EnD: Enhanced Dedensification for Graph Compressing and Embedding

Tanvir Hossain
Department of Computer Science
Georgia State University
Atalanta, GA 30002
thossain5@student.gsu.edu

Esra Akbas
Department of Computer Science
Georgia State University
Atalanta, GA 30302
eakbas1@gsu.edu

Muhammad Ifte Khairul Islam
Department of Computer Science
Oklahoma State University
Stillwater, OK 74075
ifte.islam@okstate.edu

2022 IEEE International Conference on Data Mining Workshops (ICDMW) | 979-8-3503-4609-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICDMW58026.2022.00092

***Abstract*—Graph representation learning is essential in applying machine learning methods on large-scale networks. Several embedding approaches have shown promising outcomes in recent years. Nonetheless, on massive graphs, it may be time-consuming and space inefficient for direct applications of existing embedding methods. This paper presents a novel graph compression approach based on dedensification called Enhanced Dedensification with degree-based compression (EnD). The principal goal of our system is to assure decent compression of large graphs that eloquently favor their representation learning. For this purpose, we first compress the low-degree nodes and dedensify them to reduce the high-degree nodes' loads. Then, we embed the compressed graph instead of the original graph to decrease the representation learning cost. Our approach is a general meta-strategy that attains time and space efficiency over the original graph by applying the state-of-the-art graph embedding methods: Node2vec, DeepWalk, RiWalk, and xNetMf. Comprehensive experiments on large-scale real-world graphs validate the viability of our method, which shows sound performance on single and multi-label node classification tasks without losing accuracy.**

***Index Terms*—Graph Compression, Graph Mining, Network Embedding, Node Classification, GraphML**

I. Introduction

In recent times graph mining has received special attention and achieved a great performance for different real-world problems by capturing complex information within graph data. Specifically, network embedding has become a comprehensive technique in network analysis to cope with the complexities. It encodes the structural information of graphs into low-dimensional vector spaces that preserve the local proximity of vertices, i.e., using short random walks [12], [23], tracking roles [13], [21], and decomposing matrices [24]. Through this, it plays an essential role in extracting strong and informative characteristics over the nodes of a graph. Recently, several network embedding methods are proposed for different graph mining problems: node classification [1], link prediction [27], graph clustering [2], [28] and community detection [29].

Although these methods deliver a satisfactory performance, they have some common challenges. One is that while these methods operate well on sparse networks, quadratic time complexity degrades performance on large-scale networks. Also, with a short random walk or a small order, the approaches focus on the local structure of the networks. Therefore, they cannot preserve the global structural patterns on large-scale networks. In some methods, layer-based structure capturing [25] and matrix eigendecomposition [8] require high computational space.

Furthermore, While they learn embeddings for high-degree nodes, the quality of embedding for low-degree nodes is suboptimal due to the lack of structural information and over-focus on high-degree nodes. Besides, they use stochastic gradient descent to attain a non-convex optimization goal. Hence, that can be easily bound to unsound local minima for poor hyper-parameter tuning. Since those initialize the embeddings randomly, similar nodes may get different representation with local minima. One of the solutions to overcome these problems is to reduce the graph's size through compression, eliminating noise and removing redundant information. Graph compression converts an extensive network to a smaller one without losing much information about the global structure of the graph and the local relationship between the vertices.

This paper proposes a novel graph compression technique based on 'dedensification.' It keeps high-degree nodes' connection information by installing compressor nodes that remove multiple edges without losing the graph information. We enhance it with iterative dedensification that selects high-degree nodes within a range and remove their edge loads by installing a compressor node. However, as many real-world networks follow power law distribution [16], [18], which have many low-degree and few high-degree nodes, with only dedensification, we may not be able to make the comprehensive network much smaller. We present a degree-based compression technique that creates high-degree nodes by compressing low degrees to solve this problem. We consider that high-degree nodes carry more crucial information in graphs, and low-degree nodes may contain noisy information. Hence, we compress the low-degree nodes into their high-degree neighbors. This way, we get better information by eliminating impact of high-degree nodes by eliminating noises.

We apply our compressing model for network embedding problem. We choose four embedding methods as the baseline. Two of them are community-based methods, Node2vec [12] and DeepWalk [23], that consider nodes in the same community to get similar embeddings. The other two are structural embedding methods, RiWalk [21], and xNetMf [13], that consider the role of nodes in the network.

2375-9259/22/$31.00 ©2022 IEEE
DOI 10.1109/ICDMW58026.2022.00092

Our contributions to this research are as follows -

- **New graph compression method**. We represent a new graph compression method through iterative dedensification with enhancement via a degree-based compression.
- **Efficient graph representation learning on the compressed graph**. We apply random walk-based and role-based embedding methods to our compressed graph. We achieve substantial gains in embedding time over the original graphs.
- **Better classification performance**. In the experiment, we demonstrate that embedding with a compressed graph improves efficiency and achieves similar or more satisfactory accuracy in single and multi-label classification tasks. While compressor nodes keep the high-degree nodes' information, super-nodes conserve the vertices' local neighborhood. Therefore, by preserving the local and global structures, the compressed graph does not lose its effectiveness in embedding.

The rest of this research is organized as follows: In section II, we represent related works for embedding and compression. The results and experimental design are discussed in section IV. Then, we discuss our proposed framework, including compressing and embedding methodology in section III. Finally, we report our conclusion in section V.

