCI1. This figure shows that (1) with varying α, the classification accuracy for $B = 2000$ is higher than that for $B = 1000$ or 3000; (2) given fixed B and β, the classification improves with increasing α.

Fig. 10 presents the classification accuracy of the proposed method on Enzymes. This figure shows that (1) with varying α, the classification accuracy for $B = 3000$ is higher than that for $B = 1000$ or 2000; (2) given fixed B and β, the classification improves with increasing α.

The above experiments show that by adjusting parameters B, β, and α, a high classification accuracy can always be obtained on each dataset. The parameter B affects the locality problem in clustering subtree patterns. In general, a large B is capable of

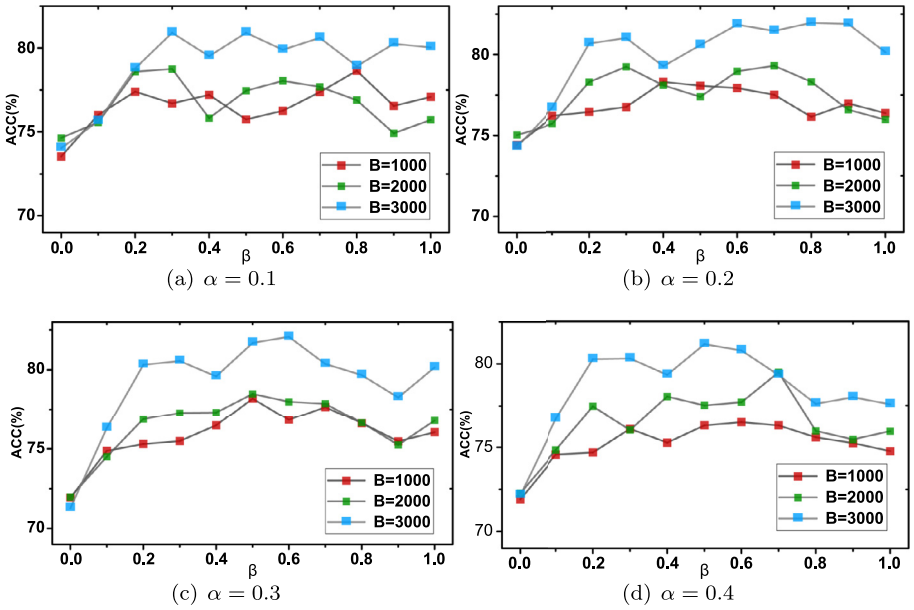


Fig. 4. Average classification accuracy with varying parameters on PTC_FR.

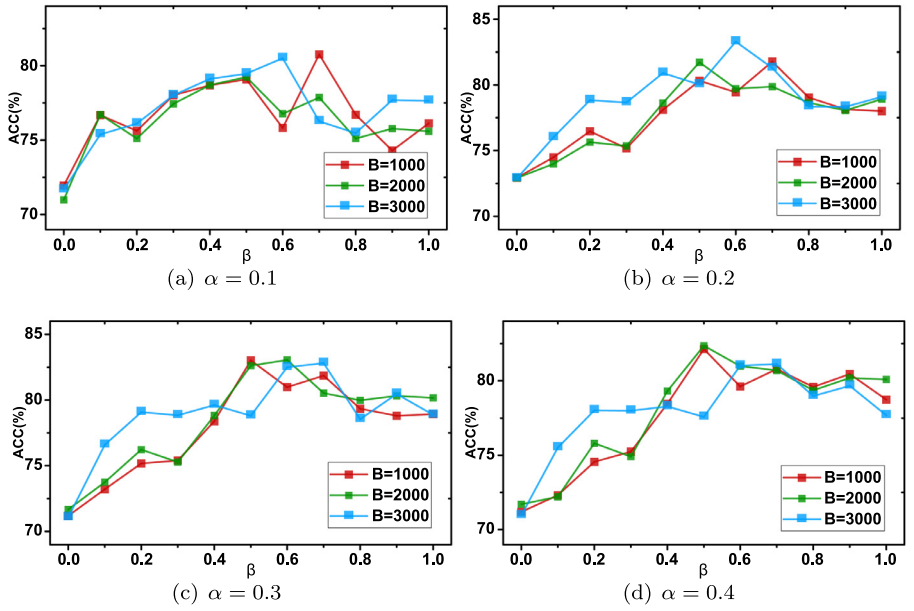


Fig. 5. Average classification accuracy with varying parameters on PTC_FM.

eliminating the locality problem such that the classification accuracy is improved; however, a large B will increase the time consumed for clustering subtree patterns. To achieve a tradeoff between classification accuracy and time performance, we set B as 1000, 2000 or 3000. A small β will increase the number of clusters in subtree pattern clustering, thus some similar subtree patterns cannot be clustered into the same cluster; in contrast, a large β will place dissimilar subtree patterns into a cluster. Because the distribution of vector representations of subtree patterns depends on each graph dataset, we tune β to a value between 0 and 1 for each dataset. The parameter α determines the number of selected highly discriminative features. If α is too small, then the selected features would lose some highly discriminative features; if α is too large, then the selected features would include some poorly discriminative features. Thus, we set α as a value between 0.1 and 0.4 for each dataset.

