Knowledge-Based Systems 228 (2021) 107301

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

Graph classification based on skeleton and component features

Xue Liu [a], Wei Wei [b,c,d,e,*], Xiangnan Feng [b,c,e,f], Xiaobo Cao [a], Dan Sun [b]

[a] Beijing System Design Institute of Electro-Mechanic Engineering, Beijing, 100854, China
[b] School of Mathematical Sciences, Beihang University, Beijing, 100191, China
[c] Key Laboratory of Mathematics, Informatics and Behavioral Semantics, Ministry of Education, 100191, China
[d] Beijing Advanced Innovation Center for Big Data and Brain Computing, Beihang University, Beijing, 100191, China
[e] Peng Cheng Laboratory, Shenzhen, Guangdong, 518066, China
[f] Center for Humans and Machines, Max Planck Institute for Human Development, Berlin 14195, Germany

ARTICLE INFO

Article history:
Received 24 January 2021
Received in revised form 8 July 2021
Accepted 11 July 2021
Available online 13 July 2021

Keywords:
Graph representation
Graph classification
Feature learning

ABSTRACT

Most existing popular methods for learning graph embedding only consider fixed-order global structural features but lack hierarchical representation for structures. To address this weakness, we propose a novel graph embedding algorithm named GraphCSC that realizes classification leveraging skeleton information from anonymous random walks with fixed-order length, and component information derived from subgraphs with different sizes. Two graphs are similar if their skeletons and components are both similar. Thus in our model, we integrate both of them together into embeddings as graph homogeneity characterization. We demonstrate our model on different datasets in comparison with a comprehensive list of up-to-date state-of-the-art baselines, and experiments show that our work is superior in real-world graph classification tasks.

1. Introduction

Graph classification to distinguish the class labels of graphs in a dataset is an important task with practical applications in a large spectrum of fields (e.g., bioinformatics [1,2], social network analysis [3] and chemoinformatics [4]). In these areas, data can be usually represented as graphs with labels. For example, in bioinformatics, a protein molecule can be represented as a graph whose nodes correspond to atoms, and edges signify the existence of chemical bonds between atoms. The graphs are allocated with different labels on the basis of various functions. To classify these graphs, we usually make a common assumption that protein molecules with similar structures exhibit similar functional properties.

More recently, there have been a surge of approaches that seek to learn representations or embeddings and then make classification [5–8]. The idea behind these learning approaches focuses on graph structure representation by learning a mapping that embeds nodes or entire (sub)graphs into low-dimensional vector space. Most methods belong to one of the two categories: (1) neural network methods that learn large-scale structures of target graph, (2) kernel methods that focus on small-size structures of target graph. Different graph structures imply distinctive features, which lead to various possible classification outcomes.

Graph neural networks (GNNs) [9–11] use a recurrent network framework to transmit information from a calculated node to another until reaching a stop situation. Analogous to image-based convolutional neural networks (CNNs) [12], PATCHY-SAN (PSCN) is motivated to operate on locally connected regions of the input to learn graph embeddings [13]. Graph convolutional networks (GCNs) [14–16] operate on graph data directly by spectral filters to exploit local areas, and then extract meaningful local features shared within the entire graph to get a large-scale representation. Recently, some important subsequent variants of graph neural networks are proposed. MLC-GCN introduces an adaptive structural coarsening module to GCNs and significantly improves representation ability [17]. GNNs with structured multi-head self-attention architecture achieves good predicting accuracy since this self-attention strategy makes full use of graph information among node, layer and graph levels [18]. The success of neural networks relies on enormous amount of data as well as iterative calculations. This iterative mechanism involves local information of graph into the overall embedding as learning process going on.

Subgraph isomorphism has been proven to be NP-complete, however graph isomorphism problem neither has been proven NP-complete nor could be solved by a polynomial-time algorithm [19]. Graph kernels [20–22] differentiate two graphs by recursively decomposing them into substructures and then defining a function to make classification based on graphs similarity measures in an unsupervised way. Graph kernels bridge the gap between graph data and a wide range of machine learning methods such as Support Vector Machines (SVM), regression,

* Corresponding author.
E-mail address: weiw@buaa.edu.cn (W. Wei).

https://doi.org/10.1016/j.knosys.2021.107301
0950-7051/

clustering and Principal Components Analysis (PCA), etc. Graph kernel approaches can be divided into two categories in general: walk-based patterns [6] and limited-size subgraph methods. In walk-based patterns, graph kernels count matched random walk pairs between two graphs [23]. The shortest path kernels count the number of shortest path pairs with same beginning node, sink labels as well as length in two graphs [24]. Graphlet kernels deal with graph isomorphism by counting the occurrences of all types of fixed size subgraphs [25,26]. Graphs are considered to be similar if they share common subgraphs as many as possible. Neighborhood Subgraph Pairwise Distance Kernel (NSPDK) sets the source nodes of two graphs at a fixed distance as roots, and then calculates the number of identical pairs of subgraphs that grow up from roots [4]. Weisfeiler–Lehman graph kernels are highly efficient kernels to measure graph isomorphism using subtree-like patterns [20].

There are two critical limitations of graph kernels: (1) Many of them do not provide explicit embeddings, so that kernels are not suitable for many proposed machine learning algorithms that require operations on vectors directly. (2) Specific substructure (i.e., random walks, subgraphs, etc.) parameters need to be preset, for instance, the length of random walk or the shapes of subgraphs. This could inevitably miss substructures not in preset shapes yet actually crucial to the embeddings.

Anonymous Walk Embeddings (AWE) is a novel graph embedding method that relies on the distribution of special random walks named anonymous random walks [27]. In analogy to graph kernels and to avoid sparse distribution, AWE method uses random walks in anonymous manner to catch the "skeletons" of the whole graph. Micali et al. has shown that anonymous walks are Markov processes from starting nodes [28] and are able to reconstruct the original graphs with adequate samplings. Two graphs with similar distributions of anonymous walks are regarded as topological similar [27].

However, due to low distributions, AWE may neglect important subgraphs which actually determine crucial graph properties. As illustrated in Fig. 1, we take phenol (C_6H_5OH) and methylbenzene ($C_6H_5CH_3$) as a simple example. Each of them can be decomposed into two major parts, the main structure (benzene ring) as the skeleton, and functional groups (i.e., hydroxy "–OH" and methyl "–CH_3"). Both of them play an important role in molecular properties and functions. Taking these two molecules as topological graphs and implementing random walks on them, it is shown that the frequency of benzene ring edges being covered is twice more than that of hydroxy or methyl edges. In AWE, these two graphs are structure similar since random walks catch same main structures as skeletons. However, phenol and methylbenzene have quite different chemical properties, which are determined by their different functional groups and are easy to be neglected because of low distributions. In addition, AWE lacks a hierarchical representation of the entire graph structure: structures obtained by AWE have identical size (walk length), thus substructures whose sizes are not equal to the anonymous walk structures could not be observed and represented efficiently.

Our approach. We design a novel data-driven framework named GraphCSC that represents entire graph in low-dimension vectors containing both main structure features and components information in a global view. On the one hand, inspired by the success of AWE, our methodology uses anonymous walks to represent graph skeleton information. On the other hand, in order to avoid huge computational complexity, we take a sampling strategy by finding special subgraphs, i.e., frequent subgraphs, to represent graph components. Frequent subgraphs are determined by a fixed threshold hyperparameter, which indicates what kinds of subgraphs can be regarded as frequent ones. The frequent-based subgraphs contain not only the underlying semantics within an individual graph but also the relationships among graphs. To learn graph embedding, motivated by a novel supervised document method Paragraph Vector-Distributed Bag of Words (**PV-DBOW**) [29], anonymous walks and frequent subgraphs in our model are both treated as words and each graph corresponds to a document. Two graphs are close in embedding space if they share similar skeletons and components.