II. Related Work

This section briefly discusses the related work in network embedding and graph compression.

Network embedding. Since the initial stage many methods have been proposed for network embedding. Earlier works like PCA [31] and LE [26] (locally linear embedding) consider graph embedding a dimension reduction problem. Although these methods give satisfactory results on small graphs, they are not scalable for massive networks as their time complexity is the least quadratic. Later approaches focus on scalability by applying matrix factorization or neural networks. DeepWalk [23] and Node2vec [12] adopt random walks with fixed lengths to sample the local neighbors. They apply skip-gram language model on these random walks by treating them as sentences while Line [30] captures global and local structures by using an objective function based on the order-based proximity of nodes.

On the other hand, some structure-based graph embedding methods use dynamic time wrapping [25] and diffusion kernel [8] to learn nodes' representation. One of the current methods, RiWalk [21], establishes a role-driven function by using Shortest-Path and Weisfiler-Lehman kernels. Another approach, REGAL, applies cross-network factorization (xNetMf [13]) over the networks to achieve graphs' structural information. Other matrix factorization based techniques, e.g., GraRep [4] and NetMF [24], create an objective matrix for embedding graphs.

Despite their considerable performance, some challenges remain to handle. Some methods only focus on the node's local neighborhood instead of long-distance neighbors. Therefore, they fail to capture the global structure of the graph. Besides, massive graphs require enormous embedding space. Our graph compressing method captures the graph's local and global information in latent embedding space with degree-based compression and iterative dedensification. It also improves the baselines' efficiency.

Graph compressing. Recently a notable quantity of research has been done on graph compression. Compressed graphs are being applied for anomaly detection [3], efficient graph reordering: on social [7] and biological networks [14], graph pattern matching [10], query processing [11], shortest path finding [19], community searching [3], [17], and numerous scientific tasks. Several methods apply compression as a preprocessing step to learn graph embedding. HARP [5], MILE [20], and GraphZoom [6] first use graph coarsening techniques, then compressed graphs learn the representation and refine in the respective multi-level framework. NECL [15] is another meta-strategy that compresses the nodes into super nodes based on common neighbors' similarity. However, our approach differs from these approaches. It compresses the networks based on a small degree value following iterative dedensification and conserves the meaningful local and global information between nodes. Besides, it decently deals with time and outperforms the baselines.

III. Methodology

In this section, first, we explain a degree-based densifying method that generates high-degree nodes. Next, we represent our iterative dedensification method that summarizes the graph by reducing the edges of the high-degree nodes. Finally, we describe how to get the representation of the compressed graph and its distribution to the original graph. Before explaining in detail, we briefly introduce the necessary preliminaries.

*Definition 1: A **graph** $G = (V_G, E_G)$ where V_G denotes the set of vertices and E_G represents the set of edges. For a vertex $v \in V_G$, its neighbor set is denoted by $N(v)$, where $N(v) = \{u | u \in V_G : (u, v) \in E_G\}$.*

*Definition 2: A **compressed graph** of an original graph $G = (V_G, E_G)$ is represented as $G_C = (V_C, E_C)$ where some super-nodes $V_{sup} \in V_C$ hold some sub-nodes $V_{sub} \notin V_C$ and some super-edges $E_{sup} \in E_C$ hold some sub-edges $E_{sub} \notin E_C$.*

*Definition 3: **Network embedding** is a mapping $\phi : V \rightarrow \mathbb{R}^d, d << |V|$ which represents each vertex $v \in V$ as a point in a low dimensional space $\mathbb{R}^d$. Here d is a parameter specifying the number of dimensions of our feature representation. For every source node $u \in V$, we define $N_S(u)$ through a sampling strategy S. An embedding method seeks to maximize the probability of observing a network neighborhood $N_S(u)$ for a node u conditioned on its feature representation, given by ϕ.*

A. Degree-Based Compression

In degree-based compression we have two motivations. Firstly, low-degree nodes could have noisy information, also, they are primarily dependent on a few neighbors, so their representation will be similar. Therefore, we may skip their

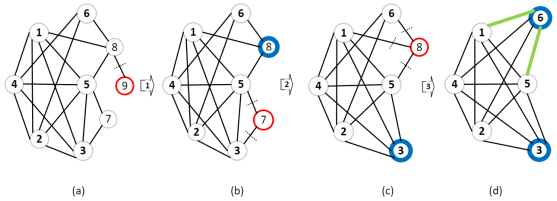


Fig. 1: Degree Based Compression

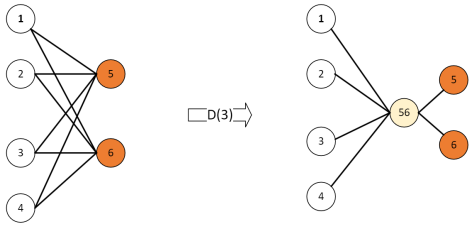


Fig. 2: Dedensification

learning but use their neighbors to assign embedding. Secondly, a densifying graph that merges low-degree nodes with a neighbor may generate more high-degree nodes to use in the dedensification part as the second step of our model.

