Minimum Entropy Principle Guided Graph Neural Networks

Zhenyu Yang
Ge Zhang
Jia Wu
Jian Yang
Quan Z. Sheng
Macquarie University
{zhenyu.yang3,ge.zhang5}@hdr.mq.edu.au
{jia.wu,jian.yang,michael.sheng}@mq.edu.au

Hao Peng
Angsheng Li
Beihang University
{penghao,angsheng}@buaa.edu.cn

Shan Xue
University Of Wollongong
sxue@uow.edu.au

Jianlin Su
Shenzhen Zhuiyi Technology Co., Ltd.
bojonesu@wezhuiyi.com

ABSTRACT

Graph neural networks (GNNs) are now the mainstream method for mining graph-structured data and learning low-dimensional node- and graph-level embeddings to serve downstream tasks. However, limited by the bottleneck of interpretability that deep neural networks present, existing GNNs have ignored the issue of estimating the appropriate number of dimensions for the embeddings. Hence, we propose a novel framework called **M**inimum **G**raph **E**ntropy principle-guided **D**imension **E**stimation, i.e. MGEDE, that learns the appropriate embedding dimensions for both node and graph representations. In terms of node-level estimation, a minimum entropy function that counts both structure and attribute entropy, appraises the appropriate number of dimensions. In terms of graph-level estimation, each graph is assigned a customized embedding dimension from a candidate set based on the number of dimensions estimated for the node-level embeddings. Comprehensive experiments with node and graph classification tasks and nine benchmark datasets verify the effectiveness and generalizability of MGEDE.

CCS CONCEPTS

• **Computing methodologies** → **Artificial intelligence**; • **Information systems** → **Data mining**.

KEYWORDS

Dimension estimation; graph neural network; graph entropy; node embedding; graph embedding

ACM Reference Format:
Zhenyu Yang, Ge Zhang, Jia Wu, Jian Yang, Quan Z. Sheng, Hao Peng, Angsheng Li, Shan Xue, and Jianlin Su. 2023. Minimum Entropy Principle Guided Graph Neural Networks. In *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining (WSDM '23), February 27-March 3, 2023, Singapore, Singapore.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3539597.3570467

WSDM '23, February 27-March 3, 2023, Singapore, Singapore

ACM ISBN 978-1-4503-9407-9/23/02…$15.00
https://doi.org/10.1145/3539597.3570467

1 INTRODUCTION

GNNs are currently the most popular graph mining methods for learning low-dimensional node or graph embeddings to serve downstream machine learning tasks, such as classification [14, 17, 44], clustering [4, 36, 51], regression [12, 33]. The relevant theories have already been applied to a range of real-world applications [16, 43, 49]. Here, the general rule is that the dimensionality of the embeddings affects the quality of the encoded semantics and the ultimate performance of the GNN. A small number of dimensions will typically result in semantic loss, while a large number of dimensions will lead to overfitting and issue with computational inefficiency [23, 28, 47]. Hence, estimating the proper dimensionality for the embeddings produced is a crucial part of harnessing the power of a GNN yet one that has seldom been studied.

Estimating this critical parameter needs to be done manually in current GNNs. But there are two challenges confronting one's guess. First, the current theoretical research [22, 26] on GNNs focuses on how to embed structural and attribute information; the issue of how to estimate an appropriate embedding dimension has not been addressed. In practical terms, practitioners tend to select the proper dimensionality through a manual enumeration search. But these types of grid searches are time-consuming and computationally expensive. The whole process is not very efficient and black-box explainability issues can be confounding. Fig. 1 (A) illustrates the problem. That said, a few studies on word embeddings have been published where a suitable dimensionality is estimated via bias-, variance-, or entropy-based metrics [10, 35, 41, 47]. Further, Luo et al. [20] used the above metrics to produce embeddings with suitable dimensionality for the nodes in a graph. The downside of this technique is, however, that the process overlooks the rich structural information in graphs. Second, different graphs will have different proper embedding dimensions. Yet current graph-level GNNs ignore this diversity and encode all graphs with a unified embedding dimension (see Fig. 1 (B)). Hence, enabling GNNs to

