Chapter 3
Broad Learning Based on Subgraph Networks for Graph Classification

Jinhuan Wang, Pengtao Chen, Yunyi Xie, Yalu Shan, Qi Xuan, and Guanrong Chen

Abstract Many real-world systems can be naturally represented by networks, such as biological networks, collaboration networks, software networks, social networks, etc., where subgraphs or motifs can be considered as network building blocks with particular functions to capture mesoscopic structures. Most existing studies ignored the interaction between these subgraphs, which could be of particular importance to represent the global structure at the subgraph level. In this chapter, the concept of subgraph network (SGN) is introduced and applied to network models, with algorithms designed for constructing the 1st-order and 2nd-order SGNs, which can be easily extended to build higher-order ones. Furthermore, these SGNs are used to expand the structural feature space of the underlying network, beneficial for network classification. The experiments demonstrate that the structural features of SGNs can complement that of the original network for better network classification. However, SGN model lacks diversity and is of high time-complexity, making it difficult to be widely applied in practice. Then, sampling strategies are introduced into SGNs and a novel sampling subgraph network (S^{2}GN) model is designed, which is scale-controllable and of higher diversity. Further, a broad learning system (BLS) is introduced into graph classification, which fully utilizes the information provided by the S^{2}GNs of different sampling strategies and thus can capture various aspects of the network structure more efficiently. Extensive experiments demonstrate that, by comparing with the SGN model, the S^{2}GN model has much lower time-complexity, which together with BLS can enhance various graph classification methods.

J. Wang · P. Chen · Y. Xie · Y. Shan · Q. Xuan (✉)
Institute of Cyberspace Security, Zhejiang University of Technology, Hangzhou, China

College of Information Engineering, Zhejiang University of Technology, Hangzhou, China
e-mail: xuanqi@zjut.edu.cn

G. Chen
Department of Electrical Engineering, City University of Hong Kong, Hong Kong SAR, China

Q. Xuan et al. (eds.), *Graph Data Mining*, Big Data Management,
https://doi.org/10.1007/978-981-16-2609-8_3

3.1 Introduction

Studying the substructure of a large-scale network, e.g., its subgraphs, is an efficient way to understand and analyze the network. Recently, a number of studies on network subgraphs for various network applications have been reported. Ugander et al. [1] treated subgraph frequency as a local property in social networks and found that subgraph frequency can provide deep insights for identifying both social structure and graph structure in a large network. Except for subgraph frequency statistics, Benson et al. [2] developed a corresponding embedding representation through Laplacian matrix analysis method. Moreover, Wang et al. [3] designed an incremental subgraph join feature selection algorithm, which forces graph classifiers to join short-pattern subgraphs so as to generate long-pattern subgraph features. Deep learning methods for graphs achieve remarkable performance on many network analysis tasks. Yang et al. [4] proposed a NEST method, which combines the motifs and convolutional neural network. Recently, Alsentzer et al. [5] introduced a SUB-GNN to learn disentangled subgraph representations by embedding subgraphs into the GNNs, which achieves considerable performance gains on subgraph classification.

The studies mentioned above try to reveal subgraph-level patterns, which can be considered as network building blocks with particular functions, to capture mesoscopic structures. However, most of them ignored the interactions among these subgraphs, which could be of particular importance to represent the global structure at the subgraph level. In order to address this issue, Xuan et al. [6] proposed a method to establish *Subgraph Networks* (SGNs) of different orders. It is expected that such SGNs can capture the structural features in different aspects and thus may benefit the follow-up tasks, such as network classification. The details of constructing SGNs will be further discussed in Sect. 3.3. Conceptually, the SGN extracts the representative parts of the original network and then assembles them to reconstruct a new network that preserves the relationship among subgraphs. Thus, this method implicitly maintains the higher-order structures while preserving the information of local structures.

Notably, the network structure of SGN can complement the original network and the integration of SGNs' features will benefit the subsequent structure-based algorithms design and applications. However, this model can be further improved. It is observed that the rule to establish SGN is deterministic, i.e., users can generate only one SGN of each order for a network. This lack of diversity will limit the capacity of SGN to expand the latent structure space. Besides, when the number of subgraphs exceeds the number of nodes in a network, the generated SGN can be even larger than the original network, which makes it extremely time consuming to process higher-order SGNs, hindering further applications of the SGN algorithm. On the other hand, it is noted that network sampling can increase the diversity by introducing randomness, and meanwhile control the scale, providing an effective and inexpensive solution for network analysis. This feature is complementary to the SGN model. Wang et al. thus introduce *Sampling Subgraph Networks* (S^2GNs) through combining sampling strategy and SGN.

Broad Learning System (BLS) [7] is a single-layer incremental neural network, which has a good performance in training speed and classification accuracy therefore offers an alternative way of learning in deep structure. In this chapter, BLS will be adopted to fully utilize the structural information captured by S^{2}GN, so as to enhance the performance of graph classification. The experiments demonstrate the effectiveness of the method. The main contents of this chapter are summarized as follows:

- SGN and S^{2}GN are utilized to expand the structural feature space, which provides more significant and potential features for the analysis of the original network and benefits the associated algorithms.
- The broad learning system is employed for the first time to fully utilize the structural information extracted from S^{2}GNs generated by different sampling strategies, to enhance various graph classification algorithms based on Graph2Vec and CapsuleGNN.
- The new models are tested on three real-world network datasets, and the experimental results demonstrate that S^{2}GN together with BLS can indeed significantly improve the algorithm efficiency.

The rest of this chapter is organized as follows. In Sect. 3.2, some related work about subgraph networks, network representation, and the broad learning system are introduced. In Sect. 3.3, the SGN is described and the algorithms for constructing the 1st-order and 2nd-order SGNs are designed. In Sect. 3.4, three sampling strategies are developed and the construction methods of S^{2}GN are presented. In Sect. 3.5, BLS is used as the classifier for the classification framework. In Sect. 3.6, two feature extraction methods are introduced, and then combined with SGNs and S^{2}GNs, which are then applied to classify graphs in three real-world datasets. In Sect. 3.7, the computational complexities of SGNs and S^{2}GNs models are analyzed and compared. Finally, Sect. 3.8 concludes the chapter, with a future research outlook.

3.2 Related Work

In this section, some necessary background information is provided on subgraph networks and graph representation methods in graph mining and network science, with a brief overview of related research on the broad learning system.

3.2.1 Subgraph Networks

Subgraph is a key component in complex networks and graph mining. The structural interaction among subgraphs also plays an important role in network analysis. Subgraph network (SGN) [6] is the first model to introduce the notion of subgraph interaction, which can capture the latent high-order structural features in the original network. However, its construction is of high time-complexity. On the other hand, network sampling is an important part of network mining. Sampling methods in graph mining have two main tasks: generating node sequences for subsequent network representation [8–10] and limiting the scale of the network to simplify graphs and achieve faster graph algorithms [11, 12]. Sampling methods can simplify the network while preserving significant structural information, which is of extreme importance in graph mining. In view of this, as a variant of the SGN, sampling subgraph network (S^2GN) combined with different sampling strategies was introduced, which can enhance the performance of graph classification and reduce the time complexity of the SGN. In this chapter, SGN and S^2GN are utilized to expand the structural space for enhancing the performance of graph classification.