For a specific degree value n, we merge the nodes with degrees 1 to n $(V^1, V^2, ...V^n)$ with their least-connected neighbors. In the first step, we remove $1-degree$ nodes (V^1) from the graph by accumulating them in the neighbor's node. We repeat this until there is no V^1 node in the graph. Then, we deduct V^2 to V^n nodes into their min-degree neighbor to create a super-node V_{sup}. The idea is that a sample node merges with its lowest-degree neighbor for a specific degree value. It produces a compressed graph G_{C_n} with high-degree nodes. In a particular stage, when a n-degree node merges with the super-node, the super-node creates connections with neighbors of the V^n node.

In Fig. 1, we demonstrate an example of degree-based compression on a toy graph. It shows the degree-based compression up to $n = 3$. At first, the V^1 node V_9 coarsens into V_8, (Fig. 1(b)). At this point, a V^2 node V_7 exists in the graph. It accumulates into V_3 because $Degree(V_3) < Degree(V_5)$. In Fig. 1(b) V_7 coarsens into V_3. Next, between two $3-degree$ nodes V_8 and V_6, in this example, V_8 compresses first. Fig. 1(b) depicts as V_6 is the lowest-degree neighbor of V_8 with degree 3, V_8 merges to V_6. The green edges denote V_8's other two neighbors, V_5 and V_1 connect to the new super-node V_6. Hence, the degree of V_6 increases to 4.

B. Iterative Dedensification

Iterative dedensification is inspired by a novel lossless compressing method, 'dedensification' [22], that reduces the neighborhood around the high-degree nodes for pattern matching problems. In many real-world graphs, low-degree nodes are connected to numerous high-degree nodes. Even a group of related low-degree nodes are connected to the same group of high-degree nodes. As an example, in the toy graph on Fig 2, low degree nodes, V_1, V_2, V_3, V_4 are all connected to the high degree nodes V_5 and V_6. Dedensification's primary motivation is to reduce high-degree nodes' loads without losing connection. Learning to embed these high-degree nodes takes more time, and also they include many random walks that cause redundant information for training. Reducing their degree without losing information helps to solve these problems.

For dedensification, the nodes in a graph are partitioned into high-degree (V_H) and low-degree nodes' (V_L) set. A threshold τ differentiates high-degree from low-degree nodes. If the degree of a node $v \in V_G$ is higher than given threshold, $Degree(v) > \tau$ and then we call the node v as high-degree node.

To reduce the load around V_H, we can remove the edges of these nodes, but we do not want to lose their information. Therefore, a 'compressor node' is installed between these high-degree nodes and their neighbors to keep the connections.

*Definition 4: A **compressor node** N_C (where $N_C \notin V_G$) is an intermediate node that decouples high-degree nodes from their incoming connections and summarizes the graph by keeping the connections' references within it.*

For example, according to Fig. 2, in the first graph, for $\tau = 3$,it has four low-degree nodes (white), and two high-degree nodes (orange). In the second graph, a compressor node N_{56} is installed that preserves the connections' information between V_H and V_L. It reduces the number of edges in the graph from eight to six, where the number of high-degree nodes' edges is one instead of four.

The high-degree nodes are decomposed into different sets to install the compressor nodes. In general, for $|V_H| = p$ number of high-degree nodes, the number of sets is supposed to be $2^p - (p + 1)$, which is exponential. An auxiliary-mapping [1] technique maps the sets with their common neighbors and executes only for p times. In each combination of high-degree nodes, for the number of common neighbors, the edges can be compressed into the 'compressor node.' For effective dedensification, a particular condition is defined.

Condition 1: In any undirected graph for a specific τ value, a compressor node can be installed for n $(n >= 2)$ numbers of high-degree nodes if they have at least $n+1$ common neighbors or vice-versa. If the condition satisfies, the number of edges must reduce from the graph.

For example in Fig. 3(a), in accordance with Condition 1, for $\tau = 3$ node V_7 and V_8, so, $(n = 2)$ are high-degree nodes (orange colored nodes) having 4 common neighbors $\{V_2, V_3, V_4, V_5\}$ that is greater than $(n + 1 = 3)$. So, the number of incoming edges to V_7 and V_8 is 8 (bold black edges). In Fig. 3(b), after dedensification, a compressor node N_{87} is installed. For those four common neighbors and two high-degree nodes, the number of edges is 6 (bold orange edges).

[1] $auxiliary - mapping$: An efficient hashing technique that maps the high-degree nodes' with their common neighbors. The details in [22].

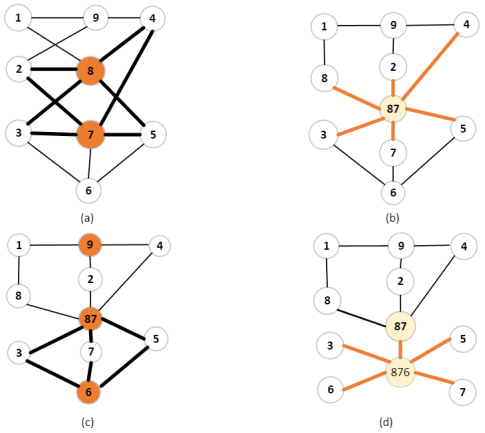


Fig. 3: Iterative Dedensification

Fig. 3(c) shows that for $\tau = 2$, node V_9 is a high-degree node, along with node V_{87} (N_{87} for Fig. 3(b)) and node V_6. In this graph, high-degree nodes of G (V_7 and V_8) are low-degree nodes for G_3's high-degree nodes. Node V_9 and V_{87} , so, $(n = 2)$ have two common neighbors $\{V_2, V_4\}$ that is less than $(n + 1)$. Therefore, dedensification is unattainable for graph size augmentation. However, nodes V_{87} and V_6 have three common neighbors $\{V_3, V_7, V_5\}$ that are equal to $n + 1$. So the condition is satisfied. Fig. 3(d) illustrates a new compressor node N_{876} where the two high-degree nodes' number of connections reduces from 6 to 5.