Our contribution. To the best of our knowledge, GraphCSC is a new framework that learns embeddings consisting of both skeleton features and component information compared to other existing embedding approaches. GraphCSC actually studies the representation of graph structures in both horizontal and vertical perspectives: in horizontal perspective, we attempt to characterize relatively high-order structures using anonymous random walks with same walk lengths, which determines the skeletons of graphs; in vertical perspective, our model focuses on what kinds of subgraphs with different sizes or shapes they have. We use a Natural Language Processing (**NLP**) training framework with skeletons and components together as inputs to learn graph representation with combined information. Through empirical evaluation on multiple real-world datasets, experiments show that our model is competitive among a series of established baselines.

2. Problem statement

In this work, we consider the graph classification task. In the general setting, for a set of graphs $\mathbb{G} = \{G_1, \ldots, G_N\}$ with labels, the goal is to learn a function $\varphi : \mathbb{G} \to \mathbb{L}$, where $\mathbb{G}$ denotes the input space of graphs and $\mathbb{L}$ is the set of graph labels. Each weighted graph G_i with label l_i, $i = 1, \ldots, N$, is a tuple $G_i = (V_i, E_i, \Omega_i)$, where V_i is the set of n_i vertices from G_i, $E_i \subseteq V_i \times V_i$ is the set of edges from G_i, and Ω_i represents the set of edge weights from G_i.

To learn the embedding vectors of graphs as inputs of machine learning model is the core part in graph classification problems. In more details, given a set of graphs $\mathbb{G} = \{G_1, \ldots, G_N\}$, the goal of embedding is to learn a graph representation matrix $\mathbf{X}_{N\times d}$ by a mapping $\psi : \mathbb{G} \to \mathbb{R}^{1\times d}$, where each ith row of $\mathbf{X}_{N\times d}$ denotes a d-dimension vector of graph G_i.

3. Background

We will leverage two techniques **Skipgram** [6] and **PV-DBOW** to learn graph representation, which have already been widely applied in **NLP**. Before we propose our approach, we review these powerful models and state them as background of our model.

3.1. word2vec and Skipgram

To get continuous-valued word vector representation is the main task in **NLP** applications. **word2vec** [30] uses **Skipgram** to learn low-dimension embeddings of words that capture rich semantic relationships among them. **Skipgram** maps words contained in similar sentences to "near" positions in embedding space, i.e., their representation vectors are similar.

Given the target word ω_t from vocabulary set V and a sequence of words $\omega_1, \ldots, \omega_t, \ldots, \omega_T$, the context $\omega_{t-c}, \ldots, \omega_t, \ldots, \omega_{t+c}$ is denoted as a fixed number of words surrounding ω_t within a window c. **Skipgram** maximizes the co-occurrence probability among words that appear in context:

$$\sum_{t=1}^{T} \log \mathbf{Pr}(\omega_{t-c}, \ldots, \omega_{t+c} | \omega_t). \quad (1)$$

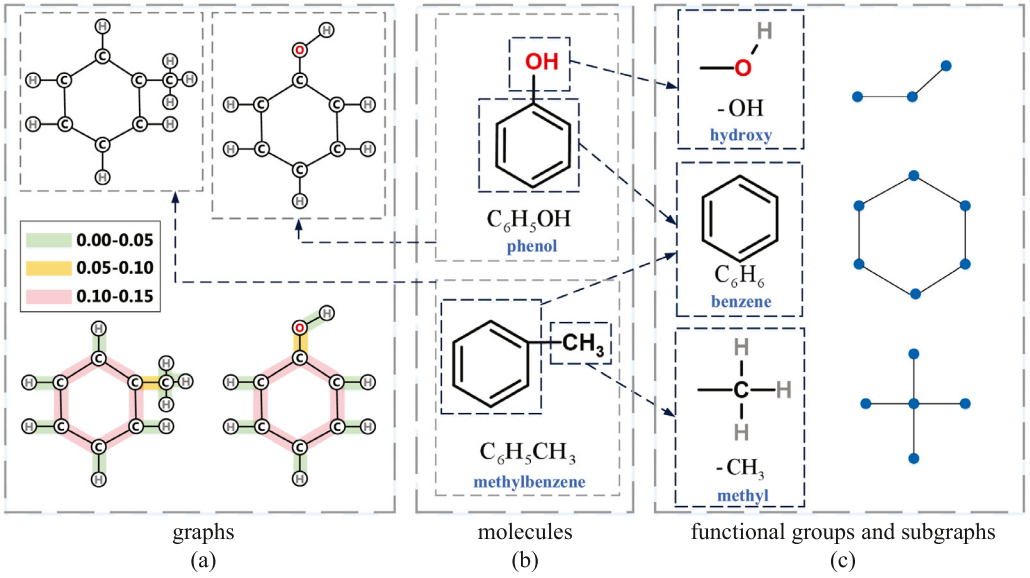

Fig. 1. Overview of topological graphs (as shown in (a)), molecular chemical structures (as shown in (b)), their corresponding functional groups and subgraphs (as shown in (c)) of phenol (C_6H_5OH) and methylbenzene ($C_6H_5CH_3$). In (a), we do random walks with walk length $l = 4$ from each node on every topological graph of phenol and methylbenzene. The frequency p of each edge covered in random walks is $p = \frac{m_e}{M}$, where m_e is the frequence of edge e being covered and M denotes the total number of edges being covered. Edges with different frequencies are represented in different colors. Phenol and methylbenzene in (c) can be decomposed into three functional groups (hydroxy, benzene and methyl), which are regarded as 2-order subgraph (2-hop path), hexagon loop (6-order subgraph) and cross, respectively.

The conditional probability $\mathbf{Pr}(\omega_{t-c}, \ldots, \omega_{t+c}|\omega_t)$ could be approximated under the following independence assumption:

$$\mathbf{Pr}(\omega_{t-c}, \ldots, \omega_{t+c}|\omega_t) = \prod_{j=t-c, j\neq t}^{t+c} \mathbf{Pr}(\omega_j|\omega_t). \tag{2}$$

3.2. Softmax and negative sampling

Learning such a posterior distribution $\mathbf{Pr}(\omega_{t+j}|\omega_t)$, conventional classifiers such as logistic regression require vast computational resources since the number of labels is huge and equals vocabulary size $|V|$. To avoid heavy calculation, conditional probability distribution $\mathbf{Pr}(\omega_j|\omega_t)$ is defined by a Hierarchical Softmax [31]:

$$\mathbf{Pr}(\omega_j|\omega_t) = \frac{\exp(\boldsymbol{\omega}_t \cdot \boldsymbol{\omega}_j)}{\sum_{i=1}^{|V|} \exp(\boldsymbol{\omega}_t \cdot \boldsymbol{\omega}_i)}, \tag{3}$$

where $\boldsymbol{\omega}_t$ and $\boldsymbol{\omega}_{t+j}$ are embedding vectors for word ω_t and ω_{t+j}. Thus the co-occurrence probability (1) is written as:

$$\sum_{t=1}^{T} \log \prod_{j=t-c, j\neq t}^{t+c} \frac{\exp(\boldsymbol{\omega}_t \cdot \boldsymbol{\omega}_j)}{\sum_{i=1}^{|V|} \exp(\boldsymbol{\omega}_t \cdot \boldsymbol{\omega}_i)}. \tag{4}$$

Here to speed up the training process of **Skipgram**, negative sampling [32] method stochastically takes a small set of words as negative samples which are not involved in context. Then only target word and negative samples are updated instead of every word from vocabulary set V during the iterative training process. This strategy would be efficient especially for cases where tasks face huge computational pressure.