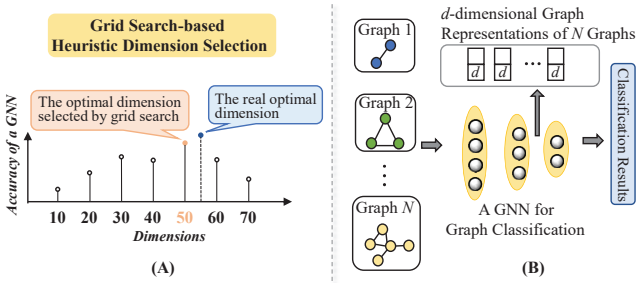

Figure 1: In (A), a grid search method parses a GNN seven times on each of seven embedding dimensions and then selects the dimensionality with the highest accuracy. By contrast, MGEDE estimates the proper dimensionality without needing to parse a GNN. In (B), a GNN generates embeddings for N graphs with a unified dimensionality of d, which ignores the differences between the graphs.

encode graphs with a range of different embedding dimensions for graph-level tasks is the second challenge.

Motivated by the minimum entropy principle [11], which indicates systems with minimum levels of uncertainty, we propose a novel framework called **M**inimum **G**raph **E**ntropy Principle-guided **D**imension **E**stimation, or MGEDE for short. MGEDE is designed to estimate the appropriate embedding dimensionalities for graph-structured data and is highly applicable for GNNs. The framework comprises both a node-level embedding dimension estimator (NDE) and a graph-level embedding dimension estimator (GDE). The NDE includes a minimum graph entropy function that simultaneously models attribute and structure entropy. Multi-order topological structures are captured to solve the appropriate number of dimensions for all node embeddings. The uncertainty within the node embeddings given different dimensionalities is approximately measured as attribute entropy based on the distributional hypothesis [29]. Meanwhile, an encoding tree [15] that naturally forms a hierarchical partition of the graph is used to calculate the structure entropy. The GDE module includes a new assignment mechanism that assigns each graph with a customized embedding dimension from a candidate set. The candidate set is assembled from the node-level embedding dimensions estimated by the NDE. Further, to embed the graphs into diversely customized dimensional spaces, we built a new training framework targeting graph-level embeddings from GNNs. For those who wish to reproduce our work, MGEDE is available at https://github.com/MGEDE.

The main contributions of this article include:

- A novel embedding dimension estimation framework called MGEDE, which is based on theoretical minimum entropy. MGEDE estimates a suitable number of dimensions for node- and graph-level embeddings, supporting GNNs to deliver competitive performance with downstream tasks.
- A new structure entropy that measures the complexity of a graph's structure by capturing the graph's multi-order topological information.
- A new assignment mechanism that assigns each graph embedding with a customized number of dimensions from a candidate set.
- Extensive experiments demonstrate that MGEDE supports GNNs and network embedding algorithms to deliver promising performance on node and graph classification tasks in terms of effectiveness.

2 DEFINITIONS

Definition 1. (Graph). A graph $G = (\mathcal{V}, \mathcal{E})$ comprises a node set $\mathcal{V}$ and an edge set $\mathcal{E}$. For G, the adjacency matrix is $\mathbf{A} \in \mathbb{R}^{n \times n}$, and $\mathbf{X} \in \mathbb{R}^{n \times f}$ is the node attribute matrix. If $\forall \mathbf{A}_{i,j} = 1$, $A_{j,i} = 1$, the graph G is an undirected graph, otherwise it is directed.

Definition 2. (Minimum Entropy). Given a variable X containing n states, n codewords are used to depict each state. In this case, Shannon entropy [31] denotes the lower bound of the average length of the codewords, which is $H(X) = -\sum_{i=1}^{n} p_i log p_i$, where p_i is the probability of the state occurrence. Hence, the graph entropy equals the average length of the codewords used to describe the graph. Minimizing the graph's entropy is equivalent to seeking the minimum length of a graph's description. Correspondingly, a graph that can be described by briefer and shorter codewords has less uncertainty.