When the conditions of dedensification satisfy in a graph, an iterative dedensification is also applicable to it. In the experiment, we apply iterative dedensification by changing the threshold τ from max to min. For the iteration range from $high$ to low the original graph is continuously dedensified from $G_{high}, G_{high-1}, ...$ to G_{low}. In Fig. 3, for a range (3 to 2), G is first dedensified to G_3, and then again, G_3 is dedensified to G_2. The number of edges reduces from 14 to 12.

The benefit of dedensification is that the compressed graph's reconstruction to the original graph is easy because the compressor node holds the edge information. Hence, it is lossless. While it increases the number of nodes by establishing the compressor nodes, it decreases the number of edges a lot; Besides, we alleviate the nodes' increment problem with degree-based compression that reduces the number of nodes. So, our final compressed graph has less number of nodes and edges.

In Algorithm 1, we briefly describe the overall compression process. After degree-based compression in line $1 - 5$, the graph is dedensified for a specific range $high$ to low. We get the high-degree nodes to dedensify for specific τ value. The high-degree nodes are decomposed into different sets (line 15). Next, for each set V_{HS}, if every node has a more common neighbor than its size $|V_{HS}|$ ($cond_1$) or vice-versa ($cond_2$), all of its nodes are compressed to a compressor node (line 22). Lines $23 - 25$ show the rest of the connecting process of the compressor node.

Algorithm 1: Overall Graph Compression Algorithm

Input: A Graph G, Degree n
Output: A Compressed Graph G_C

1 **for** $V_i \ \epsilon V_G$ **do**
2 **for** $d \leftarrow 1$ **to** n **do**
3 Compress Recursively for $Degree(V_i) = d$
4 **end**
5 **end**
6 *Next, n − degree compressed graph will be dedensified*
7 $V_H \leftarrow \emptyset$; $V_L \leftarrow \emptyset$; $S_C \leftarrow \emptyset$
8 **for** $\tau \leftarrow high$ **to** low **do**
9 **for** $(Vi \in V_{Gc})$ **do**
10 **if** $Degree(V_i) > \tau$ **then**
11 $V_H \leftarrow V_H \cup V_i$
12 $V_L \leftarrow V_{Gc} - V_H$
13 **end**
14 **end**
15 *V_H's nodes are decomposed by auxiliary mapping*
16 $S_{V_H} \leftarrow V_{HS_1}, V_{HS_2}....., V_{HS_k}$
17 **for** $V_{HS} \in S_{V_H}$ **do**
18 $C_N \leftarrow CommonNeighbors(\forall(V \in V_{HS}))$
19 $cond_1 = (|V_{HS}| \geq 2)$ and $|C_N| \geq (|V_{HS}| + 1)$
20 $cond_2 = (|C_N| \geq 2)$ and $|V_{HS}| \geq (|C_N| + 1)$
21 **if** $cond_1$ *or* $cond_2$ **then**
22 $N_C \leftarrow Compress(\forall(V \in V_{HS}))$
23 *Cut(V_{HS}, $N(V_{HS})$)*
24 *Connect (N_C, V_{HS})*
25 *Connect (N_C, $N(V_{HS})$)*
26 $S_C \leftarrow S_C \cup N_C$
27 S_C is the compressor nodes' set.
28 **end**
29 **end**
30 **end**

C. Network Embedding and Distribution

The goal of network embedding is to extract nodes' essential feature information. However, it must deal with the maintenance of space and time while preserving similar effectiveness as the baselines. Hence, we apply embedding methods on the compressed graph instead of the original graph and get the representations for compressed nodes. Then, we distribute the compressed nodes' embedding to the original graph's nodes.

We need to have a connected graph as the input for the embedding methods to distribute information across the network. This step is done before compression because our compressing strategy must generate a connected compressed graph. If a graph is not connected, we track the following steps to make it connected. We separate the largest connected component from the compressed graph and keep it in a list set. Next, For the rest of the graph, we repeatedly follow the same step and add every most significant connected component to the list set in descending order (based on node size, $|V|$). We select the node with the minimum degree (min-degree nodes) from each connected component. Finally, we connect every two components' min-degree nodes in the order within the list set. Hence, this process makes a graph connected.