3.3. doc2vec And PV-DBOW

In analogy to **word2vec**, **doc2vec** [29] uses **PV-DBOW** to learn representations of arbitrary size document in a document set. More specifically, given a document set $\mathbb{D} = \{D_1, \ldots, D_N\}$ with a set of words $V = \{\omega_1, \ldots, \omega_{|V|}\}$, for the target document $D_t \in \mathbb{D}$ which contains a sequence of words $\{\omega_1, \ldots, \omega_l\}$, the goal is to learn low-dimension vector representation of document D_t by maximize the following log probability:

$$\sum_{i=1}^{l} \log \mathbf{Pr}(\omega_j|D_t). \tag{5}$$

The conditional probability $\mathbf{Pr}(w_j|D_t)$ above is calculated as:

$$\mathbf{Pr}(\omega_j|D_t) = \frac{\exp(\boldsymbol{D}_t \cdot \boldsymbol{\omega}_j)}{\sum_{i=1}^{|V|} \exp(\boldsymbol{D}_t \cdot \boldsymbol{\omega}_i)}, \tag{6}$$

where $\boldsymbol{D}_t$ and $\boldsymbol{\omega}_j$ are corresponding representation vectors of D_t and ω_j, and $|V|$ is the number of all words across all documents in $\mathbb{D}$. The log probability (5) could be also approximated efficiently using negative sampling.

4. Proposed model

Our proposed model GraphCSC has two main modules, skeleton module and component module, as summarized in Fig. 5. Skeleton module (in Algorithm 1) and component module (in Algorithm 2) represent corresponding substructures separately but synchronously. GraphCSC integrates these two modules and optimizes overall loss function by gradient descent strategy.

4.1. Skeleton module

Here we focus on common higher-order structures representation among graphs. We utilize a special form of random walk, anonymous random walk, to represent such structures and provide the definition of skeleton of graphs.

4.1.1. Anonymous random walks

In graph G, a random walk w is defined as a finite sequence $(v_0, v_1, \ldots, v_l)$ with length l, where v_0 represents the root node and node v_{i+1} is sampled independently among the neighbors of node v_i.

Random walks are regarded as Markov processes and recently, anonymous random walks, as special forms of random walks,

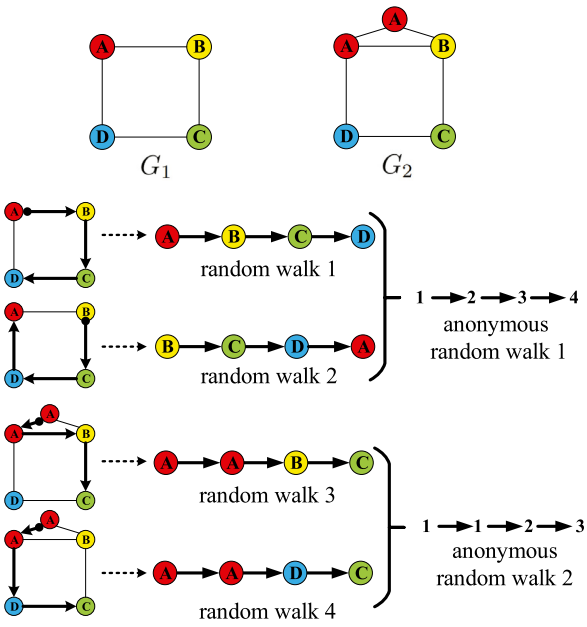

Fig. 2. We sample four random walks with length 3 (marked in different colors) in G_1, G_2 and illustrate their relation with corresponding anonymous random walks.

have been proven to be capable of learning graphs structural properties and reconstructing graphs with full descriptions of state of every node appeared along the random walk process in its own label space instead of global label space [28]. Anonymous random walks translate random walks into a sequence of integers recording positions that appear first. This makes walks in anonymous experiments more flexible and compact. More precisely, the anonymous random walk for random walk $w = (v_0, v_1, \ldots, v_l)$ with length l is a sequence of integers $a = g(w) = (f(v_0), f(v_1), \ldots, f(v_l))$, in which f is the position function defined as $f(v_i) = |(v_0, \ldots, v_{i'})|$, where i' is the smallest integer such that $v_{i'} = v_i$.

Example. Random walks (A, B, C, D) and (B, C, D, A) sampled from G_1, (A, A, B, C) and (A, A, D, C) from G_2 in Fig. 2 are completely different. However, in anonymous random walks view, the new label for each node is redefined as the position of the first occurrence of node with same label in the random walk sequence. This makes the initial four different random walks be converted into two anonymous random walks, (1, 2, 3, 4) and (1, 1, 2, 3).

As shown in Fig. 3, all different random walks with length 3 in G_1 and G_2 from Fig. 2 could be converted into only 8 anonymous random walks. Random walks record the labels information of nodes traveled, and with sufficient samples, they can accurately capture structure traits and reconstruct the original graph. But over precise information will not be efficient to capture graph structure features because of sparse distribution over all random walks.

4.1.2. Skeleton

We draw independently a set of ξ random walks $W_{G_i} = \{w_1, \ldots, w_\xi\}$ with length l on G_i from graphs set $\mathbb{G} = \{G_1, \ldots, G_i, \ldots, G_N\}$, and calculate its μ anonymous random walks $A_{G_i} = \{a_1, \ldots, a_\mu\}$.

For a large graph G_i, to count all possible anonymous random walks requires vast computational resources. However, sampling ζ random walks with length l to approximate actual distribution of anonymous random walks, the overall computing running time will be $O(\zeta l)$ [16]. The relation between the estimation of samples

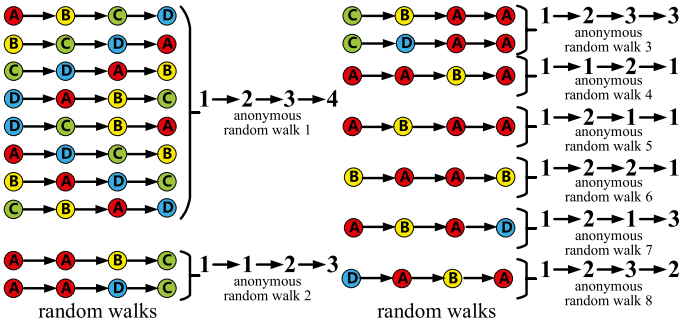

Fig. 3. All kinds of different random walks with length 3 in G_1 and G_2 from Fig. 2 and their corresponding anonymous random walks.

number ζ and the number of anonymous random walks λ is determined by hyperparameters ε and δ [33]:

$$\zeta = [\frac{2}{\varepsilon^2}(log(2^\lambda - 2) - log(\delta))], \varepsilon > 0, 0 \leq \delta \leq 1. \quad (7)$$

Next we tend to leverage vectors to indicate whether some specific anonymous random walks are contained in a graph or not. Given graphs set $\mathbb{G} = \{G_1, \ldots, G_i, \ldots, G_N\}$, we sample random walks with length l on each graph in $\mathbb{G}$ as set $W = \{w_1, \ldots, w_m, \ldots, w_\xi\}$. If the corresponding anonymous random walks set $A = \{a_1, \ldots, a_s, \ldots, a_\mu\}$ has μ unequal elements, then the **skeleton** for graph G_i is denoted as a $1 \times \mu$ shape vector $\boldsymbol{s}_{G_i} = [\xi_s]_{1\times\mu}$, where

$$\xi_s = \begin{cases} 1, & if\ a_s \in G_i \\ 0, & else \end{cases}, s = 1, \ldots, \mu. \quad (8)$$

Algorithm 1: Skeleton Module.