Definition 3. (Structure Entropy). Structure entropy is a metric for measuring the complexity of a graph's topology [15]. It represents the average length of the codewords, where each codeword depicts a random walk on the graph. Under a specific encoding scheme, structural entropy assigns a prefix codeword to each hierarchy (e.g., community) in the graph to shorten the average length of codewords. For example, the codeword of a random walk visiting any node in the community can be shortened through the prefix codeword of the community. Using prefix codewords, we only need to encode the in-community nodes. The out-of-community nodes can be ignored. The graph $G = (\mathcal{V}, \mathcal{E})$ is then encoded into a three-layer tree $\mathcal{T}$ (as shown in ④ of Fig. 2), in which the root node γ denotes the whole graph, the tree nodes in the second layer denote the communities of the graph, and nodes in each community comprise the third layer (i.e., the leaf nodes of the tree). The adjacency matrix $\mathbf{A}$ represents the structure of G, the structure entropy is defined as:

$$H_S(\mathbf{A}) = -\sum_{\alpha \in \mathcal{T}, \alpha \neq \gamma} \frac{g_\alpha}{VOL(\gamma)} log \frac{VOL(\alpha)}{VOL(\alpha^+)}, \tag{1}$$

where α is a non-root tree node, which has a father tree node α^+. $VOL(\alpha)$ is the degree summation of all leaf nodes in the subtree rooted at α. For the root node γ, $VOL(\gamma)$ is the degree summation of all leaf nodes in $\mathcal{T}$. g_α represents the number of edges in G which only have one end node in the subtree rooted as α. The probability of a random walk visiting tree node α is $\frac{g_\alpha}{VOL(\gamma)}$, and -log$\frac{VOL(\alpha)}{VOL(\alpha^+)}$ is equal to the length of codeword that encodes α in α^+.

3 METHODOLOGY

MGEDE comprises two methods: NDE, which estimates the dimensionality of the node-level embeddings, and GDE, which estimates the dimensionality of the graph-level embeddings. NDE can estimate appropriate dimensionalities for both undirected and

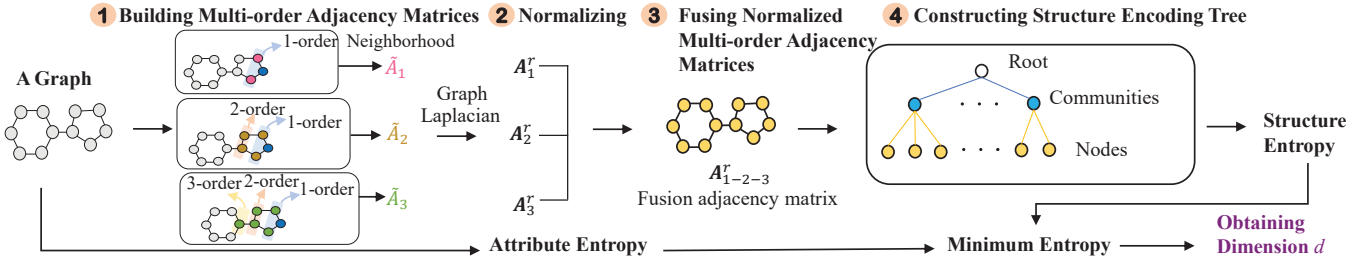

Figure 2: The figure demonstrates how NDE estimates the appropriate number of dimensions for the node embedding d and the four steps involved in calculating our novel structure entropy.

directed graphs (see section 3.1 and section 3.2, respectively). The GDE method estimates the candidate set of dimensionalities for the graph-level embeddings and selects the best fit for each graph (see section 3.3). A time complexity analyses of each method can be found in section 3.4.

3.1 Node Representation Dimension Estimator

Motivated by the minimum entropy principle (see Definition 2), NDE estimates the appropriate dimensionality of the node-level embeddings by minimizing the graph entropy H_G, which is defined as:

$$H_G = H_{Att} + H_S, \tag{2}$$

where H_{Att} denotes the attribute entropy and H_S denotes the structure entropy. Our explanation of how H_G is computed begins with a detailed, explanation of how H_{Att} and H_S are calculated. A function over d is then used to approximate H_G. The appropriate dimensionality is then the value of d that results in the minimum H_G.