For example, in Fig. 4, the connected components $C(1)$, $C(2)$, $C(3)$ and $C(4)$ are stored in the list-set where, $C(1) > C(2) > C(3) > C(4)$ in terms graph size. First, we extract

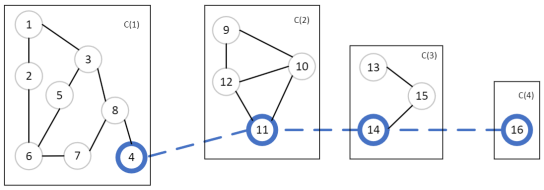


Fig. 4: Converting a disconnected graph to a connected graph

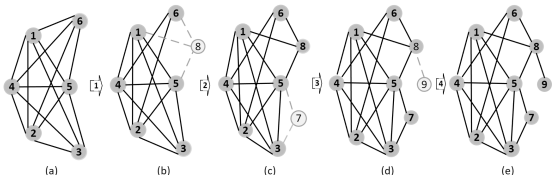


Fig. 5: Embedding Distribution

a min-degree node for each component: V_4, V_{11}, V_{14} and V_{16}. Finally, we create connections (bold blue dashed edges) through the min-degree nodes from the largest connected component to the smallest.

Next, we embed the compressed graph and distribute super-nodes' representation to their connected vertices in the original graph. The distribution process ends when all nodes get the representation. Fig. 5 depicts the process, a reversal demonstration of Fig. 1 in degree-based compression. The number of solid gray nodes is the same as the number of embedded nodes. In the beginning, Fig. 5(a) represents a graph with six vertices coarsened from a nine nodes graph. The embedded nodes (V_ϕ) are $V_1 ... V_6$ and non-embedded nodes ($V_{\neg\phi}$) are V_7, V_8 and V_9. At first, non-embedded vertex V_8 gets new representations from the average of its three embedded neighbors V_6, V_1, and V_5. Likewise, vertices $V_7, V_9 \in V_{\neg\phi}$ get their representation (Fig. 5(c) and Fig. 5(d)) and the number of embedded node is updated to 7 and 8 respectively. Finally, the original graph is fully embedded where $|V_\phi| = |V_G| = 9$ (Fig. 5(e)). After representation learning, a machine learning model is applied to classify the nodes.

IV. Experiment Design and Analysis

This section first provides an overview of the datasets and embedding methods for experiments. Then, we present the performance of the proposed method in single-label and multi-label node classification tasks. We further measure the impact of the crucial parameter $n - degree$ in these tasks.

A. Datasets

In our experiments, we use five real-world graphs[2] (Table I: graphs' statistics). Cora is a citation network of machine learning-related papers. The network consists of scientific publications classified into seven classes, and each paper has a single topic. CiteSeer is another citation network of computer science-related publications. The labels represent the papers' research topics, where papers are classified into six distinct types. Wiki treats web pages as the vertices and hyperlinks as the connections. The web pages are of 17 different types. DBLP is another co-authorship network in computer science. The labels represent the author's research areas where an author may have different research alignments. BlogCatalog is a bloggers' social network where bloggers are the vertices and relationships are the edges. It is a multi-labeled dataset where labels represent bloggers' publication categories.

[2]https://linqs.org/datasets/

B. Baseline Methods

To evaluate the performance of our compressing method, we apply DeepWalk (DW), Node2vec (N2V), RiWalk, and xNetMf as the baseline embedding methods on the compressed networks. While DeepWalk and Node2vec are proximity-based models that preserve the network's contextual information, RiWalk and xNetMf are structure-based models that maintain the network's role-based information.

Parameter Settings: For evaluation, we first obtain vertices' embedding vectors using the methods mentioned in the previous section. Next, we use these vectors as the node features to train the classifier. Then, we sample one portion of the labeled vertices for training and another for testing. For a detailed comparison, we change the percentage of labeled training vertices and the degree value n. We later calculate the Macro-F1 and Micro-F1 scores along with the embedding times to check the effectiveness and efficiency of the proposed method.

Furthermore, to have a detailed comparison between our method and the baseline methods, we vary the percentage of the training data from 10% to 80% for the BlogCatalog network and 5% to 80% for the other four networks. For DW and N2V baselines, we take 5% of labeled data for training, while for RiWalk and xNetMf, it is 80% as their original research. To ensure our experiments' reliability, we ran the classification process for ten different seeds and recorded the average Micro-F1 and Macro-F1 scores. The machine configuration for our experiment is a server running Ubuntu 14:04 with 4 intel 2.6Hz 10-core CPUs and 48 GB of Memory.

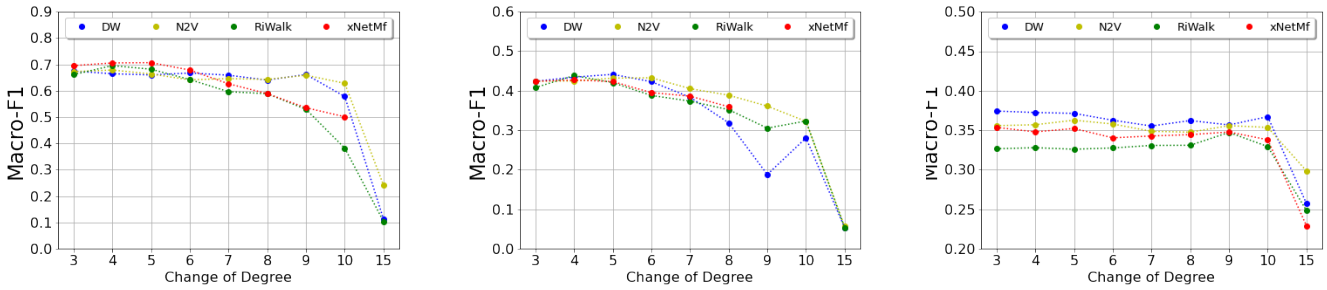
The most important parameter for degree-based compression is the n-degree. We set $n = 5$ as the cutting point for compression. The iterative dedensification range on BlogCatalog is set from 4000 to 399. The range for the other four networks is from 150 to 29. In the case of embedding, we set the representation size to 128. For the random walk-based methods DeepWalk and Node2vec, we set the following

TABLE I: Graphs Statistics.