Input: $\mathbb{G} = \{G_1, \ldots, G_N\}$: graphs set; l: random walk length; T: random walk times.
Output: A: anonymous random walks set; $\boldsymbol{s}_{G_i}$, $i = 1, \ldots, N$: **skeleton** for each graph G_i in $\mathbb{G}$.

```
% Mining Anonymous Walks
A = ∅
for each G_i in 𝔾 do
    A_{G_i} = ∅
    for each node v_j in V(G_i) do
        for k = 1; k < T; k++ do
            w = RandomWalk(G_i, v_j, l)
            a = g(w)
            if a not in A_{G_i} then
                A_{G_i} = A_{G_i} ∪ {a}
            end if
        end for
    end for
end for
A = ⋃_{i=1}^{N} A_{G_i}
% Getting skeletons
for each G_i in 𝔾 do
    initialize s_{G_i} = [s_k]_{1×|A|}, s_k = 0
    for each anonymous random walk a_k in A do
        if a_k in A_i then
            s_k = 1
        end if
    end for
end for
return A; s_{G_i}, i = 1, ..., N
```

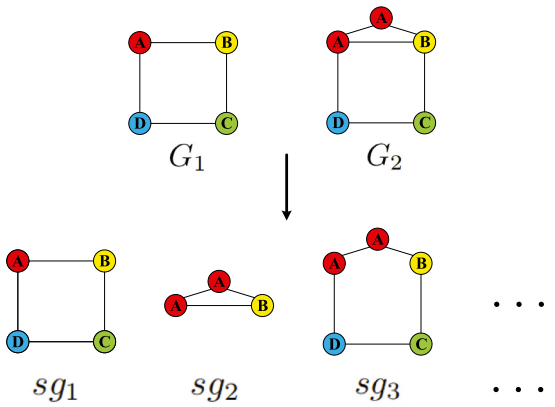

Fig. 4. Different kinds of subgraphs mined from G_1 and G_2 in Fig. 2.

4.2. Component module

Component module uses a more flexible approach to capture similarity between two graphs by frequent patterns (e.g., itemsets, subsequences, or subgraphs), which appear in a dataset with frequency no less than a user-specified minimum support (*min_sup*) threshold. Component module basically includes three steps: (1) mining frequent subgraphs, (2) selecting features, (3) learning component features.

4.2.1. Frequent subgraph

A graph sg is called a subgraph of a graph G if nodes satisfy $\mathbf{V}(sg) \subseteq \mathbf{V}(\mathbf{G})$ and edges satisfy $\mathbf{E}(sg) \subseteq \mathbf{E}(G)$. Two graphs are similar if their subgraphs are similar since subgraphs have been recognized as fundamental units and building blocks of complex networks [34–36]. But usually only a fraction of the large amount number of subgraphs are actually relevant to data mining problems.

Our methodology focuses on mining subgraphs according to different proportions in graph set, and then embeds these distribution features into graph component representation. Let sg be a subgraph in subgraphs set SG mined from $\mathbb{G} = \{G_1, \ldots, G_i, \ldots, G_N\}$, and it will be called frequent subgraph fsg if $\frac{|\{G_i \in \mathbb{G} | sg \in G_i\}|}{|\mathbb{G}|} \geq \theta$, where θ denotes the *min_sup* threshold and $0 \leq \theta \leq 1$. This fraction $\frac{|\{G_i \in \mathbb{G} | sg \in G_i\}|}{|\mathbb{G}|}$ is defined as the *support* of G_i.

Example. We still take G_1, G_2 in Fig. 2 and subgraphs sg_1, sg_2, sg_3 in Fig. 4 as examples. If we set *min_sup* threshold $\theta = 1$, only subgraph sg_1 can be regarded as frequent subgraph. However, if we set *min_sup* threshold $\theta = 0.5$, all of sg_1, sg_2 and sg_3 will be regarded frequent.

Frequent pattern, as a form of non-linear feature combinations over the set of different subgraphs, has higher discriminative power than that of single kind of subgraph because they capture more underlying semantics of the data. The key point is to specify the hyperparameter *min_sup* threshold θ used in model in frequent pattern, and we will study the influence of θ for machine learning tasks in Section 5. If an infrequent feature is selected, the model cannot generalize well on the test data since it is built based on statistically minor observations, hence the discriminative power of low-support features will be limited [37].

4.2.2. Component

To mine all frequent subgraphs will face two challenges: (1) It is computational infeasible to find all subgraphs since the time complexity equals $O(2^{|V|})$. (2) Subgraph isomorphism checking has been proven to be NP-complete [38] thus it will be unrealistic to compare all subgraphs. A slightly less restrictive measure of similarity can be defined on the basis of the largest common subgraphs in two graphs, but unfortunately the problem of finding the largest common subgraph of two graphs is NP-complete as well [38].

To tackle with such two challenges above, **gSpan** [39] algorithm builds a new lexicographic order among graphs, and maps each graph to a unique minimum DFS-code as its canonical label. With this DFS-code, **gSpan** adopts the depth-first search strategy to discover frequent subgraphs efficiently until either when the *support* of a graph is less than *min_sup* threshold θ, or its code is not a minimum code, which means this graph and all its descendants have been generated and discovered before.

Now we define the **component** to indicate what frequent subgraphs graph G_i have. Given a set of graphs $\mathbb{G} = \{G_1, \ldots, G_i, \ldots, G_N\}$ and *min_sup* threshold θ, all different frequent subgraphs obtained by **gSpan** are contained in set $FSG = \{fsg_1, \ldots, fsg_t, \ldots, fsg_v\}$. The **component** of G_i is defined as an $1 \times v$ shape vector $\boldsymbol{c}_{G_i} = [\eta_t]_{1\times v}$, where

$$\eta_t = \begin{cases} 1, & if\ fsg_t \in G_i \\ 0, & else \end{cases}, t = 1, \ldots, v. \tag{9}$$

Algorithm 2: Component Module.

Input: $\mathbb{G} = \{G_1, \ldots, G_N\}$: graphs set; θ: *min_sup* threshold.
Output: FSG: frequent subgraphs set; $\boldsymbol{c}_{G_i}$, $i = 1, \ldots, N$: **component** for each graph G_i in $\mathbb{G}$

```
% Mining Frequent Subgraphs
FSG = gSpan(𝔾, θ)
for each G_i in 𝔾 do
    FSG_{G_i} = ∅
    for each frequent subgraph fsg_j in FSG do
        if fsg_j in G_i then
            FSG_{G_i} = FSG_{G_i} ∪ {fsg_j}
        end if
    end for
end for
% Getting components
for each G_i in 𝔾 do
    initialize c_{G_i} = [c_k]_{1×|FSG|}, c_k = 0
    for each frequent subgraph fsg_k in FSG do
        if fsg_k in FSG_i then
            s_k = 1
        end if
    end for
end for
return FSG; c_{G_i}, i = 1, ..., N
```

4.3. Training

For graph dataset $\mathbb{G} = \{G_1, \ldots, G_i, \ldots, G_N\}$, suppose that all anonymous random walks with length l mined are in $A = \{a_1, \ldots, a_\mu\}$ and all frequent subgraphs mined with *min_sup* threshold θ are in $FSG = \{fsg_1, \ldots, fsg_v\}$. We leverage **PV-DBOW** to learn graphs embedding matrix $\mathbf{X}$ whose each row is an embedding of each graph, anonymous random walks matrix $\mathbf{M}_A$ whose each row is an embedding of each anonymous random walk and frequent subgraphs matrix $\mathbf{M}_{FSG}$ whose each row is an embedding of each frequent subgraph. Vectors $\boldsymbol{s}_{G_i}$ and $\boldsymbol{c}_{G_i}$ are corresponding **skeleton** and **component** for graph G_i, $i = 1, \ldots, N$.

The shapes of $\mathbf{X}$, $\mathbf{M}_A$ and $\mathbf{M}_{FSG}$ are $N \times d$, $\mu \times d$ and $v \times d$, respectively, where d is embedding dimension. **PV-DBOW** treats graph dataset $\mathbb{G}$ as documents set, each graph G_i in $\mathbb{G}$ as a document and each substructure mined in G_i as a word contained in a document.