3.1.1 ***Attribute entropy.*** Treating all nodes in the graph as isolated units, attribute entropy H_{Att} measures the amount of uncertainty present in the collection of all node attributes. Originally, there is an assumption that each node has a d-dimensional vectorized node embedding to encode the node attribute. Referring to previous studies on computing the entropy of a set of word embeddings [35, 47], we denote a pair-wise inner product among the assumed node embeddings as the basic unit for calculating H_{Att}, formally:

$$P(z_i, z_j) = \frac{e^{<z_i \bullet z_j>}}{\sum_{i,j} e^{<z_i \bullet z_j>}}, \tag{3}$$

where z_i and z_j represent the embeddings of the nodes i and j, and $< . \bullet . >$ is the dot product. Denoted as $S = \sum_{i,j} e^{<z_i \bullet z_j>}$, the attribute entropy is calculated as follows:

$$\begin{aligned} H_{Att} &= -\sum_{i,j} P(z_i, z_j) \log P(z_i, z_j) = -\sum_{i,j} \frac{e^{<z_i \bullet z_j>}}{S} \log \frac{e^{<z_i \bullet z_j>}}{S} \\ &= \log S - \frac{1}{S} \sum_{i,j} e^{<z_i \bullet z_j>} < z_i \bullet z_j > . \end{aligned} \tag{4}$$

Using n as the variable to represent the number of nodes in the graph, S and H_{Att} can be expressed as:

$$\begin{aligned} S &= \sum_{i,j} e^{<z_i \bullet z_j>} = n^2 \frac{1}{n^2} \sum_{i,j} e^{<z_i \bullet z_j>} \approx n^2 E_{z_i, z_j}[e^{<z_i \bullet z_j>}], \\ &\sum_{i,j} e^{<z_i \bullet z_j>} < z_i \bullet z_j > \approx n^2 E_{z_i, z_j}[e^{<z_i \bullet z_j>} < z_i \bullet z_j >], \\ H_{Att} &\approx \log n^2 + \log E_{z_i, z_j}[e^{<z_i \bullet z_j>}] - \frac{E_{z_i, z_j}[e^{<z_i \bullet z_j>} < z_i \bullet z_j >]}{E_{z_i, z_j}[e^{<z_i \bullet z_j>}]}. \end{aligned} \tag{5}$$

Yet, the value of $< z_i \bullet z_j >$ can not be directly obtained since both z_i and z_j are assumed embeddings not node attributes. To tackle this issue, an approximate calculation is made. As an empirical observation of the node embedding models in past experiments, we find the absolute values of each element in the vectorized node embeddings are uniformly distributed. According to the distributional hypothesis [29], we assume that each element in d-dimensional vectorized node embeddings has an absolute value of one. That is, in d-dimensional space, each node embedding maps to a vector that lies on the surface of a hyper-sphere with radius $\sqrt{d * 1}$. Subsequently, the approximate of $< z_i \bullet z_j >$ can be shown as:

$$< z_i \bullet z_j > = |z_i||z_j| \cos\theta = \sqrt{d}\sqrt{d} \cos\theta = d \cos\theta, \tag{6}$$

where θ is the angle between the vectors of z_i and z_j. Combining Eqs. (5) and (6) establishes a function to calculate the attribute entropy with a dimension of d and θ:

$$H_{Att} \approx \log n^2 + \log E_\theta[e^{d \cos\theta}] - \frac{E_\theta[e^{d \cos\theta} d \cos\theta]}{E_\theta[e^{d \cos\theta}]}. \tag{7}$$