Dataset	$\|V\|$	$\|E\|$	# of class	Label
Cora	2,708	5,278	7	Single
CiteSeer	3,264	4,536	6	Single
Wiki	2405	12,761	17	Single
DBLP	27,199	66,832	4	Multi
BlogCatolog	10,312	333,983	39	Multi

TABLE II: Performance comparison for the single-label classification task.

Graphs		Macro-F1			Micro-F1			Time		
		BL	EnD	Gain %	BL	EnD	Gain %	BL	EnD	Gain %
Cora	N2V	**66.63**	66.34	-0.43	**68.53**	67.76	-1.18	196.51	**31.31**	84.06
	DW	64.62	**65.98**	2.10	66.60	**68.36**	2.65	10.19	**1.938**	80.99
	RiWalk	64.83	**76.56**	18.08	66.86	**77.84**	16.49	9.60	**2.001**	79.15
	xNetMf	70.34	**73.95**	5.14	71.85	**75.90**	5.65	12.51	**1.28**	89.72
CiteSeer	N2V	43.01	**43.19**	0.41	46.38	**47.51**	2.45	229.48	**20.49**	91.07
	DW	39.49	**44.14**	11.770	42.43	**49.35**	16.30	11.93	**1.90**	84.08
	RiWalk	33.68	**46.43**	37.87	39.14	**52.74**	34.74	10.91	**1.84**	83.07
	xNetMf	38.94	**47.31**	21.51	45.48	**55.18**	21.31	12.13	**1.264**	89.59
Wiki	N2V	35.89	**36.27**	1.08	48.23	**48.71**	0.99	198.96	**127.32**	36.00
	DW	36.91	**37.13**	0.61	49.21	**51.11**	3.86	9.67	**5.302**	45.17
	RiWalk	46.44	**49.82**	7.28	61.54	**63.60**	3.34	17.21	**9.939**	42.24
	xNetMf	47.37	**49.60**	4.70	63.47	**63.95**	0.75	12.37	**6.68**	45.99



(a) *Cora* (b) *CiteSeer* (c) *Wiki*

Fig. 6: Macro-F1 scores for single-label classification with n-degree variation

parameters: the number of random walks = 40, walk-length = 10, and window size = 10. In the case of Node2vec, we set the return parameter $p = 1$ and the in-out parameter $p = 0.25$. In the case of RiWalk, the number of random walks, window size, and walk length are the same as DeepWalk. We set the $Weisfiler - Lehman$ kernel for getting the networks' structural information and set the neighborhood distance $(k) = 4$. In xNetMf, The maximum hop distance has been set to 30 for the BlogCatalog. For other networks, the distance is set to 5. We also set the discount factor as 0.1 and the learning rate as 1.

C. Classification

This section compares our method with the baselines in single-label and multi-label classification tasks. In the first task, we study the graphs whose nodes have only one label, whereas, in the second task, graphs' nodes have multiple labels.

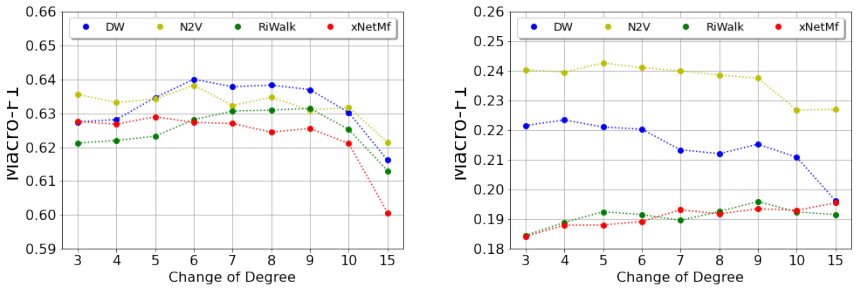
1) Single-label Classification: In single-labeled datasets, each node has a single label from multi-class values. We employ the multi-class Support Vector Machine (SVM) as the classifier for the classification task, which uses the one-vs-rest scheme. Table II shows the performance comparisons in single-label classification task. EnD either achieves almost similar scores or outperforms the baselines. Among the notable outcomes on the CiteSeer network, EnD achieves 37.87% and 21.51% in the Macro-F1 score for RiWalk and xNetMf. Besides, we obtain 18.08% and 5.14% gain for these methods on the Cora network. In Micro-F1 performance, on CiteSeer, the gains are 16.30%, 34.74%, and 21.31% gain for DW, RiWalk, and xNetMf, respectively. For the last two baselines on the Cora network, the gains are 16.49% and 5.65%. Furthermore, our method obtains significant gain on embedding time over the baselines for all single labeled graphs. For Cora and CiteSeer, the gains are about or over 80% for all the baselines.