3.2.2 Network Representation

Network representation has received considerable attention in recent years, which allows the relational knowledge of interacting entities to be stored and accessed efficiently. The most naive network representation method is to calculate graph attributes according to certain typical topological metrics [13]. Early graph embedding methods were significantly affected by Natural Language Processing (NLP). For example, as graph-level embedding algorithms, Narayanan et al. developed Subgraph2Vec [14] and Graph2Vec [15], which achieve good performances on graph classification. Graph kernel methods [16, 17] are popular tools to capture the similarity between graphs where the kernel is equivalent to an internal product in the associated feature space. Although representing networks well, they generally have relatively high computational complexity [13], which makes it unrealizable to process large-scale networks. Graph Convolutional Networks (GCN) process the obtained information without weighting, i.e., the information of important neighbors and non-important neighbors will be put into the convolution layer in an unbiased manner. Later, Graph Attention Networks (GAT) [18] overcome this shortage by supplementing a self-attention coefficient before the convolution layer. Based on the newly proposed capsule network architecture, Zhang et al. [19] designed a CapsuleGNN to generate multiple embeddings for each graph, thereby capturing the classification-related information and the potential information with respect to the graph properties at the same time, which achieved good performance.

3.2.3 Broad Learning System

Broad Learning System (BLS) [7, 20, 21] is a single-layer incremental neural network based on the random vector function-link neural network (RVFLNN), which aims to offer an alternative way of learning in deep structure. Chen et al. [7]

showed that BLS outperforms the existing deep structure neural networks in terms of training speed. Indeed, compared with other multi-layer perceptron (MLP) training methods, BLS has a promising performance in classification accuracy and learning speed. In view of this advantages, BLS has found many applications in various fields. For example, Gao et al. [22] presented an incremental BLS for event-based object classification, which demonstrated that increasing the broad network by adding feature nodes and enhancement nodes is effective for asynchronous event-based data and provides an alternative way to deal with neuromorphic cameras. Chen et al. [23] designed a deep-broad learning system for traffic flow prediction, which increases the accuracy of traffic flow prediction, and maintains low complexity and running time. To date, BLS has been widely applied in the filed of computer vision but rarely in graph data mining. In this chapter, BLS is used for network analysis task in combination with S^2GNs. It will be shown that BLS achieves good performances in graph classification.

3.3 Subgraph Networks

To be self-contained, SGN is first reviewed, followed by the 1st-order SGN (SGN$^{(1)}$) and 2nd-order SGN (SGN$^{(2)}$) construction algorithms.

Subgraph network (SGN) maps the links in the original network into the nodes in the SGN, thereby transforming the node-level original network into a subgraph-level network.

Definition 1 (Network) Let $G(V, E)$ be an undirected network, where V and $E \subseteq (V, V)$ respectively denote the nodes and links in the network. The element $(v_i, v_j) \in E$ denotes an unordered pair of nodes v_i and v_j, i.e.,$(v_i, v_j) = (v_j, v_i)$, for $i, j = 1, 2, 3, \ldots, N$, where N is the number of nodes in the network.

Definition 2 (Subgraph) For a network $G(V, E)$ and a subgraph $g_i = (V_i, E_i)$, where $g_i \subseteq G$ if and only if $V_i \subseteq V$ and $E_i \subseteq E$. Denote the sequence of subgraphs as $g = \{g_i \subseteq G | i = 1, 2, \ldots, n\}, n \leq N$.

Definition 3 (Subgraph Network) Consider an undirected network $G(V, E)$, and a SGN $G^* = f(G)$, which is a mapping from G to $G^*(V^*, E^*)$, with the nodes and links denoted by $V^* = \{g_i | j = 0, 1, \ldots, n\}$ and $E^* \subseteq (V^*, V^*)$, respectively. Two subgraphs g_i and g_j are connected if they share some common nodes or links in the original graph, i.e., $V_i \cap V_j \neq \emptyset$ or $E_i \cap E_j \neq \emptyset$. Moreover, an element $(g_i, g_j) \in E^*$ is an unordered pair of subgraphs g_i and g_j, i.e., $(g_i, g_j) = (g_j, g_i), i = 1, 2, \ldots, n$, with $n \leq N$.

According to the above definitions [6], one can actually find that SGN is derived from a higher-order mapping of the original network. Agarwal et al. [24] discussed the problem of graph representation in the domain with higher-order relations, where the node set is constructed as a p-chain, corresponding to points (0 chain), lines (1 chain), triangles (2-chain) and so on. Here, similarly, the SGN constructs the subgraphs as 1st-order, 2st-order, etc. For clarity, the following are three steps in building a SGN:

- **Extract subgraphs.** The first step is extracting subgraphs from the original network. The network has rich subgraph structures, some of which appear frequently, such as motifs [25].
- **Choose subgraph blocks.** The second step is choosing appropriate subgraph blocks. Generally, a subgraph should not be too large, otherwise the SGN may only contain a very small number of nodes, which makes subsequent analysis less significant.
- **Construct the SGN.** The final step is constructing the SGN by utilizing the subgraph blocks. After extracting enough subgraphs from the original network, the key issue is to define rules for building SGN and establish connections between these blocks. Here, for simplicity, consider two subgraphs, which are connected if they share the same node or link in the original network.

In this chapter, the most basic subgraphs (that is, lines and open triangles) are selected as subgraphs because they are simple and common in most networks.

3.3.1 First-Order SGN

From the 1st-order SGN, denoted as SGN$^{(1)}$, one can select a line namely a link as the subgraph to construct the SGN. The 1st-order SGN is also called a line graph [26].

The process of constructing SGN$^{(1)}$ from a given network is shown in Fig. 3.1. In this example, the original graph has 6 nodes connected by 6 links. First, one extracts lines as subgraphs and labels them with their corresponding terminal nodes. Then, one treats these lines as nodes in SGN, and connects them according to their labels, i.e., two lines are connected if they share the same terminal node, as shown in Fig. 3.1b. Finally, one obtains the structure of SGN with 6 nodes and 9 links as

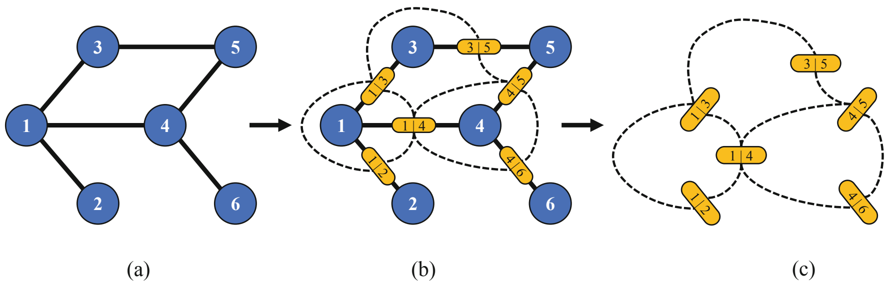

Fig. 3.1 The process to build SGN$^{(1)}$ from the original network: (**a**) the original graph, (**b**) extracting lines and establishing connection among these lines, (**c**) the structure of SGN$^{(1)}$

Algorithm 1: Constracting first-order SGN

```
   Input: A network $G(V, E)$ with node set $V$ and link set $E \subseteq (V \times V)$.
   Output: First-order SGN, denoted by $G_1(V_1, E_1)$.
1  Initialize a node set $V_1$ and a link set $E_1$;
2  for each node $u \in V$ do
3      Obtain the neighbors set $\Gamma$ of u;
4      for each node $v \in V$ do
5          A temporary link $L$ = sorted pair of nodes set$(u, v)$;
6          Regard link $L$ as a new node in the first-order SGN;
7          Append $L$ to node set $\widehat{V}$;
8      end
9      for $i, j \in \widehat{V}$ and $i \neq j$ do
10         Append the link $(i, j)$ to $E_1$;
11     end
12     Append $\widehat{V}$ to $V_1$;
13 end
14 return $G_1(V_1, E_1)$;
```

shown in Fig. 3.1c. The pseudocode for constructing SGN$^{(1)}$ is given in Algorithm 1. The input of the algorithm is the original network $G(V, E)$, and the output is the constructed SGN$^{(1)}$, denoted as $G_1(V_1, E_1)$, where V_1 and E_1 represent the nodes and links in SGN$^{(1)}$, respectively.