Referring to previous research [9, 34], the probability density of the angle θ between two arbitrary vectors on the surface of a hyper-sphere with radius $\sqrt{d}$ in d-dimensional space is $p_n(\theta) = \frac{\Gamma\left(\frac{d}{2}\right)}{\Gamma\left(\frac{d-1}{2}\right)\sqrt{\pi}} \sin^{d-2}\theta$. Here, the H_{Att} is denoted as:

$$\begin{aligned} H_{Att} &\approx \log n^2 + \log \int_0^\pi p_n(\theta) e^{d \cos\theta} d\theta - \frac{\int_0^\pi p_n(\theta) e^{d \cos\theta} d \cos\theta d\theta}{\int_0^\pi p_n(\theta) e^{d \cos\theta} d\theta} \\ &\approx \log n^2 + \log \frac{\int_0^\pi \sin^{d-2}\theta e^{d \cos\theta} d\theta}{\int_0^\pi \sin^{d-2}\theta d\theta} - d \frac{\int_0^\pi \sin^{d-2}\theta e^{d \cos\theta} \cos\theta d\theta}{\int_0^\pi \sin^{d-2}\theta e^{d \cos\theta} d\theta}. \end{aligned} \tag{8}$$

According to the Laplace approximation [1], small changes between d and $(d-2)$ can be ignored when d is large enough. Formally:

$$\log\left[\sin^{d-2}\theta e^{d\cos\theta}\right] = (d-2)\log\sin\theta + d\cos\theta \approx d(\log\sin\theta + \cos\theta). \tag{9}$$

The maximum value of $(\log\sin\theta + \cos\theta)$ is achieved by $\theta = \arctan\sqrt{\frac{\sqrt{5}+1}{2}} \approx 0.905$, thus, $(\log\sin\theta + \cos\theta) \approx 0.377 - 1.12(\theta - 0.905)^2$. That is:

$$\begin{aligned}\int_0^{\pi}\sin^{d-2}\theta e^{d\cos\theta}d\theta &= \int_0^{\pi} e^{\log\left[\sin^{d-2}\theta e^{d\cos\theta}\right]}d\theta \\ &\approx \int_{-\infty}^{\infty} e^{d\left[0.377-1.12(\theta-0.905)^2\right]}d\theta \approx \frac{1.676}{\sqrt{d}}e^{0.377d}.\end{aligned} \tag{10}$$

$$\begin{aligned}&\frac{\int_0^{\pi}\sin^{d-2}\theta e^{d\cos\theta}\cos\theta d\theta}{\int_0^{\pi}\sin^{d-2}\theta e^{d\cos\theta}d\theta} \approx \frac{\int_{-\infty}^{\infty} e^{d\left[0.377-1.12(\theta-0.905)^2\right]}\cos\theta d\theta}{\int_{-\infty}^{\infty} e^{d\left[0.377-1.12(\theta-0.905)^2\right]}d\theta} \\ &\approx \int_{-\infty}^{\infty}\delta(\theta-0.905)\cos\theta d\theta = \cos 0.905 \approx 0.618.\end{aligned} \tag{11}$$

Similarly, a Laplace approximation [1] is used to calculate $\int_0^{\pi}\sin^{d-2}\theta d\theta \approx \frac{2.507}{\sqrt{d}}$. Lastly, the attribute entropy H_{Att} is:

$$H_{Att} \approx \log n^2 + \log\frac{\frac{1.676}{\sqrt{d}}e^{0.377d}}{\frac{2.507}{\sqrt{d}}} - 0.618d \approx \log n^2 - 0.24d. \tag{12}$$

3.1.2 ***Structure entropy.*** Structure entropy $H_S(\mathbf{A})$ is defined in Eq. (1), as reflecting the complexity of the topological information contained in $\mathbf{A}$. The NDE method includes a novel form of structure entropy $H_S(\mathbf{A}^r_{1-2-3})$, that is specifically designed for GNNs, where the $H_S(\cdot)$ defined in Eq. (1) is used on a normalized adjacency matrix $\mathbf{A}^r_{1-2-3}$ containing multi-order link information. There are four steps to calculating $H_S(\mathbf{A}^r_{1-2-3})$ (see Fig. 2). Each step is explained in detail next along with why we have used $\mathbf{A}^r_{1-2-3}$ instead of $\mathbf{A}$.