4.3. The selection of clustering methods

In this section we compare the graph classification accuracy of FRS_KELM with different clustering methods in merging similar subtree patterns. DBSCAN [11] is a density-based clustering method, it views clusters as areas of high density separated by areas of low density. BIRCH [40] is a hierarchical clustering method. It builds a Clustering Feature Tree (CF Tree) for the given data. Then a agglomerative hierarchical clustering algorithm is applied on CF Tree nodes to obtain the final result. In the experiments, we use DBSCAN and BIRCH to replace AP clustering in FRS_KELM, and get FRS_KELM_DBSCAN and FRS_KELM_BIRCH, respectively. Table 2 shows the comparison result of FRS_KELM with different clustering methods.

Table 2 shows that generally speaking, when applying AP to cluster subtree patterns, we can obtain better performance on

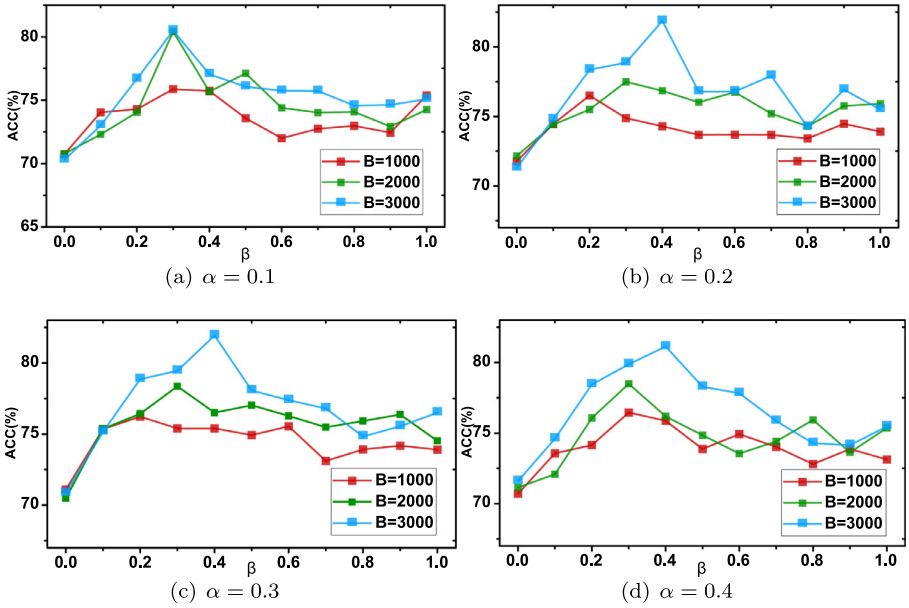


Fig. 6. Average classification accuracy with varying parameters on PTC_MM.

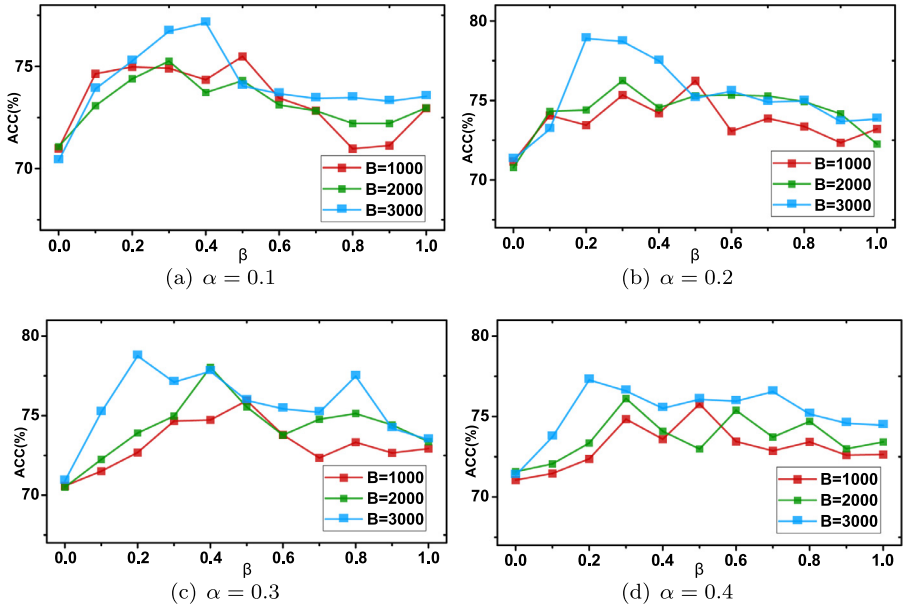


Fig. 7. Average classification accuracy with varying parameters on PTC_MR.