3.3.2 Second-Order SGN

Compared with lines, triangles can provide more insights about the local structure of the network. For example, Schiöberg et al. [27] studied the evolution of triangles in the Google+ online social network, where some valuable information was obtained during the appearance and pruning of various triangles.

Now, construct a higher-order subgraph by considering the connection pattern between three nodes. Compared with two nodes, the connection mode between three nodes is more diverse. Here, only the connected subgraphs are considered, and the subgraphs with less than two links are ignored. Here, the open triangle is defined as a subgraph to establish a 2nd-order SGN, denoted by SGN$^{(2)}$. Second order means that there are two links in each open triangle, and if two open triangles share the same link, the two open triangles are connected in SGN$^{(2)}$. Note that the same link instead of the same node is used here to avoid getting a very dense SGN$^{(2)}$. This is because, usually, dense networks with higher connection probability of each node pair tend to provide less discriminative information for local structures.

The construction process from SGN$^{(1)}$ to SGN$^{(2)}$ is illustrated by Fig. 3.2. In the line graph SGN$^{(1)}$, further extract lines to obtain hollow triangles as subgraphs, and mark them with the corresponding three nodes, as shown in Fig. 3.2b. Finally, an SGN$^{(2)}$ with 8 nodes and 15 links is obtained, as shown in Fig. 3.2c. The pseudocode

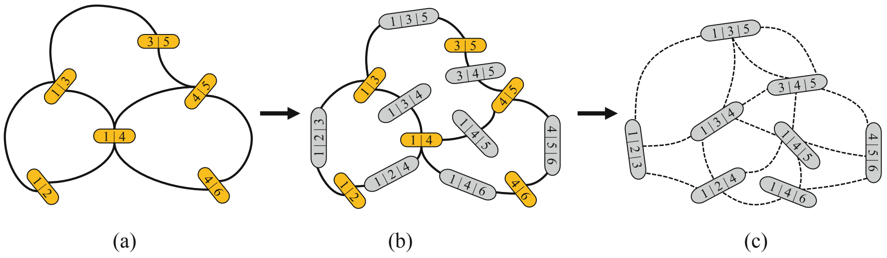

Fig. 3.2 The process to build SGN$^{(2)}$ from the first-order subgraph network: (**a**) SGN$^{(1)}$ in Fig. 3.1, (**b**) extracting lines and establishing connections among these lines, (**c**) the structure of SGN$^{(2)}$

Algorithm 2: Constracting second-order SGN

```
Input: A network $G(V, E)$ with node set $V$ and link set $E \subseteq (V \times V)$.
Output: Second-order SGN, denoted by $G_2(V_2, E_2)$.
1  Initialize a node set $V_2$ and a link set $E_2$;
2  for each node $u \in V$ do
3      Obtain the neighbors set $\Gamma$ of u;
4      The set of all node pairs in the neighbor collection $\widehat{\Gamma}$;
5      for each node pair $(v_1, v_2) \in \widehat{\Gamma}$ do
6          A temporary link $L$ = sorted pair of nodes set$(u, v_1, v_2)$;
7          Regard link $L$ as a new node in the first-order SGN;
8          Append $L$ to node set $\widehat{V}$;
9      end
10     for $i, j \in \widehat{V}$ and $i \neq j$ do
11         Append the link $(i, j)$ to $E_2$;
12     end
13     Append $\widehat{V}$ to $V_2$;
14 end
15 return $G_2(V_2, E_2)$;
```

for constructing SGN$^{(2)}$ is given in Algorithm 2. The input of the algorithm is the original network $G(V, E)$, and the output is the constructed SGN$^{(2)}$, represented by $G_2(V_2, E_2)$, where V_2 and E_2 represent the node and link sets in SGN$^{(2)}$, respectively.

As SGN gradually maps to the higher-order network, one can obtain more and richer feature information. SGN$^{(1)}$ can reveal the topological interaction between the links of the original network. Fu et al. [28] adopted SGN to predict the link weights of given networks. SGN$^{(2)}$ is obtained by further iterative mapping based on SGN$^{(1)}$, so the second-order information of the nodes can be captured. Higher-order SGNs will contain more hidden information, but these hidden information may play a smaller role in subsequent applications. Therefore, here the focus is on the SGNs of the first two orders.

3.4 Sampling Subgraph Networks

Next, the S^{2}GN is reviewed. S^{2}GN is proposed as a variant of SGN through introducing sampling strategies into the SGN algorithm. In this section, several sampling strategies are described, and the construction of S^{2}GN is discussed.

3.4.1 Sampling Strategies

Network sampling can simplify a graph while preserving its significant structural information, which is of extreme importance in graph data mining. Here, three sampling algorithms are reviewed: biased walk, spanning tree and forest fire.

3.4.1.1 Biased Walk (BW)

Biased walk sampling is a very common sampling strategy. Here, the walking mechanism of Node2Vec [29] is adopted to preserve the homogeneity and structure of nodes by integrating depth-first search (DFS) and breadth-first search (BFS) (as shown in Fig. 3.3a). In the mechanism of Node2Vec, it defines a 2nd-order random walk, which is guided by the two parameters p and q (as shown in Fig. 3.3b). To start, assume that one walks from node A to node B and then needs to determine the next step. The transition probability P from node B to node C is then defined as

$$P_{(B,C)} = f_{pq}(A, C) = \begin{cases} \frac{1}{p}, d_{(A,C)} = 0 \\ 1, d_{(A,C)} = 1 \\ \frac{1}{q}, d_{(A,C)} = 2 \end{cases}$$

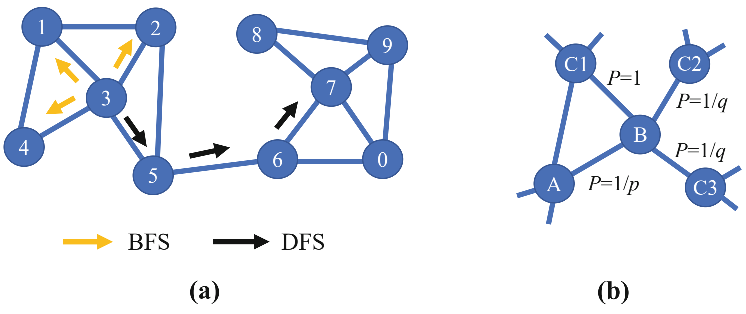

Fig. 3.3 (**a**) BFS and DFS walk strategies from node 3. (**b**) Illustration of evaluating the next step out of node B. Edge labels indicate search biases P

where $d_{A,C}$ represents the shortest distance between nodes A and C, and its value must be in {0, 1, 2}. The pseudocode for biased walk is given in Algorithm 3.