Step 1. Computing the Multi-order Adjacency Matrices. GNNs capture multi-order link structures separately in multi-layer convolutional layers. Thus, a multi-order adjacency matrices is used to calculate structure entropy instead of only a first-order adjacency matrix $\mathbf{A}$. Given an undirected graph, the first-order adjacency matrix is $\mathbf{A}$. Similar to GNNs, a self-loop is added into $\mathbf{A}$ to yield $\tilde{\mathbf{A}}$, i.e., $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$. The second-order adjacency matrix is $\tilde{\mathbf{A}}_2 = \tilde{\mathbf{A}}^T\tilde{\mathbf{A}}$, and the third-order adjacency matrix is regarded as $\tilde{\mathbf{A}}_3 = \tilde{\mathbf{A}}\tilde{\mathbf{A}}^T\tilde{\mathbf{A}}$.

Step 2. Applying Graph Laplacian Normalization to the Multi-order Adjacency Matrices. Inspired by graph Laplacian [14], we hope to capture the information transmission rate between two nodes to replace the explicit weight of the edge. The implicit information transmission rate establishes a more stable random walk probability distribution on the graph, so as to calculate the structure entropy more precisely. Therefore, we employ graph Laplacian to normalize the multi-order adjacent matrices $\tilde{\mathbf{A}}_i, i = 1, 2, 3$. The normalized multi-order adjacent matrices $\mathbf{A}^r_i$ are defined as $\mathbf{A}^r_i = \tilde{\mathbf{D}}_i^{-\frac{1}{2}}\tilde{\mathbf{A}}_i\tilde{\mathbf{D}}_i^{-\frac{1}{2}}$, where $\tilde{\mathbf{D}}_i$ indicates the diagonal degree matrix of $\tilde{\mathbf{A}}_i$, and $\mathbf{A}^r_i[u][v]$ represents the information transfer rate between node u and v. Next, we multiply $\mathbf{D}_i$ with $\mathbf{A}^r_i$ to obtain $\mathbf{D}^r_i$, that is, $\mathbf{D}^r_i = \mathbf{D}_i\mathbf{A}^r_i$, where each matrix element $\mathbf{D}^r_i[u][v]$ denotes the amount of information transformation from node u to node v. The normalized degree of node u is $d^r_i[u]$, where $d^r_i = \sum_x \mathbf{D}^r_i[x,:]$.

Step 3. Fusing the Normalized Multi-order Adjacency Matrices. In most cases, GNN will only set several layers to avoid over-smoothing. Therefore, in this setting, only the normalized 1-order, 2-order, and 3-order adjacency matrices $\mathbf{A}^r_1$, $\mathbf{A}^r_2$, and $\mathbf{A}^r_3$ are used to compute the structure entropy. However, there is much redundant structural information in $\mathbf{A}^r_1$, $\mathbf{A}^r_2$, and $\mathbf{A}^r_3$. Hence, if we fuse them, we should yield an adjacency matrix $\mathbf{A}^r_{1-2-3}$ with reduced redundancy. $\mathbf{A}^r_2$ and $\mathbf{A}^r_3$ reflect the second- and third-layered graph convolutions in the GNNs. In our experiments, we found that the structure entropy of $\mathbf{A}^r_2$ and $\mathbf{A}^r_3$ tended to be larger than $\mathbf{A}^r_1$ since they represent more complex structural information. Meanwhile, $\mathbf{A}^r_2$ and $\mathbf{A}^r_3$ bring more redundancy and over-smoothing. Hence, for a better fusion, we assigned different probabilities to $\mathbf{A}^r_1$, $\mathbf{A}^r_2$, and $\mathbf{A}^r_3$, where the probability of the normalized adjacency matrix under the order i is calculated as:

$$p_i = log\frac{2\cdot\Delta}{|H_S(\mathbf{A}^r_i) - H_S(\mathbf{A}^r_1)| + \Delta}, \quad \Delta = \sum_i |H_S(\mathbf{A}^r_i) - H_S(\mathbf{A}^r_1)|, \tag{13}$$

as shown in the above equation, $H_S(\cdot)$ as defined in Eq. (1) is used to calculate the structure entropy of $\mathbf{A}^r_i$. However, in a slight variation on Eq. (1) for NDE, g_α denotes the summation of $\mathbf{D}^r_i[u][v]$, u is the node not in the subtree rooted at α, and v is the node in the partition of subtree rooted at α.