Table 2
Comparison of graph classification accuracy with different clustering methods.

Dataset	FRS_KELM	FRS_KELM_DBSCAN	FRS_KELM_BIRCH
PTC_MM	**81.57 ± 1.84**	65.92 ± 1.91	66.15 ± 1.64
PTC_MR	**79.66 ± 1.36**	65.92 ± 1.91	69.15 ± 1.55
PTC_FR	**81.72 ± 1.28**	70.11 ± 1.11	71.40 ± 1.39
PTC_FM	**82.89 ± 1.28**	62.55 ± 2.51	62.68 ± 1.10
Mutag	91.47 ± 1.02	93.19 ± 0.76	**93.47 ± 0.91**
Enzymes	**65.45 ± 0.91**	57.33 ± 0.82	57.03 ± 1.24

graph classification accuracy over that applying DBSCAN or BIRCH. Specifically, FRS_KELM has a significant advantage on classification accuracy over FRS_KELM_DBSCAN and FRS_KELM_BIRCH on all but Mutag dataset. All of the three methods have similar classification accuracy on Mutag dataset. The main reason could be that the distribution of vector representations of subtree patterns on Mutag is not sensitive to the selected clustering methods.

4.4. Comparison with other methods

In this section, we compare the classification accuracy of the proposed method FRS_KELM with the following graph kernel-based and neural language model-based methods:

WL Kernel [31]: using WL subtree kernel matrix and SVM to classify graphs;

Deep WL [38]: using Deep WL subtree kernel matrix and SVM to classify graphs;

GC_LASSO_ELM [39]: using LASSO to reduce the dimension of the WL subtree kernel matrix and ELM to classify graphs;

GSR_GK_KDA_ELM [21]: using graph set reconstruction and KDA to reduce the feature dimension of the WL shortest path kernel matrix, and then using ELM to classify graphs; GSR_GK_KDA_ELM only applies to graph datasets with two classes.

FRS_SVM: using our method to merge and select features and SVM to classify graphs;

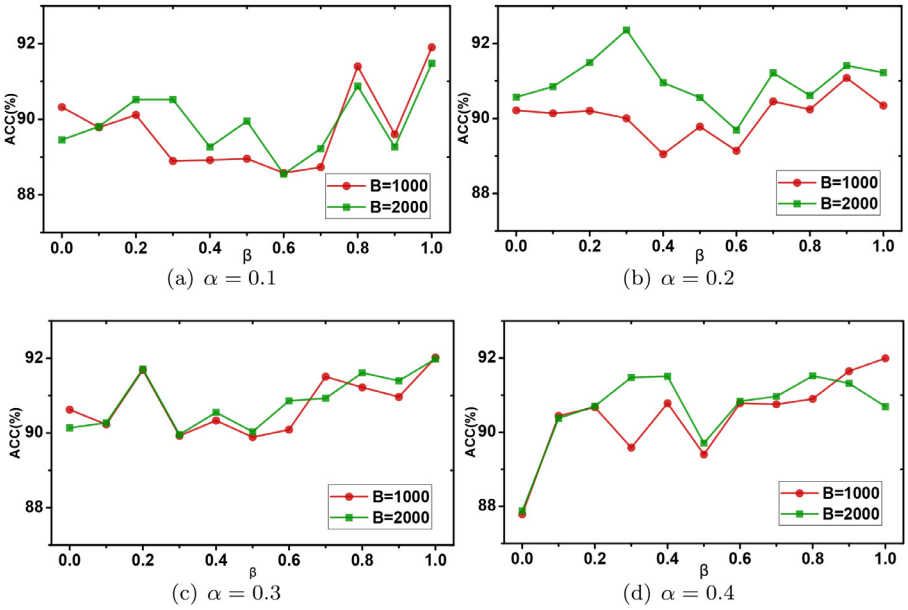


Fig. 8. Average classification accuracy with varying parameters on Mutag.