Algorithm 3: Biased walk sampling

```
   Input: A network $G(V, E)$ with node set $V$ and link set $E \subseteq (V \times V)$.
   Output: The substructure $G_b(V_b, E_b)$.
1  Initialize a source node $v_0 \in V$, added into $V_b$;
2  Append $v_0$ to $V_b$;
3  while walk length $L$ do
4      $v = V_b$[-1];
5      $V_v$ = GetNeighbors($v, G$);
6      $v_{next}$ = AliasSample($V_v, \pi$);
7      Append $v_{next}$ to $V_b$;
8      Append $(v, v_{next})$ to $E_b$;
9  end
10 return $G_b(V_b, E_b)$;
```

3.4.1.2 Spanning Tree (ST)

A spanning tree [30] is defined as a subgraph of a connected tree graph G, which connects all the nodes together with minimum possible number of edges. Here, since the datasets used are all unweighted networks, the largest spanning tree is equivalent to the smallest spanning tree. In this chapter, the classical Kruskal algorithm [31] is used to generate spanning trees (as shown in Fig. 3.4) and the weight values of the links are all set to one.

The Kruskal algorithm is a greedy algorithm for generating spanning tree as follow:

- Step 1. Create a set of trees $\mathscr{F}$ where each node in the graph belongs to a tree;
- Step 2. Create a set of edges $\mathscr{E}$ including all edges of the graph;

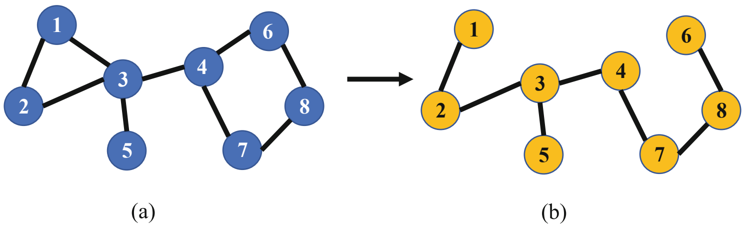

Fig. 3.4 Obtaining a spanning tree. (**a**) Original network. (**b**) Spanning tree structure

- Step 3. While $\mathscr{E} \neq \emptyset$ and $|\mathscr{F}| \neq 1$
  - select any edge from $\mathscr{E}$;
  - if this edge connects two different trees, then combine it with the two trees to generate a new tree and add it into $\mathscr{F}$;
  - otherwise discard this edge;
- Step 4. The $\mathscr{F}$ has a minimum spanning tree at the termination of the iteration.

3.4.1.3 Forest Fire (FF)

Forest fire sampling was first proposed by Leskovec and Faloutsos in 2006 [11]. Here, a specific algorithm is introduced for forest fire sampling. Given a network, first randomly select a node v_0, and then generate a random number X, following the geometric distribution with mean $p_f/(1-p_f)$. Here, the parameter p_f is called the forward burning probability (set to 0.2 [11]). Then, X edges with node v_0 as terminal node will be selected, where the another terminal node of each edge has not been visited. One can use any method of generating random numbers to find the unvisited nodes with the current burning nodes as source nodes, one by one, until enough nodes are burned. To avoid duplication, nodes cannot be visited twice during the forest fire sampling method. If the fire dies, select a node randomly to restart again. The pseudocode for forest fire sampling is given in Algorithm 4.

Algorithm 4: Forest fire sampling

```
   Input: A network $G(V, E)$ with node set $V$ and link set $E \subseteq (V \times V)$, the forward burning
          probability $p_f$.
   Output: The substructure $G_s(V_s, E_s)$.
 1 Initialize a neighbor list $N$ and a temporary variable $G_s(V_s, E_s)$;
 2 Randomly choose the first node $v_0$;
 3 Generate $X$ following the geometric distribution with mean $p_f/(1-p_f)$;
 4 $n$ = The number of $v_0$' neighbor;
 5 if $X \leq n$ then
 6     N← Sort the neighbors of $v_0$ according to the degree and choose the top $X$ neighbors;
 7 end
 8 for node $T$ in $N$ do
 9     if $T$ in $V_s$ then
10         continue;
11     else
12         Append the node $T$ to $V_s$;
13         Append the link $(T, v_0)$ to $E_s$;
14         Forest fire recursive function($G, p_f, T$);
15     end
16 end
17 return $G_s(V_s, E_s)$;
```

Using any of the above three sampling strategies, one can map the original network into many substructures. As a result, more characteristic information in the network can be abstracted, which also provides favorable preconditions for downstream algorithms.

3.4.2 Construction of S^{2}GN

Most networks in the real world have complex structures. Therefore, the generated SGNs are typically of large-scale and even denser than the original networks. This will not only reduce the efficiency of the algorithms, but also introduce some "noise" into structures, which will reduce the accuracy of the algorithms. In view of this, the original SGN model is optimised to construct S^{2}GN. In particular, multiple sampling strategies are introduced, filtering the original complex network as substructures, thereby establishing a new SGN. The pseudocode for constructing S^{2}GN is given in Algorithm 5. Generally, the S^{2}GN algorithm is divided into three parts: selecting source node, sampling substructure and constructing subgraph network. The algorithm steps are described as follows:

- **Selecting source node.** There are many ways to select the source node: (i) randomly selecting a node as the source node; (ii) selecting an initial node according to the importance of the node. In this chapter, the second method is adopted to better capture the key structure of a network.
- **Sampling substructures.** After determining the initial source node, a substructure can be obtained by performing a certain sampling strategy to extract the main context of the current network. According to different sampling strategies, various sampling substructures can be generated, and the rich structural information in the current network can be obtained.
- **Constructing S^{2}GN.** The sampling substructure is used as input to construct a subgraph network. Note that sampling and SGN construction are repeated interactively, so as to obtain higher-order S^{2}GNs. That is, each time the current network is mapped to a subgraph network, and then a sampling operation is performed to obtain various relatively simple sampling substructures.

3.5 BLS Classifier Based on S^{2}GN

3.5.1 BLS Classifier

BLS [7, 20] is proposed as an alternative method of deep learning network. It is designed such that mapping features are input into RVFLNN. As shown in Fig. 3.5, a specific illustration of BLS is given, which will be used as the classifier of the network.

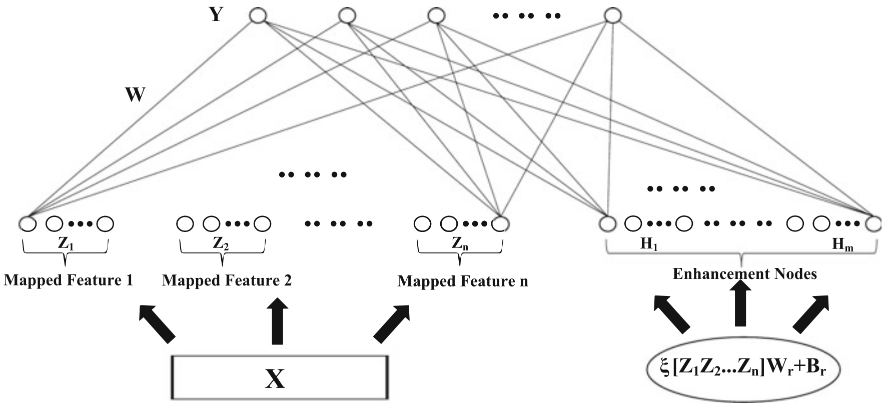

Fig. 3.5 The framework of BLS classifier

Algorithm 5: Constracting sampling subgraph network

```
   Input: A network $G(V, E)$ with node set $V$ and link set $E \subseteq (V \times V)$, the order of SGN $T$,
          sampling strategy $f_s(\cdot)$, sampling walks $L$.
   Output: Sampling subgraph network, denoted by $G'(V', E')$.
 1 Initialize a temporary variable $G' = G$;
 2 while $T$ do
 3     $G'$ = GetMaxSubstracture($G'$);
 4     Source node $o$ = NodeRanking($V'$);
 5     Initial sampling $v_i = o, W_v \leftarrow [o], W_e \leftarrow \emptyset$ ;
 6     for $i = 1$ to $L - 1$ do
 7         Sampling link $e_i = f_s(v_i)$;
 8         Update current node $v_i = dst(e_i)$;
 9         Append $v_i$ to $W_v$, $e_i$ to $W_e$;
10     end
11     $V_s \leftarrow W_v$, $E_s \leftarrow W_e$;
12     $G_{sgn}$ =SGN Algorithms($G_s$);
13     $G' \leftarrow$ Relabel ($G_{sgn}$);
14     $T = T - 1$;
15 end
16 return $G'(V', E')$;
```