Since $\mathbf{A}^r_1$ contain the minimum over-smoothing information, it is regarded as the basis of the probability calculation process. The smoothing degree of the higher-order normalized adjacency matrix (≥ 1) is reflected in the differences between the structure entropy themselves and $\mathbf{A}^r_1$'s. Therefore, Eq. (13) assigns the lower probability to the higher-order normalized adjacency matrix (≥ 1) that is the most smooth. Such an approach to probability assignment is referred to as Inverse Document Frequency (IDF) [13]. The fusion adjacency matrix and the corresponding degree set are then defined as:

$$\mathbf{A}^r_{1-2-3} = \sum_{i=1}^{3}\mathbf{A}^r_i \cdot p_i, \qquad d^r_{1-2-3} = \sum_{i=1}^{3} d^r_i \cdot p_i. \tag{14}$$

Step 4. Calculating High-level Structure Entropy. According to Eq. (14), structure entropy is calculated through:

$$H_S(\mathbf{A}^r_{1-2-3}). \tag{15}$$

We used the Python toolkit Louvain [2] to obtain the community partition and the encoding tree for $\mathbf{A}^r_{1-2-3}$. We chose the Louvain algorithm because it is fast and performs well [18].

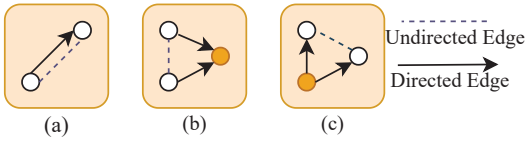

Figure 3: This figure demonstrates how to transform the directed edge to the undirected edge.

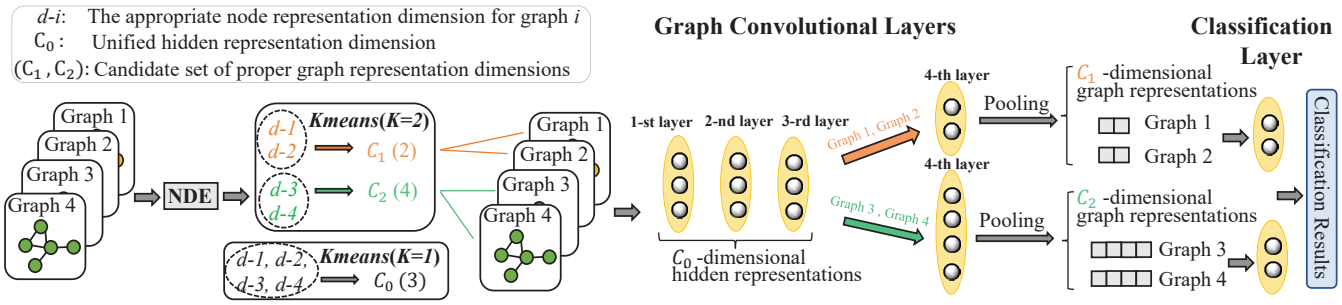

Figure 4: Taking four graphs as an example, this figure demonstrates how GDE estimates the appropriate graph representation dimensions for graphs and the new training framework of GNNs for graph classification. Based on NDE and K-means, the unified hidden representation dimension estimated by GDE is C_0, and the proper graph representation dimension of Graphs 1 and 2, Graphs 3 and 4 estimated by GDE is C_1 and C_2, respectively. Then, a GNN model represents Graphs 1 and 2 and Graphs 3 and 4 as C_1- dimensional and C_2-dimensional graph representations, respectively, and outputs the classification results.

3.1.3 ***Minimum entropy*** & ***Appropriate Dimension***. Based on Eqs. (2), (12), and (15), the graph entropy of the graph G is:

$$H_G \approx log n^2 - 0.24d + H_S(\mathbf{A}^r_{1-2-3}), \tag{16}$$

where d denotes the dimensionality of the node embedding and n is the number of nodes in the graph G. The appropriate node representation dimension d is the one that makes $H_G = 0$.