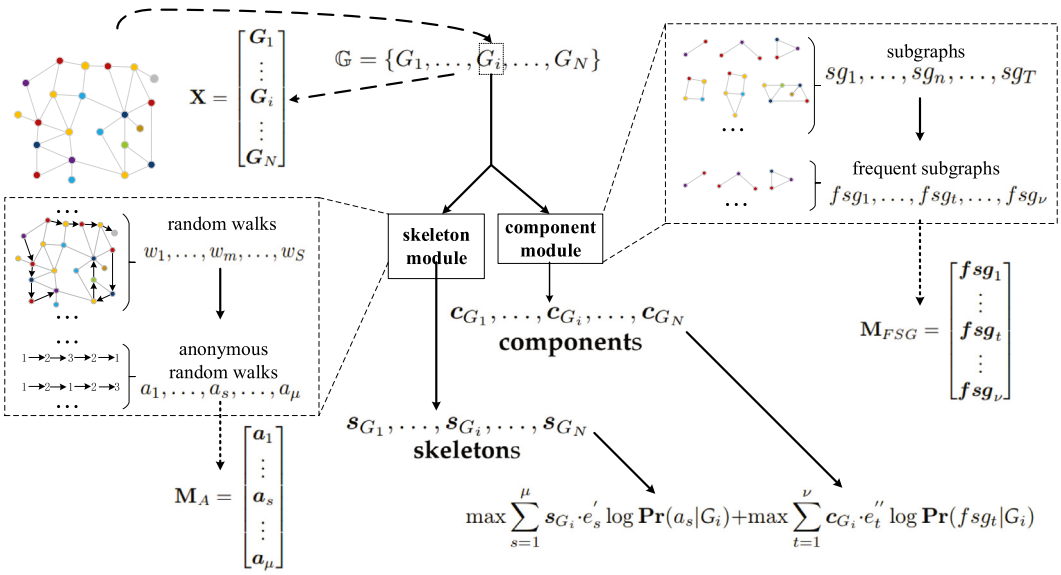

Fig. 5. Overview of GraphCSC which can be decomposed into two main parts, the skeleton module and component module. Skeleton module outputs anonymous random walks matrix $\mathbf{M}_A$ in $\mu \times d$ size and **skeleton**s for graphs from $\mathbb{G}$. Meanwhile, component module gets frequent subgraphs matrix $\mathbf{M}_{FSG}$ in $\nu \times d$ size and **component**s for graphs from $\mathbb{G}$. Finally, GraphCSC integrates these two modules and yields global optimization object (14).

To begin with, we focus on anonymous random walks and seek to optimize the following objective function, which maximizes the log-probability of anonymous random walks appearing in graph G_i:

$$\max \sum_{s=1}^{\mu} \boldsymbol{s}_{G_i} \cdot e'_s \log \mathbf{Pr}(a_s|G_i), \tag{10}$$

where e'_s is an $1 \times \mu$ binary vector with sth element 1 and jth element 0 if $j \neq s$.

The probability $\mathbf{Pr}(a_s|G_i)$ is defined as a softmax unit parametrized by a dot product of $\boldsymbol{a}_s$ and $\boldsymbol{G}_i$, which are the embedding vectors of a_s and G_i:

$$\mathbf{Pr}(a_s|G_i) = \frac{\exp(\boldsymbol{a}_s \cdot \boldsymbol{G}_i)}{\sum_{p=1}^{\mu} \exp(\boldsymbol{a}_p \cdot \boldsymbol{G}_i)}. \tag{11}$$

Next, we use same method to maximize the log-probability of predicting the frequent subgraphs that appear in graph G_i:

$$\max \sum_{t=1}^{\nu} \boldsymbol{c}_{G_i} \cdot e''_t \log \mathbf{Pr}(fsg_t|G_i), \tag{12}$$

where e''_t is an $1 \times \nu$ binary vector whose tth column element is 1 and jth column element equals 0 if $j \neq t$.

The probability $\mathbf{Pr}(fsg_t|G_i)$ is defined as a softmax function parametrized by a dot product of $\boldsymbol{fsg}_t$ and $\boldsymbol{G}_i$, which are embedding vectors of subgraph fsg_t and G_i:

$$\mathbf{Pr}(fsg_t|G_i) = \frac{\exp(\boldsymbol{fsg}_t \cdot \boldsymbol{G}_i)}{\sum_{q=1}^{\nu} \exp(\boldsymbol{fsg}_q \cdot \boldsymbol{G}_i)}. \tag{13}$$

Finally, the global optimization object could be derived from (10) and (12) as:

$$\max \sum_{s=1}^{\mu} \boldsymbol{s}_{G_i} \cdot e'_s \log \mathbf{Pr}(a_s|G_i) + \max \sum_{t=1}^{\nu} \boldsymbol{c}_{G_i} \cdot e''_t \log \mathbf{Pr}(fsg_t|G_i). \tag{14}$$

We yield its loss function as following:

$$\begin{aligned} \mathbf{L} = &\sum_{s=1}^{\mu} \boldsymbol{s}_{G_i} \cdot e'_s \log \sigma(\boldsymbol{a}_s \cdot \boldsymbol{G}_i) + \sum_{t=1}^{\nu} \boldsymbol{c}_{G_i} \cdot e''_t \log \sigma(\boldsymbol{fsg}_t \cdot \boldsymbol{G}_i) \\ &\sum_{p=1}^{\mu} \log(\sigma(-\boldsymbol{a}_p \cdot \boldsymbol{G}_i)) + \sum_{q=1}^{\nu} \log(\sigma(-\boldsymbol{fsg}_q \cdot \boldsymbol{G}_i)), \end{aligned} \tag{15}$$

where $\sigma(x) = \frac{1}{1+exp(-x)}$ is sigmoid function.

The last two items in Eq. (15) sum over all anonymous random walks and subgraphs directly, which are too expensive since μ and ν usually tend to be very large. Hence we proceed with an approximation by negative sampling to make the optimization problem tractable. The normalization terms from the softmax are replaced by K_1 anonymous random walks negative samples $\{a'_1, \ldots, a'_{K_1}\}$ from A but not contained in G_i and K_2 frequent subgraphs negative samples $\{fsg'_1, \ldots, fsg'_{K_2}\}$ from F but not contained in G_i. Thus Eq. (15) can be rewritten as:

$$\begin{aligned} \mathbf{L} = &\sum_{s=1}^{\mu} \boldsymbol{s}_{G_i} \cdot e'_s \log \sigma(\boldsymbol{a}_s \cdot \boldsymbol{G}_i) + \sum_{t=1}^{\nu} \boldsymbol{c}_{G_i} \cdot e''_t \log \sigma(\boldsymbol{fsg}_t \cdot \boldsymbol{G}_i) \\ &\sum_{p=0}^{K_1} log(\sigma(-\boldsymbol{a}'_p \cdot \boldsymbol{G}_i)) + \sum_{q=0}^{K_2} log(\sigma(-\boldsymbol{fsg}'_q \cdot \boldsymbol{G}_i)), \end{aligned} \tag{16}$$

where $\boldsymbol{a}'_p$ and $\boldsymbol{fsg}'_q$ are the embedding vectors of samples a'_p and fsg'_q respectively, and a'_p belongs to G_i when $p = 0$, fsg'_q is in G_i when $q = 0$.

We optimize loss function (16) with stochastic gradient descent and update $\boldsymbol{G}_i$. After the learning process finishes, two graphs are mapped into closed positions in embedding space if they share similar skeletons and components, and training process is summarized in Algorithm 3.

5. Experiments

In this section, to quantitatively evaluate classifying capability of our model, we conduct extensive experiments on a variety of widely-used datasets to compare with several state-of-the-art baselines.

5.1. Datasets

We evaluate our proposed method on binary classification task using seven real-world graph datasets whose statistics are summarized in Table 1. MUTAG [40] is a dataset of aromatic and heteroaromatic nitro compounds labeled according to whether they have a mutagenic effect on bacteria or not. PROTEINS [41] is

Algorithm 3: Training.

Input: $\mathbb{G} = \{G_1, \ldots, G_N\}$: graphs set; A: anonymous random walks set; $\boldsymbol{s}_{G_i}, i = 1, \ldots, N$: **skeleton** for each graph G_i in $\mathbb{G}$; FSG: frequent subgraphs set; $\boldsymbol{c}_{G_i}, i = 1, \ldots, N$: **component** for each graph G_i in $\mathbb{G}$; $e'_1, \ldots, e'_{|A|}$: $|A|$ binary vectors for **skeleton**s; $e''_1, \ldots, e''_{|FSG|}$: $|FSG|$ binary vectors for **component**s; d: embedding dimension; α: learning rate.