In this process, we treat the training features as the graph input X, and then transform X into n random feature spaces by feature mapping:

$$Z_i = \phi(XW_{z_i} + \beta_{z_i}), i = 1 \dots n \tag{3.1}$$

where the weights W_{z_i} and the bias term β_{z_i} are generated randomly with appropriate dimensions, n is the number of groups of mapped features, and $\phi(\cdot)$ indicates the linear mapping.

Here, denote the feature space of training samples by $Z^n = [Z_1, Z_2, \dots, Z_n]$. Then, the jth group of enhancement nodes is defined by

$$H_j = \xi(Z^n W_{r_j} + B_{r_j}),\ j = 1, 2, \dots, m \tag{3.2}$$

where W_{r_j} is the enhancement weight, B_{r_j} is the bias term, and $\xi(\cdot)$ is a nonlinear activation function.

Similarly, denote the enhancement layer by $H^m = [H_1, H_2, \dots, H_m]$. Thus, the form of output $\hat{Y}$ is as follows:

$$\hat{Y} = [Z^n, H^m]W = AW \tag{3.3}$$

where $A = [Z^n, H^m]$ are the features, combining the enhancement nodes and feature nodes, and W is the weight matrix that connects the feature nodes and enhancement nodes to the output layer. The W should be optimized by

$$min_W ||Y - AW||_2^2 + \lambda ||W||_2^2 \tag{3.4}$$

where λ is a regularization coefficient. Then, through a simple equivalent transformation [7], one finally gets the following formula:

$$W = (A^T A + \lambda I)^{-1} A^T Y \tag{3.5}$$

Now, one has the trained model with the weight matrix W, therefore can test its performance using the rest of the dataset. In fact, there are several hyperparameters in this model, such as the number of feature nodes k, the number of enhancement nodes m, the regularization coefficient λ and the shrink coefficient s of $\xi(\cdot)$. In the experiment, the regularization coefficient λ is set to 2^{-10} and shrink coefficient s to 0.8. For MUTAG, PTC, and PROTEINS, set the number of feature nodes k as 50, 80, and 100, and the number of enhancement nodes m as 40, 60, and 80, respectively. For different samples, the optimal parameters are set near the above parameters.

3.5.2 Classification Framework

According to the above setting, one can design a framework for graph classification by combining S^{2}GN with the BLS classifier, as shown in Fig. 3.6.

To begin with, construct three S^{2}GNs according to the method described in Sect. 3.4.2, i.e., S^{2}GN$^{(0)}$, S^{2}GN$^{(1)}$ and S^{2}GN$^{(2)}$, and then map them into different feature spaces. By graph feature extraction methods, one can get the feature representations of S^{2}GNs and then fusion them into $X = [X_0||X_1||X_2]$, where $[a||b]$ means to merge vector a and vector b. The fused vector X can be regarded as the input of the BLS classifier as shown in Fig. 3.5. Here, the same sampling strategy and feature extraction method are adopted for conducting classification. Since the

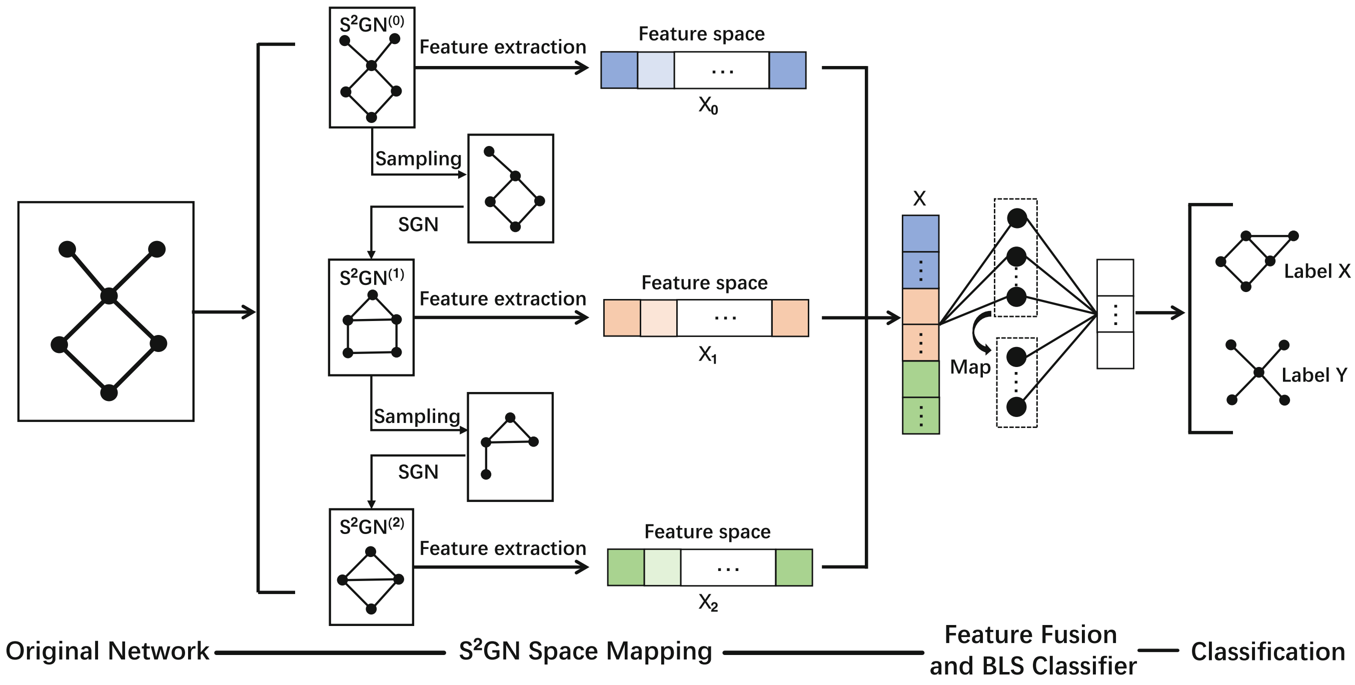

Fig. 3.6 The overall framework with BLS of the S^2GN algorithm for graph classification

same original network can generate various S^{2}GNs by different sampling strategies, X contains abundant structural information of different aspects. Therefore, this framework combining S^{2}GNs with BLS can enhance the performance of the original network classification.

3.6 Experiment

3.6.1 Graph Classification

The performance of the above framework on graph classification is evaluated by simulation. Graph classification is one of the most important data mining tasks, which has been widely used in the field of biochemistry, such as protein classification and molecular toxicity classification. Typically, graph classification focuses on transforming discrete graphs into numerical features. One can use some machine learning algorithms to effectively classify various graphs.