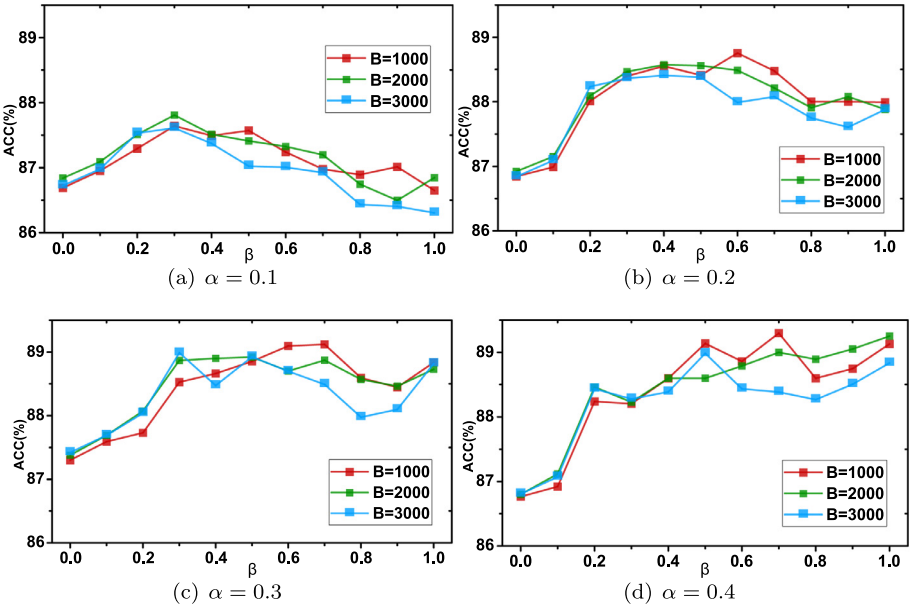


Fig. 9. Average classification accuracy with varying parameters on NCI1.

node2vec [13] uses CBOW model to learn vector representations of nodes in a graph; we refer to the average value of all nodes' vector representations as the graph's vector representation by Taheri et al. [34] and KELM to classify graphs;

graph2vec [24] uses DBOW model to learn the vector representation of an entire graph and KELM to classify graphs.

For the WL kernel, the number of WL subtree iterations is set as $h = \{2, 3, 4, 5\}$, and the best result on each dataset is given. For GC_LASSO_ELM, the number of WL subtree iterations is set as its default value of 5. For the Deep WL kernel, the number of WL subtree iterations on Mutag, PTC and Enzymes is set as $h = \{2, 3, 4, 5\}$; the number of WL subtree iterations on NCI1 is set as $h = \{2, 3\}$ because when $h > 3$, Deep WL kernel throws a memory error. For GSR_GK_KDA_ELM, the number of WL shortest path iterations on Mutag, PTC and Enzymes is set as its default value of 5; the number of WL shortest path iterations on NCI1 is set as $h = 2$ because when $h > 2$, GSR_GK_KDA_ELM throws a memory error. For FRS_KELM and FRS_SVM, the number of WL subtree iterations is set as 5. The other parameters of each method are set as their default values. Standard 10-fold cross-validation is used to obtain the graph classification accuracy of each method. For SVM-based classification methods, the parameter C for each fold is independently tuned using training data from that fold. The experiments are repeated 10 times. The comparison of the average classification accuracy and standard deviation of each method is shown in Table 3.

Table 3 shows that the graph classification accuracy of the proposed methods FRS_KELM and FRS_SVM are better than that of the other compared methods. FRS_KELM and FRS_SVM have improvements of at least 5% in classification accuracy over the compared methods. The improvement of FRS_KELM can be explained as follows. (1) By merging similar subtree patterns in each cluster into a new feature, FRS_KELM could be capable of measuring the similarity among graphs more accurately. (2) By