In addition, Fig. 6a, 6b and 6c demonstrate the variation of Macro-F1 scores for varying degrees from 3 to 15 in single-label classification tasks. Generally, the scores gradually degrade for the increment of the degree values from 5. Note that xNetMf embeds a graph if its number of vertices is greater than or equal to the representation size. In Fig. 6a for $n > 10$ and in Fig. 6b for $n > 8$ the compressed networks' number of nodes drop below the representation size (128) and those become unable for embedding by xNetMf.

2) Multi-label Classification: In multi-labeled networks, a node can belong to more than one class. We employ a one-vs-rest logistic regression model as the classifier for the classification task. The model fits with L2 regularization and

TABLE III: Performance comparison for the multi-label classification task.

Graphs		Macro-F1			Micro-F1			Time		
		BL	EnD	Gain %	BL	EnD	Gain %	BL	EnD	Gain %
DBLP	N2V	63.01	**63.42**	0.65	66.03	**66.96**	1.41	2048.24	**474.62**	76.89
	DW	60.99	**63.46**	4.05	64.26	**66.69**	3.78	113.06	**26.57**	76.49
	RiWalk	63.39	**65.15**	2.77	67.60	**68.91**	1.934	353.67	**38.28**	89.18
	xNetMf	64.81	**65.76**	1.47	68.77	**69.35**	0.85	998.56	**85.411**	91.45
Blog-Catalog	N2V	**24.29**	24.28	-0.072	**38.33**	37.91	-1.116	3058.79	**1414.04**	53.77
	DW	21.05	**22.04**	4.68	36.53	**37.04**	1.39	49.33	**41.017**	16.86
	RiWalk	15.09	**19.18**	27.16	29.79	**33.17**	11.37	484.34	**456.10**	5.83
	xNetMf	14.79	**18.80**	27.11	29.84	**32.89**	10.24	719.35	**685.346**	4.73



(a) $DBLP$ (b) $BlogCatalog$

Fig. 7: Macro-F1 scores for nulti-label classification with n-degree variation

is implemented by LibLinear [9].

Table III illustrates that our method performs near or better than most of the baselines on all networks. The Marco-F1 and Micro-F1 scores on the DBLP dataset are almost close to the baselines. On the BlogCatalog network for RiWalk and xNetMf, the Macro-F1 scores' gains are near 27%, and the Micro-F1 scores' gains are about 11%. However, the improvement in embedding time is close to or more than 75% for each baseline. Our method archives the highest gain for xNetMf, which is 91.447%. Although in the BlogCatalog network, the gain in embedding time is little for DW, RiWalk, and xNetMf, it is significant for N2V, which is 53.771%. Furthermore, for the increase of degree values from 3 to 15, Figure 7a and 7b depict Macro-F1 scores for multi-label classification. On the DBLP dataset, score curves raise slightly and decrease after 8 degrees. On BlogCatalog, for DW and N2V, after little fluctuation, scores gradually decrease. However, for RiWalk and xNetMf, after a slight increase, the score curves fluctuate between 19% and 20%.

D. Graph Compression and Degree Increment

This section presents how graph size decreases and nodes' degree increase by compression for each dataset. In Table IV, symbol $|V^{d+}|$ represents number of nodes whose degrees increase after $5-degree$ compression. Besides, $\frac{\mathbf{d+}}{|\mathbf{V^{d+}}|}$ denotes the degree-increment per incremented node that is over 1.5 for all networks and more than 2 on Cora (2.28), CiteSeer (2.15) and DBLP networks (2.06). As we see in Table V, Cora, CiteSeer and DBLP networks obtain an impressive gain in both node (84.45%, 91.57% and 77.99%) and edge (67.89%, 75.15% and 59.46%) compression. The ratio is also notable for Wiki, 34.47% on vertices and 25.69% on edges. For scale-freeness and complex structure, compression of BlogCatalog is challenging. Although the vertex compression gain is negative, its edge compression achieves a sensible gain (19.42%).

V. Conclusion

This paper proposes a new graph compression method for efficient graph representation learning. Our approach uses degree-based compression and iterative dedensification to ensure effective graph compression for fluent graph embedding in different networks. Moreover, these learned representation vectors achieve notable Macro-F1 and Micro-F1 scores on single and multi-label classification tasks over the baselines. Hence, our method leverages efficient graph embedding on networks and performs competently on node classification tasks without losing the effectiveness of the originals.

Acknowledgment

This work is funded partially by National Science Foundation (NSF) under Grant No 2104720.

TABLE IV: Degree Increment Statistics after 5-degree compression

Cora		CiteSeer		Wiki		DBLP		BlogCatalog	
$\|\mathbf{V^{d+}}\|$	$\frac{\mathbf{d+}}{\|\mathbf{V^{d+}}\|}$	$\|\mathbf{V^{d+}}\|$	$\frac{\mathbf{d+}}{\|\mathbf{V^{d+}}\|}$	$\|\mathbf{V^{d+}}\|$	$\frac{\mathbf{d+}}{\|\mathbf{V^{d+}}\|}$	$\|\mathbf{V^{d+}}\|$	$\frac{\mathbf{d+}}{\|\mathbf{V^{d+}}\|}$	$\|\mathbf{V^{d+}}\|$	$\frac{\mathbf{d+}}{\|\mathbf{V^{d+}}\|}$
171	2.28	61	2.15	121	1.66	173	2.06	372	1.84