Consider graph $G = (V, E)$ from the set $\mathscr{G} = \{G_i\}$, where $i = 1, 2, 3 \ldots, N$. The node and edge collections are $V = \{v_1, v_2, v_3, \ldots, v_n\}$ and $E = \{e_1, e_2, e_3, \ldots, e_n\} \subseteq (V \times V)$, respectively. Each graph G has a corresponding category label $y \in \mathscr{C}$, where $\mathscr{C} = \{1, 2, 3, \ldots, k\}$ is the set containing k different labels. The aim of graph classification is to find a mapping function $f : \mathscr{G} \to \mathscr{C}$ to predict the label of each graph in $\mathscr{G}$. Generally, one can train the model using the training set with known category labels and evaluate its performance using the test set with unknown labels. By comparing the output $\widehat{y} = f(G)$ with the true test label y, one can evaluate the classification algorithm by accuracy.

To date, a large number of graph classification methods have been proposed, like graph embedding method Graph2Vec and deep learning method CapsuleGNN. In this section, a brief introduction of these two network representation methods is given, and then some experiments on three datasets are performed for evaluating the performances of both SGN and S^{2}GN models.

3.6.2 Datasets

Three datasets, MUTAG, PTC and PROTEINS, will be used for graph classification experiments. The basic statistics of these datasets are presented in Table 3.1.

- MUTAG [32] dataset is about heteroaromatic nitro and mutagenic aromatic compounds, with nodes and links representing atoms and the chemical bonds between them, respectively. They are labeled according to whether there is a mutagenic effect on a special bacteria.
- PTC [33] dataset includes 344 chemical compound graphs whose labels are determined by their carcinogenicity for rats.

Table 3.1 Basic statistics of the 3 datasets. #Graphs is the number of graphs. #Classes is the number of classes. #Positive and #Negative are the numbers of graphs in the two different classes

Dataset	#Graphs	#Classses	#Positive	#Negative
MUTAG	188	2	125	63
PTC	344	2	152	192
PROTEINS	1113	2	663	450

- PROTEINS [34] dataset comprises of 1113 graphs. The nodes are Secondary Structure Elements (SSEs) and the links are neighbors in the amino-acid sequence or in the 3D space. These graphs represent either enzyme or non-enzyme proteins.

3.6.3 Network Representation

Network representation is a method of mapping graphs into vectors while retaining as many topological features as possible. Here, two network representation methods are used to extract graph features, including graph embedding method Graph2Vec and deep learning method CapsuleGNN.

- **Graph2Vec**: This is the first unsupervised embedding approach for an entire network, which is based on the extending word-and-document embedding technique that has shown great advantages in NLP. Graph2Vec establishes the relationship between a network and the rooted subgraphs using a similar model to Doc2Vec. Graph2Vec first extracts rooted subgraphs and provides corresponding labels into the vocabulary, and then trains a skipgram model to obtain a representation of the entire network.
- **CapsuleGNN**: This method is inspired by CapsNet [35], which utilizes the concept of capsules to overcome the shortcoming of existing GNN-based graph representation algorithms. CapsuleGNN extracts node features in the form of capsules and uses routing mechanisms to capture important information at the graph level. The model generates multiple embeddings for each graph in order to capture graph properties from different aspects.

For *Graph2Vec*, the embedding dimension is adopted following [15]. Graph2vec is based on the rooted subgraphs adopted in the WL kernel. The parameter height of the WL kernel is set to 3. Since the embedding dimension is predominant for learning performances, a commonly-used value of 1024 is adopted. The other parameters are set to defaults: the learning rate is set to 0.5, the batch size is set to 512 and the epochs is set to 1000. The default parameters are used for *CapsuleGNN* and the multiple embeddings of each graph are flattened as the input.

3.6.4 SGN for Graph Classification

As described in Sect. 3.3, the proposed SGNs can be used to expand structural feature spaces. In order to study the effectiveness of the 1st-order and 2nd-order SGNs, i.e., SGN$^{(1)}$ and SGN$^{(2)}$, the classification results were compared based on different numbers of networks, i.e., SGN$^{(0)}$, SGN$^{(1)}$, SGN$^{(2)}$ SGN$^{(0,1)}$, SGN$^{(0,2)}$ and SGN$^{(0,1,2)}$. Without loss of generality, the well-known logistic regression is chosen as the classification model. Meanwhile, for each feature extraction method, the feature space is first expanded by using SGNs, and then the dimension of the feature vectors is reduced to the same value as that of the feature vector obtained from the original network using PCA in the experiments, for a fair comparison. Each dataset is randomly split into ninefolds for training and onefold for testing. Here, the F_1-$Score$ is adopted as the metric to evaluate the classification performance:

$$\mathscr{F} = \frac{2\mathscr{P}\mathscr{R}}{\mathscr{P} + \mathscr{R}}, \tag{3.6}$$

and the Gain can be calculated by

$$Gain = \frac{\mathscr{F}^{(0,1,2)} - \mathscr{F}^{(0)}}{\mathscr{F}^{(0)}} \times 100\% \tag{3.7}$$

where $\mathscr{P}$ and $\mathscr{R}$ are the precision and recall, respectively. To exclude the random effect of the fold assignment, experiment is repeated for 500 times and then the average F_1-$Score$ and its standard deviation are recorded. The results for Graph2Vec and CapsuleGNN methods are shown in Tables 3.2 and 3.3, respectively.

From Tables 3.2 and 3.3, one can see that the original network appears to provide more structural information. The classification model based on SGN$^{(0)}$ performs better, with a higher F_1-$Score$, than those based on SGN$^{(1)}$ or SGN$^{(2)}$. This is reasonable because there is information loss in the processes of constructing SGNs. More interestingly, the performance of the classification model based on two networks i.e., SGN$^{(0,1)}$ and SGN$^{(0,2)}$, is better than the classification model based on

Table 3.2 Classification results on MUTAG, PTC and PROTEINS, in terms of F_1-$Score$, based on Graph2Vec method with the combinations of SGNs of different orders, the bold values are the best results

Dataset	MUTAG	PTC	PROTEINS
Original	83.15 ± 9.25	60.17 ± 6.86	73.30 ± 2.05
SGN$^{(1)}$	63.16 ± 4.68	56.80 ± 5.39	60.27 ± 2.05
SGN$^{(2)}$	68.95 ± 8.47	57.35 ± 3.83	59.82 ± 4.11
SGN$^{(0,1)}$	83.42 ± 5.40	59.03 ± 3.36	74.12 ± 1.57
SGN$^{(0,2)}$	81.32 ± 3.80	61.76 ± 3.73	73.09 ± 1.28
SGN$^{(0,1,2)}$	$\mathbf{86.84 \pm 5.70}$	$\mathbf{63.24 \pm 6.70}$	$\mathbf{74.44 \pm 3.09}$
Gain	4.44%	5.10%	1.56%

Table 3.3 Classification results on MUTAG, PTC and PROTEINS, in terms of F_1-$Score$, based on CapsuleGNN method with the combinations of SGNs of different orders, the bold values are the best results

Dataset	MUTAG	PTC	PROTEINS
Original	86.32 ± 7.52	62.06 ± 4.25	75.89 ± 3.51
SGN$^{(1)}$	83.68 ± 8.95	61.76 ± 5.00	74.64 ± 3.55
SGN$^{(2)}$	82.63 ± 7.08	58.82 ± 3.95	73.39 ± 6.03
SGN$^{(0,1)}$	87.37 ± 8.55	63.53 ± 4.40	76.25 ± 3.53
SGN$^{(0,2)}$	87.89 ± 5.29	62.20 ± 6.14	73.00 ± 3.17
SGN$^{(0,1,2)}$	$\mathbf{89.47 \pm 7.44}$	$\mathbf{64.12 \pm 3.67}$	$\mathbf{76.34 \pm 4.13}$
Gain	3.65%	2.19%	0.59%

a single network in most cases, which prove that SGNs can indeed provide the latent and significant structural information. Furthermore, when the three single networks are considered together, i.e., SGN$^{(0,1,2)}$, they can achieve the best performance in the classification.