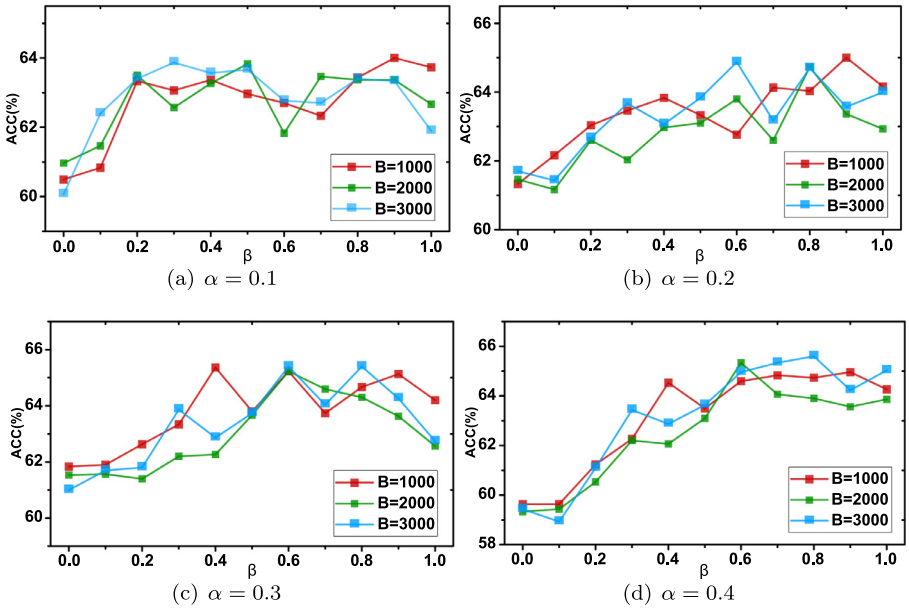


Fig. 10. Average classification accuracy with varying parameters on Enzymes.

Table 3
Comparison of graph classification accuracy.

Dataset	PTC_MM	PTC_MR	PTC_FR	PTC_FM	Mutag	NCI1	Enzymes
FRS_KELM	**81.57 ± 1.84**	**79.66 ± 1.36**	**81.72 ± 1.20**	**82.89 ± 1.28**	**91.49 ± 1.02**	**89.18 ± 0.14**	**65.45 ± 0.91**
FRS_SVM	80.05 ± 1.71	78.63 ± 2.72	78.60 ± 2.25	79.60 ± 3.02	88.78 ± 1.43	88.77 ± 0.14	60.13 ± 1.22
Deep WL	69.21 ± 2.00	63.53 ± 1.34	69.50 ± 2.87	63.65 ± 2.41	83.83 ± 1.32	84.37 ± 0.23	55.31 ± 0.80
WL kernel	67.21 ± 1.27	63.42 ± 2.17	67.95 ± 2.13	62.25 ± 2.47	83.61 ± 2.02	84.67 ± 0.26	53.88 ± 0.95
GC_LE	71.22 ± 1.89	69.39 ± 2.37	72.04 ± 1.63	68.71 ± 3.022	87.56 ± 1.56	81.29 ± 0.3	46.68 ± 0.87
GSR_GKE	73.84 ± 2.31	71.84 ± 1.86	71.25 ± 2.32	70.60 ± 2.81	86.50 ± 1.59	79.16 ± 0.24	–
node2vec	67.37 ± 0.71	64.36 ± 0.21	69.16 ± 0.26	67.65 ± 0.93	84.44 ± 0.39	62.47 ± 0.23	22.27 ± 0.90
graph2vec	70.78 ± 1.72	64.35 ± 0.30	70.71 ± 1.36	70.11 ± 0.78	86.17 ± 1.27	84.88 ± 0.22	31.03 ± 1.14

* Where GC_LE stands for GC_LASSO_ELM. GSR_GKE stands for GSR_GK_KDA_ELM.

Table 4
Comparison of time performance for graph classification (s).

Dataset	PTC_MM	PTC_MR	PTC_FR	PTC_FM	Mutag	NCI1	Enzymes
FRS_KELM	33.04	87.07	64.04	80.05	18.13	424.45	403.50
FRS_SVM	33.59	49.35	99.47	44.49	11.44	704.28	92.32
Deep WL	4.49	2.66	2.35	4.55	4.42	986.22	48.72
WL kernel	**3.04**	**2.60**	**3.16**	**3.29**	**2.43**	**211.46**	**13.82**
GC_LE	3.45	3.00	3.44	4.29	2.456	482.51	27.95
GSR_GKE	19.38	15.07	25.21	25.34	1392.80	5134.77	–

* Where GC_LE stands for GC_LASSO_ELM. GSR_GKE stands for GSR_GK_KDA_ELM.

selecting highly discriminative features for graph classification, FRS_KELM can filter out poorly discriminative features, which would be noises. By comparison, the WL kernel treats subtree patterns as atomic structures and uses all subtree patterns to compute the kernel matrix. Although the Deep WL kernel takes the similarity among subtree patterns into account, it uses all subtree patterns to compute the kernel matrix. GC_LASSO_ELM and GSR_GK_KDA_ELM reduce the dimensions of features, but they do not consider the similarity among substructures. The random walks used in node2vec are linear substructures while the subtree patterns are non-linear substructures. graph2vec uses all subtree patterns to a given hop to learn representation vector of a graph, it does not select discriminative features. The classification accuracy of FRS_KELM is better than that of FRS_SVM, which means that when using KELM to classify graphs, we can obtain additional accuracy improvements compared to using SVM.