TABLE V: Compression Ratio

	Cora		CiteSeer		Wiki		DBLP		BlogCatalog	
	$\|\mathbf{V}\|$	$\|\mathbf{E}\|$	$\|\mathbf{V}\|$	$\|\mathbf{E}\|$	$\|\mathbf{V}\|$	$\|\mathbf{E}\|$	$\|\mathbf{V}\|$	$\|\mathbf{E}\|$	$\|\mathbf{V}\|$	$\|\mathbf{E}\|$
Original	2708	5278	3264	4536	2405	12761	27199	66832	10312	333983
Compressed	421	1695	275	1127	1576	9483	5986	27097	11036	269128
Gain %	**84.45**	**67.89**	**91.57**	**75.15**	**34.47**	**25.69**	**77.99**	**59.46**	-7.02	**19.42**

REFERENCES

[1] E. Akbas and M. E. Aktas. Network embedding: on compression and learning. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 4763–4772. IEEE, 2019.

[2] E. Akbas and P. Zhao. Graph clustering based on attribute-aware graph embedding. In *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 109–131. Springer, 2017.

[3] G. Buehrer and K. Chellapilla. A scalable pattern mining approach to web graph compression with communities. In *Proceedings of the 2008 international conference on web search and data mining*, pages 95–106, 2008.

[4] S. Cao, W. Lu, and Q. Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 891–900, 2015.

[5] H. Chen, B. Perozzi, Y. Hu, and S. Skiena. Harp: Hierarchical representation learning for networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[6] Y. W. Z. Z. Z. F. Chenhui Deng, Zhiqiang Zhao. Graphzoom: A multi-level spectral approach for accurate and scalable graph embedding. In *In ICLR 2020*, 2020.

[7] L. Dhulipala, I. Kabiljo, B. Karrer, G. Ottaviano, S. Pupyrev, and A. Shalita. Compressing graphs and indexes with recursive graph bisection. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1535–1544, 2016.

[8] C. Donnat, M. Zitnik, D. Hallac, and J. Leskovec. Learning structural node embeddings via diffusion wavelets. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1320–1329, 2018.

[9] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *the Journal of machine Learning research*, 9:1871–1874, 2008.

[10] W. Fan. Graph pattern matching revised for social network analysis. In *Proceedings of the 15th International Conference on Database Theory*, pages 8–21, 2012.

[11] W. Fan, Y. Li, M. Liu, and C. Lu. Making graphs compact by lossless contraction. In *Proceedings of the 2021 International Conference on Management of Data*, pages 472–484, 2021.

[12] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[13] M. Heimann, H. Shen, T. Safavi, and D. Koutra. Regal: Representation learning-based graph alignment. In *Proceedings of the 27th ACM international conference on information and knowledge management*, pages 117–126, 2018.

[14] S. Houghten, A. Romualdo, T. K. Collins, and J. A. Brown. Compression of biological networks using a genetic algorithm with localized merge. In *2019 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, pages 1–8. IEEE, 2019.

[15] M. I. Islam, F. Tanvir, G. Johnson, E. Akbas, and M. E. Aktas. Proximity-based compression for network embedding. *Frontiers in big Data*, 3:608043, 2021.

[16] Y.-Y. Jo, M.-H. Jang, S.-W. Kim, and S. Park. Realgraph: A graph engine leveraging the power-law distribution of real-world graphs. In *The World Wide Web Conference*, WWW '19, page 807–817, New York, NY, USA, 2019. Association for Computing Machinery.

[17] K. U. Khan, T. N. Anh, M. R. Akhond, W. Nawaz, and Y.-K. Lee. Accelerating community-search problem through faster graph dedensification. In *2017 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 340–347. IEEE, 2017.

[18] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM transactions on Knowledge Discovery from Data (TKDD)*, 1(1):2–es, 2007.

[19] F. Li, Z. Zou, J. Li, and Y. Li. Graph compression with stars. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 449–461. Springer, 2019.

[20] J. Liang, S. Gurukar, and S. Parthasarathy. Mile: A multi-level framework for scalable graph embedding. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 15, pages 361–372, 2021.

[21] X. Ma, G. Qin, Z. Qiu, M. Zheng, and Z. Wang. Riwalk: Fast structural node embedding via role identification. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 478–487. IEEE, 2019.

[22] A. Maccioni and D. J. Abadi. Scalable pattern matching over compressed graphs via dedensification. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1755–1764, 2016.

[23] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[24] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of the eleventh ACM international conference on web search and data mining*, pages 459–467, 2018.

[25] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 385–394, 2017.

[26] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[27] A. Saxena, G. Fletcher, and M. Pechenizkiy. Nodesim: node similarity based network embedding for diverse link prediction. *EPJ Data Science*, 11(1):24, 2022.

[28] H. Sun, F. He, J. Huang, Y. Sun, Y. Li, C. Wang, L. He, Z. Sun, and X. Jia. Network embedding for community detection in attributed networks. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 14(3):1–25, 2020.

[29] H. Sun, F. He, J. Huang, Y. Sun, Y. Li, C. Wang, L. He, Z. Sun, and X. Jia. Network embedding for community detection in attributed networks. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 14(3):1–25, 2020.

[30] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077, 2015.

[31] S. Wold, K. Esbensen, and P. Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.