3.6.5 *S^{2}GN for Graph Classification*

In this experiment, the network in the dataset is divided to ten equal parts, two of which are selected as the test set, and the remaining eight are used as the training set. The above algorithms are used to generate S^{2}GNs of different orders, and then Graph2Vec and CapsuleGNN methods are adopted to learn the feature representations of S^{2}GNs. Finally, the BLS method is applied for classification and to calculate the F1-Score. In order to avoid accidental sampling, each sampling strategy was carried out 10 sampling averaging. Based on Graph2Vec and CapsuleGNN methods, the obtained experimental results are shown in Tables 3.4 and 3.5, respectively.

The influence of each sampling method on the model is compared between the two feature extraction methods. It is found that, under the same feature extraction method, each sampling strategy has different advantages and disadvantages, which may be related to the specific network structure in the dataset. With the two feature extraction algorithms, the dataset MUTAG has the best classification effect under biased walk. Here, the results based on three sampling methods are compared with those based on the original method. It is found that the improved sampling subgraph network algorithm can maintain the original accuracy and even outperform

Table 3.4 Classification results of three sampling strategies in MUTAG, PTC and PROTEINS, in terms of F_1-$Score$, based on Graph2Vec method, the bold values are the best results

Dataset	MUTAG	PTC	PROTEINS
Original	83.15 ± 9.25	60.17 ± 6.86	73.30 ± 2.05
S^{2}GN-BW	$\mathbf{86.80 \pm 5.02}$	62.90 ± 2.19	75.44 ± 3.85
S^{2}GN-ST	82.03 ± 3.76	62.39 ± 6.36	74.33 ± 2.86
S^{2}GN-FF	83.33 ± 6.13	62.46 ± 5.17	73.77 ± 2.15
BLS-S^{2}GN	83.63 ± 6.84	$\mathbf{63.28 \pm 6.06}$	$\mathbf{74.92 \pm 2.58}$
Gain	0.58%	5.17%	2.21%

Table 3.5 Classification results of three sampling strategies in MUTAG, PTC and PROTEINS, in terms of F_1-$Score$, based on CapsuleGNN method, the bold values are the best results

Dataset	MUTAG	PTC	PROTEINS
Original	86.32 ± 7.52	62.06 ± 4.25	75.89 ± 3.51
S^2GN-BW	$\mathbf{91.84 \pm 5.00}$	66.06 ± 3.34	77.47 ± 2.35
S^2GN-ST	89.21 ± 5.05	63.50 ± 3.71	76.85 ± 2.54
S^2GN-FF	86.10 ± 5.32	65.77 ± 4.42	76.37 ± 1.90
BLS-S^2GN	91.63 ± 2.89	$\mathbf{66.10 \pm 6.37}$	$\mathbf{78.32 \pm 2.94}$
Gain	6.39%	6.50%	3.20%

the optimal result of the original algorithm. Experiments show that the variance of all accuracy on the dataset is reduced. For example, under the condition of CapsuleGNN feature extraction for the MUTAG dataset, the highest average accuracy is around 86.32%, and the variance is 0.0752. While using BLS-S^2GN, the highest average accuracy can reach 91.84%, and the variance is reduced to 0.0289. All cases show that the newly proposed classification model can be combined with different feature extraction methods, which can further improve the accuracy and stability of classification.

3.7 Computational Complexity

Now, it is to analyze the computational complexity in building SGNs. Denote by $|V|$ and $|E|$ the numbers of nodes and links, respectively, in the original network. The average degree of the network is calculated by

$$K = \frac{1}{|V|}\sum_{i=1}^{|V|} k_i = \frac{2|E|}{|V|}\,, \tag{3.8}$$

where k_i is the degree of node v_i. Based on Algorithm 1, the time complexity in transforming the original network to SGN$^{(1)}$ is

$$\mathscr{T}_1 = \mathscr{O}(K|V| + |E|^2) = \mathscr{O}(|E|^2 + |E|) = \mathscr{O}(|E|^2)\,. \tag{3.9}$$

Then, the number of nodes in SGN$^{(1)}$ is equal to $|E|$ and the number of links is $\sum_{i=1}^{|V|} k_i^2 - |E| \le |E|^2 - |E|$ [26]. Similarly, one can get the time complexity in transforming SGN$^{(1)}$ to SGN$^{(2)}$, as

$$\mathscr{T}_2 \le \mathscr{O}((|E|^2 - |E|)^2) = \mathscr{O}(|E|^4)\,. \tag{3.10}$$

Meanwhile, the computational complexity of these methods is evaluated in terms of the average computational time of SGN and S^2GN generated by the three sampling strategies on the three datasets. The results are presented in Table 3.6, where one can see that, overall, the computational time of S^2GN is much less than

Table 3.6 Average computational time to establish SGN and S^{2}GNs by the three sampling strategies on the three datasets

Time (Seconds)	SGN	S^{2}GN		
		BW	ST	FF
MUTAG	1.58×10^2	0.252	0.090	0.382
PTC	1.93×10^3	0.804	0.607	0.985
PROTEINS	3.20×10^3	1.161	1.625	3.697

that of SGN for each sampling strategy on each dataset, decreasing from hundreds of seconds to less than 4 s. These results suggest that, by comparing with SGN, the S^{2}GN model can indeed largely increase the efficiency of the network algorithms.

In fact, it is possible to estimate the time complexity of S^{2}GN model in theory. For biased walk, consider the 2nd random walk mechanism of Node2Vec, where each step of a random walk is based on the transition probability α, which can be precomputed, so the time consumption of each step using alias sampling is $\mathcal{O}(1)$. The Kruskal algorithm used to generate spanning trees is a greedy algorithm, which has $\mathcal{O}(|E|log(|E|))$ time complexity. Fire forest is an exploration-based method. The difference between this method and the random walk method is that, when a node is visited, it will no longer be visited again in the fire forest. It is known that the computational complexity of SGN$^{(1)}$ is $\mathcal{O}(|E|^2)$ and that of constructing SGN$^{(2)}$ is $\mathcal{O}(|E|^4)$. The S^{2}GN model constrains the expansion of the network scale and reduces the cost of constructing SGNs to the fixed $\mathcal{O}(|E|^2)$. Thus, the time computational complexity $\mathcal{T}$ of the S^{2}GN model can be calculated as

$$\mathcal{T} \leq \mathcal{O}(|E|log|E| + |E|^2|) \tag{3.11}$$

Combining with the different sampling strategies, one can see that the time complexity of S^{2}GN is much lower than that of SGN.