In Table 4, we present the comparison of the time performance of the compared methods. For each compared method, we show the total time of computing the feature matrix or kernel matrix, reducing or selecting features and one 10-fold cross-validation for graph classification. Note that the total time of GSR_GK_KDA_ELM also includes the time consumed for reconstructing graph datasets. On the small datasets, Mutag, Enzymes and PTC, the time performance of FRS_KELM is worse than that of WL kernel, Deep WL and GC_LASSO_ELM, and it is slightly worse than that of GSR_GK_KDA_ELM. This result occurs because the proposed method includes extra AP clustering on subtree patterns, whose time complexity of one iteration is $O(NB)$, where B is the block size and N is the number of subtree patterns to be clustered. This improves the classification accuracy at the cost of extra time consumption. One solution to this problem is parallel processing because the clusterings on subtree pattern blocks are independent tasks that have no need of information interaction during clustering. On the large dataset, NCI1, the time performance of FRS_KELM is better than that of the Deep WL kernel and GSR_GK_KDA_ELM. The main reason can be explained as follows. In Deep WL kernel, the time complexity of computing the kernel matrix is $O(N^4)$. In GSR_GK_KDA_ELM, the time complexity of kernel discriminant analysis is $O(n^3)$, where n is the number of graphs in the training dataset.

5. Conclusion and discussion

In this paper, we study the problem of graph feature reduction based on semantic similarity for graph classification. Rather than taking substructures in graphs as atomic structures, we study the similarity among subtree patterns in graphs using neural language

models and merge similar subtree patterns into a new feature. Considering the problem that poorly discriminative features affect graph classification, we provide a new feature discrimination score to select highly discriminative features for graph classification. The experiments show that the proposed method significantly improves the graph classification accuracy. In the future, we will attempt to apply our feature reduction method in convolutional neural networks for graph classification and then decrease the gap between feature selection and training graph classifier. Furthermore, regarding the time complexity of clustering subtree pattern blocks, we will attempt to design a more effective subtree pattern blocking and merging method.

Author contribution

Initial idea of the research was from ZS. ZS, HH, JH and JSV designed the proposed algorithm. ZS and HH implemented the proposed algorithm and carried out the experiments. All authors participated in analysis and manuscript preparation. All authors read and approved the final version of the manuscript.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Zhigang Sun: Conceptualization, Data curation, Formal analysis, Methodology, Software, Validation, Writing - review & editing. **Hongwei Huo:** Conceptualization, Data curation, Formal analysis, Funding acquisition, Methodology, Supervision, Software, Validation, Writing - review & editing. **Jun Huan:** Conceptualization, Data curation, Formal analysis, Methodology, Validation, Writing - review & editing. **Jeffrey Scott Vitter:** Conceptualization, Data curation, Formal analysis, Funding acquisition, Methodology, Validation, Writing - review & editing.

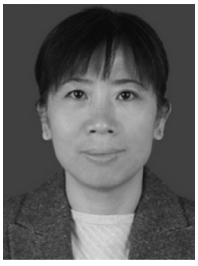
Acknowledgments

We thank to anonymous reviewer for their constructive comments. We also thank to Tinghuai Ma for providing some codes of GSR_GK_KDA_ELM. This work was supported in part by the National Natural Science Foundation of China grants 61741215 and 61373044, and by the U.S. National Science Foundation grant CCF-1017623.