Output: $\mathbf{X}^{N\times d}$: graphs representation matrix

initialize $\mathbf{X} = [x_{ij}]_{N\times d}, x_{ij} \sim \boldsymbol{N}(0, 0.001)$
for each anonymous walk a_s in A **do**
  for each frequent subgraph fsg_t in FSG **do**
    $\mathbf{L}(\mathbf{X}) = -\boldsymbol{s}_{G_i} \cdot e'_s \log \mathbf{Pr}(a_s|G_i) - \boldsymbol{c}_{G_i} \cdot e''_t \log \mathbf{Pr}(fsg_t|G_i)$
    $\mathbf{X} = \mathbf{X} - \alpha \frac{\partial \mathbf{L}(\mathbf{X})}{\mathbf{X}}$
  end for
end for
return X

Table 1
Statistics of the benchmark graph datasets. The columns are: name of dataset, number of graphs, number of classes (maximum number of graphs in a class), average number of nodes/edges.

Datasets	Graph ♯	Class ♯	Average node ♯	Average edge ♯
MUTAG	188	2	17.93	19.79
DD	1178	2	284.32	715.66
PTC-MR	344	2	14.29	14.69
NCI-1	4110	2	29.87	32.30
NCI-109	4127	2	29.68	32.13
PROTEINS	1113	2	39.06	72.82
ENZYMES	600	6	32.63	64.14

a set of protein graphs where nodes represent secondary structure elements and edges indicate neighborhood in the amino-acid sequence or in 3-dimension space. ENZYMES [41] consists of protein tertiary structures obtained from the BRENDA enzyme database. DD [42] is a dataset of protein structures where nodes represent amino acids and edges indicate spatial closeness, which are classified into enzymes or non-enzymes. PTC-MR [43] consists of graph representations of chemical molecules labeled according to carcinogenicity on rodents. NCI-1, NCI-109 [1] are datasets of chemical compounds divided by the anti-cancer property (active or negative). These datasets have been made publicly available by the National Cancer Institute (NCI).

5.2. Baselines

In order to demonstrate the effectiveness of our proposed approach, we compare it with several baseline methods, all of which utilize the entire graph for feature extraction. These competitors can be categorized into four main groups:

- **Graph kernels based methods:** The shortest path (SP) [24] kernel measures the similarity of a pair of graphs by comparing the distance of the shortest paths between nodes in the graphs. Graphlet kernel (GK) [33] measures graph similarity by counting the number of different graphlets and Deep GK [44] is deep graphlet kernel. Weisfeiler–Lehman kernel (WL) [20] uses subtree pattern to mine structure information and Deep WL [44] is deep Weisfeiler–Lehman kernel.
- **Unsupervised graph embedding methods:** node2vec [13] is an unsupervised task agnostic method that learns entire graph embedding. It proposes every graph into a fixed size vector containing distributed representation of graph structures.
- **Supervised graph embedding methods:** PSCN [14] is a convolutional neural network algorithm which has achieved high classification accuracy on many datasets.
- **Unsupervised graph embedding methods:** AWE [27] uses anonymous random walks to embed entire graphs in an unsupervised manner.

5.3. Evaluation metrics

To evaluate the performance of GraphCSC, we randomly split the data into 10 roughly equal-size batches and perform a 10-fold cross validation on each dataset, in which 9 folds are used for training and 1 fold for validation. This process is repeated 10 times and an average accuracy is reported as prediction result. Since we focus on graph embedding and not on the classifier, we feed the embedding vectors to Support Vector Machine (SVM) with RBF kernel function and parameter σ varies from the range $[10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10]$.

5.4. Parameter sensitivity

Since we tend to show that the performance of our model GraphCSC could have a gain on experiments results than method only considers skeletons, to explore how the hyperparameters in skeleton module affect tasks performance will not be our priority. For brevity, the length l is set as 10 to generate a corpus of co-occurred anonymous random walks for all given datasets. To approximate actual distribution of anonymous random walks, we follow AWE [27] to set the sampling hyperparameters $\varepsilon = 1$ and $\delta = 0.05$. In order to evaluate how the parameter sensitivity of *min_sup* threshold θ and embedding dimensions *dim* affect the classification performance of GraphCSC on datasets, all parameters are assumed to be default except θ and *dim*.

We first conduct experiments with $dim = 128$ and then assess the classification accuracy as a function of *min_sup* threshold θ for different datasets. Best performance is indicated with red mark in Fig. 6. Then in Fig. 7, experiments examine the influence of varying *dim* from [8, 16, 32, 64, 128, 256, 512, 1024] with the best θs obtained from Fig. 5. The best accuracy is also signed in red as shown in each figure. We summarize the hyperparameters used in our model in Table 3.

5.5. Results and discussion

The average classification accuracy (standard deviation) over 10-fold cross-validation of GraphCSC (with θ and *dim* that lead to optimal result from Figs. 6 and 7) and baselines on seven real-world datasets are summarized in Table 2. From the results, it is evidently shown that GraphCSC is always the best in terms of performance on 6 datasets with exception on MUTAG, where GraphCSC gets second best result. More specifically, the proposed model achieves 3.41% − 30.60% improvement over graph kernels based methods (WL, Deep WL, GK and Deep GK), and is competitive against graph embedding method (graph2vec) with 3.41% − 30.74% gain in accuracy. However, supervised graph embedding method PSCN is more superior with 4.21% higher in accuracy in MUTAG dataset classification. It is also obvious that GraphCSC outperforms AWE approach in every single dataset. This significantly demonstrates the effectiveness of the proposed model on classification tasks because it has access to skeleton and component features, which enable GraphCSC to get more complete and complex structures information. As for PTC-MR and ENZYMES, GraphCSC and other baselines generally do not perform well. This may due to the fact that these two datasets suffer insufficient data: there are only 344 graphs in PTC-MR and 600 graphs in ENZYMES, while there are 1178 graphs in

Table 2
Classification accuracy (standard deviation) of our method GraphCSC and state-of-the-art baselines on benchmark datasets. The last two rows denote the θ values and the *dim* values used by our method for each dataset; these values are determined in Fig. 5 and Fig. 6. "-" means the classification accuracy (standard deviation) is not available in the original papers.

Algorithm	MUTAG	DD	PTC-MR	NCI-1	NCI-109	PROTEINS	ENZYMES
WL	80.63 (3.07)	77.95 (0.70)	56.97 (2.01)	80.13 (0.50)	80.22 (0.34)	72.92 (0.56)	53.15 (1.14)
Deep WL	82.95 (1.96)	–	59.17 (1.56)	80.31 (0.46)	80.32 (0.33)	73.30 (0.82)	53.43 (0.91)
GK	81.66 (2.11)	78.45 (0.26)	57.26 (1.41)	62.28 (0.29)	62.60 (0.19)	71.67 (0.55)	26.61 (0.99)
Deep GK	82.66 (1.45)	–	57.32 (1.13)	62.48 (0.25)	62.69 (0.23)	71.68 (0.50)	27.08 (0.79)
graph2vec	83.15 (9.25)	58.64 (0.01)	60.17 (6.86)	73.22 (1.81)	74.26 (1.47)	73.30 (2.05)	44.33 (0.09)
PSCN	**92.63 (4.21)**	77.12 (2.41)	60.00 (4.82)	78.59 (1.89)	–	75.89 (2.76)	–
AWE	87.87 (9.76)	71.51 (4.02)	59.14 (1.83)	62.72 (1.67)	63.21 (1.42)	70.01 (2.52)	35.77 (5.93)
GraphCSC	88.42 (6.47)	**89.38 (2.73)**	**64.04 (3.61)**	**85.70 (4.29)**	**85.46 (3.66)**	**76.71 (3.06)**	**57.21 (5.70)**
θ	0.15	0.25	0.75	0.20	0.20	0.60	0.55
dim	128	8	128	16	16	32	128

Table 3
Hyperparameters selected for GraphCSC on different datasets. The first column denotes anonymous random walk length l, sampling hyperparameters ε and δ, *min_sup* threshold θ and embedding dimension *dim*.