3.8 Conclusion

In this chapter, after reviewing the notions of SGN and S^{2}GN, the BLS is introduced to graph data mining. Moreover, a classification framework is introduced that combines S^{2}GN, feature representation, and BLS to enhance the performance of graph classification task. SGN and S^{2}GN can generate different higher-order graphs to capture latent structural information of the original network from various aspects and expand the feature space. Experiments on three datasets demonstrate that BLS can fully utilize these latent features to achieve significant improvement in graph classification. In addition, it is found that, compared with SGN, S^{2}GN has much lower time complexity, which was reduced by almost two orders of magnitude. More significantly, combined with BLS, it has competitive performances on graph classification.

References

1. Ugander, J., Backstrom, L., Kleinberg, J.: Subgraph frequencies: mapping the empirical and extremal geography of large graph collections. In: Proceedings of the 22nd International Conference on World Wide Web, pp. 1307–1318. ACM, New York (2013)
2. Benson, A.R., Gleich, D.F., Leskovec, J.: Higher-order organization of complex networks. Science **353**(6295), 163–166 (2016)
3. Wang, H., Zhang, P., Zhu, X., Tsang, I.W.-H., Chen, L., Zhang, C., Wu, X.: Incremental subgraph feature selection for graph classification. IEEE Trans. Know. Data Eng. **29**(1), 128–142 (2017)
4. Yang, C., Liu, M., Zheng, V.W., Han, J.: Node, motif and subgraph: leveraging network functional blocks through structural convolution. In: 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), pp. 47–52. IEEE, Piscataway (2018)
5. Alsentzer, E., Finlayson, S.G., Li, M.M., Zitnik, M.: Subgraph neural networks. Preprint. arXiv:2006.10538 (2020)
6. Xuan, Q., Wang, J., Zhao, M., Yuan, J., Fu, C., Ruan, Z., Chen, G.: Subgraph networks with application to structural feature space expansion. IEEE Trans. Knowl. Data Eng. **33**(6), 2776–2789 (2021)
7. Philip Chen, C.L., Liu, Z.: Broad learning system: an effective and efficient incremental learning system without the need for deep architecture. IEEE Trans. Neural Netw. Learn. Syst. 29(1):10–24 (2017)
8. Kurant, M., Markopoulou, A., Thiran, P.: On the bias of BFS (breadth first search). In: 2010 22nd International Teletraffic Congress (lTC 22), pp. 1–8. IEEE, Piscataway (2010)
9. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: online learning of social representations. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 701–710 (2014)
10. Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: Line: large-scale information network embedding. In: Proceedings of the 24th International Conference on World Wide Web, pp. 1067–1077 (2015)
11. Leskovec, J., Faloutsos, C.: Sampling from large graphs. In: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 631–636 (2006)
12. Satuluri, V., Parthasarathy, S., Ruan, Y.: Local graph sparsification for scalable clustering. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, SIGMOD '11, New York, pp. 721732 (2011)
13. Li, G., Semerci, M., Yener, B., Zaki, M.J.: Graph classification via topological and label attributes. In: Proceedings of the 9th International Workshop on Mining and Learning with Graphs (MLG), San Diego, USA, vol. 2 (2011)
14. Narayanan, A., Chandramohan, M., Chen, L., Liu, Y., Saminathan, S.: subgraph2vec: learning distributed representations of rooted sub-graphs from large graphs. In: International Workshop on Mining and Learning with Graphs (2016)
15. Narayanan, A., Chandramohan, M., Venkatesan, R., Chen, L., Liu, Y., Jaiswal, S.: graph2vec: learning distributed representations of graphs. In: International Workshop on Mining and Learning with Graphs (2017)
16. Shervashidze, N., Schweitzer, P., van Leeuwen, E.J., Mehlhorn, K., Borgwardt, K.M.: Weisfeiler-Lehman graph kernels. J. Mach. Learn. Res. **12**, 25392561 (2011)
17. Yanardag, P., Vishwanathan, S.V.N.: Deep graph kernels. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1365–1374. ACM, New York (2015)
18. Velikovi, P., Cucurull, G., Casanova, A., Romero, A., Li, P., Bengio, Y.: Graph attention networks. In: International Conference on Learning Representations (2018)

19. Zhang, X., Chen, L.: Capsule graph neural network. In: International Conference on Learning Representations (2019)
20. Philip Chen, C.L., Liu, Z.: Broad learning system: a new learning paradigm and system without going deep. In: 2017 32nd Youth Academic Annual Conference of Chinese Association of Automation (YAC), pp. 1271–1276. IEEE, Piscataway (2017)
21. Jin, J.-W., Philip Chen, C.L.: Regularized robust broad learning system for uncertain data modeling. Neurocomputing **322**, 58–69 (2018)
22. Gao, S., Guo, G., Philip Chen, C.L.: Event-based incremental broad learning system for object classification. In: Proceedings of the IEEE International Conference on Computer Vision Workshops, pp. 2989–2998 (2019)
23. Chen, M., Wei, X., Gao, Y., Huang, L., Chen, M., Kang, B.: Deep-broad learning system for traffic flow prediction toward 5g cellular wireless network. In: 2020 International Wireless Communications and Mobile Computing (IWCMC), pp. 940–945. IEEE, Piscataway (2020)
24. Agarwal, S., Branson, K., Belongie, S.: Higher order learning with graphs. In: Proceedings of the 23rd International Conference on Machine Learning, pp. 17–24. ACM, New York (2006)
25. Wernicke, S.: Efficient detection of network motifs. IEEE/ACM Trans. Comput. Biol. Bioinform. (TCBB) **3**(4), 347–359 (2006)
26. Harary, F., Norman, R.Z.: Some properties of line digraphs. Rendiconti del Circolo Matematico di Palermo **9**(2), 161–168 (1960)
27. Schiöberg, D., Schneider, F., Schmid, S., Uhlig, S., Feldmann, A.: Evolution of directed triangle motifs in the google+ OSN. Preprint. arXiv:1502.04321 (2015)
28. Fu, C., Zhao, M., Fan, L., Chen, X., Chen, J., Wu, Z., Xia, Y., Xuan, Q.: Link weight prediction using supervised learning methods and its application to Yelp layered network. IEEE Trans. Knowl. Data Eng. **30**(8), 1507–1518 (2018)
29. Grover, A., Leskovec, J.: node2vec: scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 855–864. ACM, New York (2016)
30. Dey, A., Broumi, S., Bakali, A., Talea, M., Smarandache, F., et al.: A new algorithm for finding minimum spanning trees with undirected neutrosophic graphs. Granular Comput. **4**(1), 63–69 (2019)
31. Najman, L., Cousty, J., Perret, B.: Playing with kruskal: algorithms for morphological trees in edge-weighted graphs. In: International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing, pp. 135–146. Springer, Berlin (2013)
32. Debnath, A.K.., de Compadre, R.L.L., Debnath, R.L.L., Shusterman, A.J., Hansch, C.: Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. J. Med. Chem. **34**(2), 786–797 (1991)
33. Toivonen, H., Srinivasan, A., King, R.D., Kramer, S., Helma, C.: Statistical evaluation of the predictive toxicology challenge 2000–2001. Bioinformatics **19**(10), 1183–1193 (2003)
34. Nguyen, D., Luo, W., Nguyen, T.D., Venkatesh, S., Phung, D.: Learning graph representation via frequent subgraphs. In: Proceedings of the 2018 SIAM International Conference on Data Mining, pp. 306–314. SIAM, Philadelphia (2018)
35. Sabour, S., Frosst, N., Hinton, G.: Matrix capsules with EM routing. In: 6th International Conference on Learning Representations, ICLR, pp. 1–15 (2018)