References

[1] K.M. Borgwardt, H.P. Kriegel, Shortest-path kernels on graphs, in: Proceedings of ICDM, 2005, pp. 74–81.
[2] K.M. Borgwardt, C.S. Ong, S. Schonauer, S.V.N. Vishwanathan, A.J. Smola, H.P. Kriegel, Protein function prediction via graph kernels, Bioinformatics 21 (1) (2005) 47–56.
[3] H. Cai, V.W. Zhang, K.C. Chang, A comprehensive survey of graph embedding: problems, techniques, and applications, IEEE Trans. Knowl. Data Eng. 30 (9) (2018) 1616–1637.
[4] S. Cao, W. Lu, Q. Xu, Grarep: learning graph representations with global structural information, in: Proceedings of CIKM, 2015, pp. 891–900.
[5] C.C. Chang, C. Lin, LIBSVM: a library for support vector machines, ACM Trans. Intell. Syst. Technol. 2 (3) (2011) 1–27.
[6] S. Chen, C.F.N. Cowan, P.M. Grant, Orthogonal least squares learning algorithm for radial basis function networks, IEEE Trans. Neural Netw. 2 (2) (1991) 302–309.
[7] C. Cortes, V. Vapnik, Support vector networks, Mach. Learn. 20 (3) (1995) 273–297.
[8] F. Costa, K.D. Grave, Fast neighborhood subgraph pairwise distance kernel, in: Proceedings of ICML, 2010, pp. 255–262.
[9] A.K. Debnath, R.L.L.D. Compadre, G. Debnath, A.J. Shusterman, C. Hansch, Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity, Int. J. Med. Chem. 34 (2) (1991) 786–797.
[10] M. Defferrard, X. Bresson, P. Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, in: Proceedings of NIPS, 2016, pp. 3844–3852.
[11] M. Ester, H.P. Krigegl, J. Sander, X. Xu, A density-based algorithm for discoveringclusters in large spatial databases with noise, in: Proceedings of KDD, 1996, pp. 226–231.
[12] B.J. Frey, D. Dueck, Clustering by passing messages between data points, Science 315 (5814) (2007) 972–976.
[13] A. Grover, J. Leskovec, node2vec:scalable feature learning for networks, in: Proceedings of KDD, 2016, pp. 855–864.
[14] D. Han, Y. Hu, S. Ai, G. Wang, Uncertain graph classification based on extreme learning machine, Cognit. Comput. 7 (3) (2015) 346–358.
[15] G. Huang, L. Chen, C.K. Siew, Universal approximation using incremental constructive feedforward networks with random hidden nodes, IEEE Trans. Neural Netw. 17 (4) (2006) 879–892.
[16] G. Huang, H. Zhou, X. Ding, R. Zhang, Extreme learning machine for regression and multiclass classification, IEEE Trans. Syst. Man Cybern. 42 (2) (2011) 513–529.
[17] H. Kashima, K. Tsuda, A. Inokuchi, Marginalized kernels between labeled graphs, in: Proceedings of ICML, 2003, pp. 321–328.
[18] N.M. Kriege, C. Morris, A. Rey, C. Sohler, A property testing frameword for the theoretical expressivity of graph kernels, in: Proceedings of IJCAI, 2018, pp. 2348–2354.
[19] A. Lavecchia, Machine-learning approaches in drug discovery: methods and applications, Drug Disc. Today 20 (3) (2015) 318–331.
[20] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.
[21] T. Ma, W. Shao, Y. Hao, J. Cao, Graph classification based on graph set reconstruction and graph kernel feature reduction, Neurocomputing 296 (2018) 33–45.
[22] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, arXiv preprint (2013). ArXiv:1301.3781.
[23] A. Narayanan, M. Chandramohan, L. Chen, Y. Liu, S. Saminathan, subgraph2vec: learning distributed representations of rooted sub-graphs from large graphs, arXiv preprint (2016). ArXiv:1606.08928.
[24] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, S. Jaiswal, graph2vec: learning distributed representations of graphs, arXiv preprint (2017). ArXiv:1707.05005.
[25] I. Newman, C. Sohler, Every property of hyperfinite graphs is testable, SIAM J. Comput. 42 (3) (2013) 1095–1112.
[26] B. Neyshabur, A. Khadem, S. Hashemifar, S.S. Arab, NETAL: a new graph-based method for global alignment of protein-protein interaction networks, Bioinformatics 29 (13) (2013) 1654–1662.
[27] S. Pan, J. Wu, X. Zhu, G. Long, C. Zhang, Finding the best not the most: regularized loss minimization subgraph selection for graph classification, Pattern Recognit. 48 (11) (2015) 3783–3796.
[28] B. Perozzi, A. Rami, S. Skiena, Deepwalk: online learning of social representations, in: Proceedings of KDD, 2014, pp. 701–710.
[29] N. Przulj, D.G. Corneil, I. Jurisica, Modeling interactome: scale-free or geometric? Bioinformatics 20 (18) (2004) 3508–3515.
[30] R. Rehurek, P. Sojka, Software framework for topic modelling with large corpora, in: Proceedings of LREC Workshop, 2010, pp. 45–50.
[31] N. Shervashidze, P. Schweitzer, E.J. Leeuwen, K. Mehlborn, K.M. Borgwardt, J. Mach, Weisfeiler-lehman graph kernels, Learn. Res. 12 (2011) 2539–2561.
[32] A. Smalter, J. Huan, G. Lushington, Structure-based pattern mining for chemical compound classification, APBC, poster, 2008.
[33] A. Smalter, J. Huan, G. Lushington, Graph wavelet alignment kernels for drug virtual screening, J. Bioinf. Comput. Biol. 7 (3) (2009) 473–497.
[34] A. Taheri, K. Gimpel, T. Berger-Wolf, Learning graph representations with recurrent neural network autoencoders, in: Proceedings of KDD Deep Learning Day, 2018, pp. 1–8.
[35] H. Toivonen, A. Srinivasan, R.D. King, S. Kramer, C. Helma, Statistical evaluation of the predictive toxicology challenge 2000–2001, Bioinformatics 19 (10) (2003) 1183–1193.
[36] N. Wale, I.A. Watson, G. Karypis, Comparison of descriptor spaces for chemical compound retrieval and classification, Knowl. Inf. Syst. 14 (3) (2008) 347–375.
[37] X. Yan, J. Han, gspan: graph-based substructure pattern mining, in: Proceedings of ICDM, 2002, pp. 721–724.
[38] P. Yanardag, S.V.N. Vishwanathan, Deep graph kernels, in: Proceedings of SIGKDD, 2015, pp. 1365–1374.
[39] Y. Yu, Z. Pan, G. Hu, H. Ren, Graph classification based on sparse graph feature selection and extreme learning machine, Neurocomputing 261 (2017) 20–27.
[40] T. Zhang, R. Ramakrishnan, M. Livny, BIRCH: an efficient data clustering method for very large databases, in: ACM Sigmod ICMD, 1996, pp. 103–114.
[41] Y. Zhu, J. Yu, H. Cheng, L. Qin, Graph classification: a diversified discriminative feature selection approach, in: Proceedings of CIKM, 2012, pp. 205–214.