Hyperparameter	MUTAG	DD	PTC-MR	NCI-1	NCI-109	PROTEINS	ENZYMES
l	10	10	10	10	10	10	10
ε	1	1	1	1	1	1	1
δ	0.05	0.05	0.05	0.05	0.05	0.05	0.05
θ	0.15	0.25	0.75	0.20	0.20	0.60	0.55
dim	128	8	128	16	16	32	128

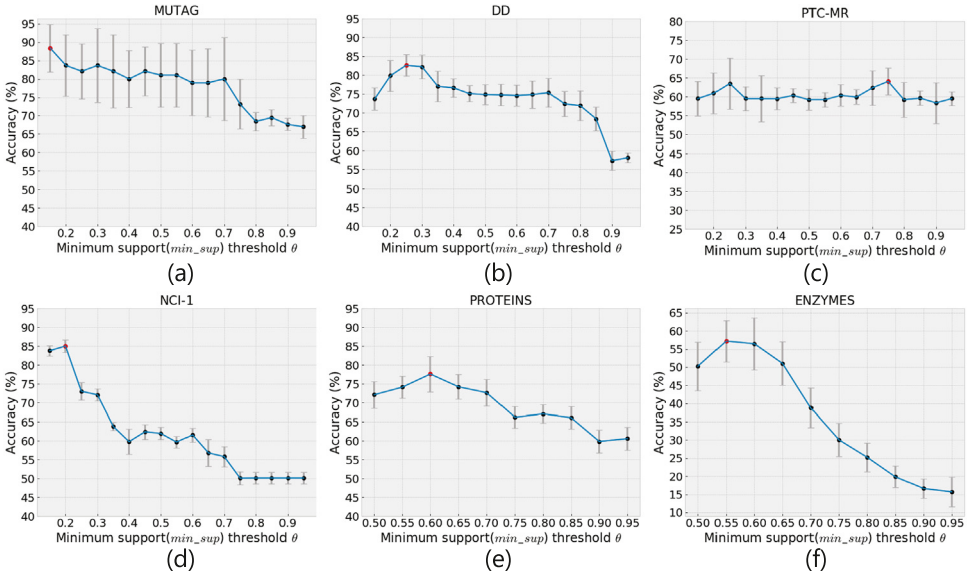

Fig. 6. Here we show parameter sensitivity of *min_sup* threshold θ in graph classification on MUTAG, DD, PTC-MR, NCI-1, PROTEINS and ENZYMES with fixed embedding dimension $dim = 128$. The best expressions marked by red points are used in our experiments and the ($\pm$) standard deviation of each result is indicated with gray error bar.

DD, 1113 graphs in PROTEINS, 4110 graphs in NCI-1 and 4127 graphs in NCI-109. The size of dataset is associated with learning abilities of machine learning models, and insufficient data may cause insufficient training of our proposed approach.

Fig. 6 presents the classification accuracy of increasing *min_sup* threshold θ. In MUTAG and NCI-1, experiments get best performance with $\theta = 0.15$ and 0.2, respectively; then results decrease slowly and maintain stable; after θ reaching 0.7 (for MUTAG) and 0.6 (for NCI-1), lines drop rapidly until getting to another flat states. A common phenomenon is that for DD, PROTEINS and ENZYMES, accuracy lines increase from the beginnings and decrease slowly after meeting the tops. A possible explanation is that low θ will lead to more sufficient frequent subgraphs which will provide adequate complementary structure information for pattern just using skeletons. As a consequence, this enforces distinguishing ability. Another interesting observation is that in PTC-MR, performance shows a relatively stable trend with the growth of θ. The curve has 2 closed top best results when $\theta = 0.25$ and 0.75, while the result at $\theta = 0.75$ has only 1% gain over that at $\theta = 0.25$. This means that *min_sup* threshold θ seems to not have stronger influence on PTC-MR than on other datasets.

Fig. 7 examines the effects of embedding dimension hyperparameter *dim*. As we can see from the figure, in MUTAG and PROTEINS, there is a relatively gentle first-increasing and then-decreasing accuracy line when *dim* increases. We note that for DD, the performance line goes down slowly and maintains stable at around 83% and then again falls quickly. Accuracy in ENZYMES changes roughly, indicating that classification result significantly suffers from embedding dimension *dim*. While it seems that *dim* has no clearly effects on classification results for PTC-MR and NCI-1, the beat *dims* for MUTAG and ENZYMES center at 128. The *dim* values tend to be small (usually no more than 32) in DD, PTC-MR, NCI-1 and PROTEINS, from which we can infer that small *dims* would reflect more obvious features for these 4 datasets.

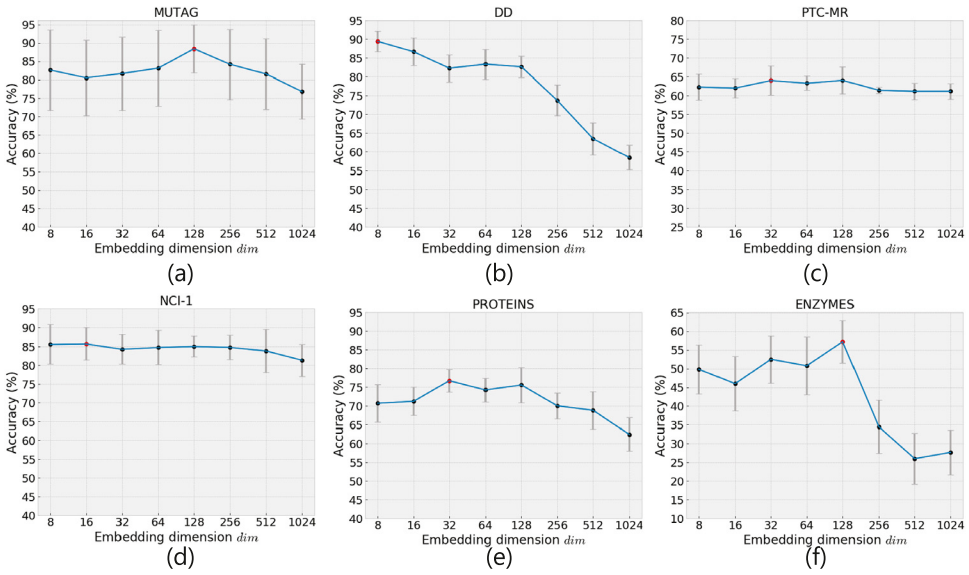

Fig. 7. Parameter sensitivity of embedding dimension *dim* in graph classification on MUTAG, DD, PTC-MR, NCI-1, PROTEINS and ENZYMES with best *min_sup* threshold θ selected in Fig. 6. The best expressions marked by red points are used in our experiments. The ($\pm$) standard deviation of each result is indicated with gray error bar.

6. Conclusion

In this paper, we studied the problem of graph classification and proposed GraphCSC, a new methodology that learns graph representation from horizontal and vertical perspectives to mine graph structures, i.e., skeletons and components. Our model is demonstrated to be superior to approaches which only utilize skeletons. This method treats a graph as a document and different substructures or subgraphs in it as words by **NLP** framework **PV-DBOW**, thus the embeddings GraphCSC learns integrate various size structures information. Several real-world graph classification tasks show that our model can achieve promising performances than a list of state-of-art baselines.

For future work, several interesting problems need to be study further. We tend to focus on more complex graph structures, for example, clusters. Since a cluster usually contains dozens or even hundreds of nodes and edges, it could show more valuable features which may reflect more implicit graph properties. We would also like to investigate graph embedding with mixed structures from low-order subgraphs to large clusters, and to verify whether this would be helpful to get better results than GraphCSC.

CRediT authorship contribution statement

Xue Liu: Conceptualization, Methodology, Writing – original draft. **Wei Wei:** Supervision, Resources. **Xiangnan Feng:** Software, Validation. **Xiaobo Cao:** Writing – review & editing. **Dan Sun:** Visualization, Investigation.

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.

Acknowledgments

This work is supported by the Fundamental Research Funds for the Central Universities, the National Natural Science Foundation of China (No. 11201019), the International Cooperation Project, China No.2010DFR00700, Fundamental Research of Civil Aircraft, China No. MJ-F-2012-04, the Beijing Natural Science Foundation (1192012, Z180005) and National Natural Science Foundation of China (No. 62050132).

References

[1] J.B. Lee, R. Rossi, X. Kong, Graph classification using structural attention, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery &Amp; Data Mining, 2018, pp. 1666–1674.
[2] M. Tsubaki, K. Tomii, J. Sese, Compound–protein interaction prediction with end-to-end learning of neural networks for graphs and sequences, Bioinformatics 35 (2) (2019) 309–318.
[3] L. Backstrom, J. Leskovec, Supervised random walks: predicting and recommending links in social networks, in: The Fourth International Conference on Web Search and Web Data Mining, 2011, pp. 635–644.
[4] F. Costa, K.D. Grave, Fast neighborhood subgraph pairwise distance kernel, in: International Conference on International Conference on Machine Learning, 2010.
[5] A. Grover, J. Leskovec, node2vec: Scalable feature learning for networks, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 855–864.
[6] B. Perozzi, R. Al-Rfou, S. Skiena, Deepwalk: Online learning of social representations, in: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2014, pp. 701–710.
[7] W.L. Hamilton, Graph representation learning, Synth. Lect. Artif. Intell. Mach. Learn. 14 (3) (2020) 1–159.
[8] R. Murphy, B. Srinivasan, V. Rao, B. Ribeiro, Relational pooling for graph representations, in: International Conference on Machine Learning, PMLR, 2019, pp. 4663–4673.
[9] Z. Wu, S. Pan, F. Chen, G. Long, P.S. Yu, A comprehensive survey on graph neural networks, IEEE Trans. Neural Netw. Learn. Syst. PP (99) (2020) 1–21.
[10] M. Zitnik, J. Leskovec, Predicting multicellular function through multi-layer tissue networks, Bioinformatics 33 (14) (2017) i190–i198.
[11] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, M. Sun, Graph neural networks: A review of methods and applications, 2018, ArXiv Preprint ArXiv:1812.08434.
[12] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems, 2012, pp. 1097–1105.
[13] M. Niepert, M. Ahmed, K. Kutzkov, Learning convolutional neural networks for graphs, in: International Conference on Machine Learning, 2016, pp. 2014–2023.
[14] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, in: Proceedings International Conference on Learning Representations, 2017.
[15] S. Abu-El-Haija, B. Perozzi, A. Kapoor, N. Alipourfard, K. Lerman, H. Harutyunyan, G. Ver Steeg, A. Galstyan, Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing, in: International Conference on Machine Learning, PMLR, 2019, pp. 21–29.

[16] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, Graph attention networks, in: Proceedings International Conference on Learning Representations, 2018.
[17] Y. Xie, C. Yao, M. Gong, C. Chen, A. Qin, Graph convolutional networks with multi-level coarsening for graph classification, Knowl.-Based Syst. 194 (2020) 105578.
[18] X. Fan, M. Gong, Y. Xie, F. Jiang, H. Li, Structured self-attention architecture for graph-level representation learning, Pattern Recognit. 100 (2020) 107084.
[19] J. Hartmanis, Computers and intractability: a guide to the theory of NP-completeness (michael r. garey and david s. johnson), Siam Review 24 (1) (1982) 90.
[20] N. Shervashidze, P. Schweitzer, E. Jan, V. Leeuwen, K.M. Borgwardt, Weisfeiler-lehman graph kernels, J. Mach. Learn. Res. 1 (3) (2010) 1–48.
[21] G. Nikolentzos, P. Meladianos, M. Vazirgiannis, Matching node embeddings for graph similarity, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 31 (1), 2017.
[22] C. Morris, K. Kersting, P. Mutzel, Glocalized weisfeiler-lehman graph kernels: Global-local feature maps of graphs, in: 2017 IEEE International Conference on Data Mining (ICDM), IEEE, 2017, pp. 327–336.
[23] H. Kashima, K. Tsuda, A. Inokuchi, Marginalized kernels between labeled graphs, in: Proceedings of the 20th International Conference on Machine Learning (ICML-03), 2003, pp. 321–328.
[24] K.M. Borgwardt, H.-P. Kriegel, Shortest-path kernels on graphs, in: Fifth IEEE International Conference on Data Mining (ICDM'05), IEEE, 2005, pp. 8–pp.
[25] R. Kondor, N. Shervashidze, K.M. Borgwardt, The graphlet spectrum, in: Proceedings of the 26th Annual International Conference on Machine Learning, 2009, pp. 529–536.
[26] N.M. Kriege, F.D. Johansson, C. Morris, A survey on graph kernels, Appl. Netw. Sci. 5 (1) (2020) 1–42.
[27] S. Ivanov, E. Burnaev, Anonymous walk embeddings, in: International Conference on Machine Learning, PMLR, 2018, pp. 2186–2195.
[28] S. Micali, Z.A. Zhu, Reconstructing markov processes from independent and anonymous experiments, Discrete Appl. Math. 200 (2016) 108–122.
[29] Q. Le, T. Mikolov, Distributed representations of sentences and documents, in: International Conference on Machine Learning, 2014, pp. 1188–1196.
[30] T. Mikolov, Distributed representations of words and phrases and their compositionality, Adv. Neural Inf. Process. Syst. 26 (2013) 3111–3119.
[31] A. Mnih, G.E. Hinton, A scalable hierarchical distributed language model, in: Advances in Neural Information Processing Systems, 2009, pp. 1081–1088.
[32] F. Rousseau, E. Kiagias, M. Vazirgiannis, Text categorization as a graph classification problem, in: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), 2015, pp. 1702–1712.
[33] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, K. Borgwardt, Efficient graphlet kernels for large graph comparison, in: Artificial Intelligence and Statistics, 2009, pp. 488–495.
[34] A.R. Benson, D.F. Gleich, J. Leskovec, Higher-order organization of complex networks, Science 353 (6295) (2016) 163–166.
[35] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, U. Alon, Network motifs: simple building blocks of complex networks, Science 298 (5594) (2002) 824–827.
[36] P.-Z. Li, L. Huang, C.-D. Wang, J.-H. Lai, Edmot: An edge enhancement approach for motif-aware community detection, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 479–487.
[37] C. Hong, Towards Accurate and Efficient Classification: A Discriminative and Frequent Pattern-Based Approach, (Ph.D. thesis), University of Illinois at Urbana-Champaign, 2008.
[38] M.J. Zaki, W. Meira, Data Mining and Analysis: Fundamental Concepts and Algorithms, Cambridge University Press, 2014.
[39] X. Yan, J. Han, Gspan: Graph-based substructure pattern mining, in: 2002 IEEE International Conference on Data Mining, 2002. Proceedings, IEEE, 2002, pp. 721–724.
[40] 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, J. Med. Chem. 34 (2) (1991) 786–797.
[41] K.M. Borgwardt, O.C. Soon, S. Stefan, S.V.N. Vishwanathan, A.J. Smola, K. Hans-Peter, Protein function prediction via graph kernels, Bioinformatics 21 (suppl_1) (2005) i47–i56.
[42] P.D. Dobson, A.J. Doig, Distinguishing enzyme structures from non-enzymes without alignments, J. Molecular Biol. 330 (4) (2003) 771–783.
[43] C. Helma, R.D. King, S. Kramer, A. Srinivasan, The predictive toxicology challenge 2000-2001, Bioinformatics 17 (1) (2001) 107–108.
[44] P. Yanardag, S. Vishwanathan, Deep graph kernels, in: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2015, pp. 1365–1374.