Zhigang Sun received the B.S. degree from University of Jinan in 2007, and the M.S. degree from Guilin University of Electronic Technology in 2010. He is currently pursuing the Ph.D. degree at Xidian University, China. His research interests include graph classification, graph indexing and search, design and analysis of algorithms, and compressed indexes.



Hongwei Huo (M'00-SM'17) received the B.S. degree in mathematics from Northwest University, China, and the M.S. degree in computer science and the Ph.D. degree in electronic engineering from Xidian University. She is currently a Professor with the Department of Computer Science, Xidian University. Her research interests include the design and analysis of algorithms, graph indexing and classification, compressed data structures, external memory algorithms and compressed indexes, genome compression and pattern searching, parallel and distributed algorithms, and algorithm engineering.

Jun Huan (SM'11) received the Ph.D. degree in computer science from the University of North Carolina. He is currently the head of the Big Data Lab at Baidu Research. He was formerly a professor in the Department of Electrical Engineering and Computer Science at the University of Kansas, where he directed the Data Science and Computational Life Sciences Laboratory. He held courtesy appointments at the KU Bioinformatics Center, the KU Bioengineering Program, and a visiting professorship from GlaxoSmithKline plc. He has authored over 120 peer-reviewed papers in leading conferences and journals and has graduated more than ten PhD students. He works on data science, deep learning, data mining and interdisciplinary topics including bioinformatics. He serves the editorial boards of several international journals, including the Springer Journal of Big Data, the Elsevier Journal of Big Data Research, and the International Journal of Data Mining and Bioinformatics. He regularly serves the program committee of top-tier international conferences on machine learning, data mining, big data, and bioinformatics.

Jeffrey Scott Vitter (F'93) received the B.S. degree (with highest honors) in mathematics from the University of Notre Dame in 1977, the Ph.D. degree in computer science from Stanford University in 1980, and an M.B.A degree from Duke University in 2002. He is currently Distinguished Professor of Computer and Information Science with the University of Mississippi, where he served as chancellor from 2016 to 2019. From 2010 to 2015, he was provost and executive vice chancellor and Roy A. Roberts Distinguished Professor at the University of Kansas. From 2008 to 2010, he was on the faculty at Texas A&M University, where he also served as a provost and executive vice president for academics. From 2002 to 2008, he was the Frederick L. Hovde Dean of Science with Purdue University. From 1993 to 2002, he held the Gilbert, Louis, and Edward Lehrman Distinguished Professorship with Duke University, where he also served as a chair of the Department of Computer Science and a co-director of Duke's Center for Geometric and Biological Computing. From 1980 to 1992, he advanced through the faculty ranks in computer science at Brown University. He is a Fellow of the Guggenheim Foundation, National Academy of Inventors, AAAS, ACM, and IEEE, and he is a U.S. National Science Foundation Presidential Young Investigator and Fulbright Scholar. He has authored the book Algorithms and Data Structures for External Memory, co-authored the books Design and Analysis of Coalesced Hashing and Efficient Algorithms for MPEG Video Compression, co-edited the collections External Memory Algorithms and Algorithm Engineering, and co-holder of patents in the areas of external sorting, prediction, and approximate data structures. His research interests span the design and analysis of algorithms, external memory algorithms, data compression, databases, compressed data structures, parallel algorithms, machine learning, random variate generation, and sampling. He has received the IBM Faculty Development Award, the ACM Recognition of Service Award (twice), and the 2009 ACM SIGMOD Test of Time Award. He has served on several boards and professional committees. He serves or has served on the editorial boards of Algorithmica, Communications of the ACM, IEEE Transactions on Computers, Theory of Computing Systems, and SIAM Journal on Computing, and has edited several special issues.