3.2 A Variant: Node Representation Dimension Estimator for Directed Graphs.

We also built a variant of NDE that can estimate the appropriate dimensionality of node embeddings for directed graphs. This is because the idea of using a fused adjacency matrix to do the estimation cannot be directly applied to directed graphs. The reason is that graph Laplacian operations are only applicable to symmetric matrices in theory, but the adjacency matrix of a directed graph is asymmetric. Hence, in this variant of NDE, we must build a symmetric adjacency matrix for the directed graph, that is:

$$\mathbf{A}_{sym}[i,j] = max(\mathbf{A}[i,j], \mathbf{A}[j,i]), \tag{17}$$

$$\mathbf{A}_{out}[i,j] = \sum_{u \in \mathcal{N}_{out}(i,j)} \frac{\mathbf{A}[i,u] + \mathbf{A}[j,u]}{\sum_v \mathbf{A}[v,u]}, \tag{18}$$

$$\mathbf{A}_{in}[i,j] = \sum_{u \in \mathcal{N}_{in}(i,j)} \frac{\mathbf{A}[u,i] + \mathbf{A}[u,j]}{\sum_v \mathbf{A}[u,v]}, \tag{19}$$

where $\mathbf{A}$ is the asymmetric adjacency matrix of the directed graph. Eq. (17) defines how to build the symmetric adjacency matrix for a directed graph, as shown in Fig. 3(a). To build a symmetric adjacency matrix that can encode the direction information in the directed graph, we also define $\mathbf{A}_{out}$ in Eq. (18). $\mathbf{A}_{out}$ connects two nodes that both have the out-degree edge to the same node. The process is illustrated in Fig. 3(b). Similarly, we can connect two nodes that both have in-degree edges from the same node in Eq. (19) to obtain $\mathbf{A}_{in}$. The process is shown in Fig. 3(c). Here, $\mathcal{N}_{out}(i,j)$ denotes the nodes that have in-degree edges from the nodes i, j while $\mathcal{N}_{in}(i,j)$ denotes the set of nodes that have out-degree edges to nodes i, j.

The graph Laplacian can then be used to normalize $\mathbf{A}_{sym}$, $\mathbf{A}_{out}$ and $\mathbf{A}_{in}$, and fuse them with probabilities to obtain the fusion adjacency matrix of a directed graph. $\mathbf{A}_{sym}$ is the basis in the fusion process. Next, in the same way, as we estimate the embedding dimensionality with undirected graphs, we can derive the appropriate dimensionality with a directed graph.

3.3 Graph Representation Dimension Estimator

Most GNNs for graph-level tasks generate graph-level embeddings by aggregating the embeddings of all the nodes. That is to say, in these GNNs, node and graph representations share the same dimensional space. Hence, the appropriate dimensionalities of the node-level embeddings as estimated by the NDE should also be valuable for determining the proper dimensionality of a graph-level embedding. Thus, the GDE assembles a candidate set of dimensions from the NDE. Then, for each graph, the GDE selects the best-fit dimensionality from the candidate set.

To assemble the candidate set of dimensions, the NDE method is applied to each of the graphs $\{G_1, ..., G_N\}$ in the given graph database $\mathbb{G}$. A vector $[d-1, ..., d-N]^\top$ containing all proper dimensionalities for each graph $\{G_1, ..., G_N\}$ is then fed into a 1-D K-means [21] clustering algorithm, from which K clustering centroids, $C_1, ..., C_K$ are derived. The value of K is a predefined hyper-parameter, and $\{C_1, ..., C_K\}$ is the candidate set of proper dimensionalities for the graph-level embedding.

The GDE method also includes a new training framework, specifically designed to support GNNs for graph-level tasks. The method is, outlined in Fig. 4. In this training framework, all graph convolution layers except for the last layer encode the graphs into a unified dimensionality C_0 for the hidden embedding. This is done by applying another 1-D K-means to the vector $[d-1, ..., d-N]^\top$ with $K = 1$. In the last convolution layer, the graphs are assigned with their best-fit dimensionality $C_i, i \geq 1$.

To summarize, GDE learns $K + 1$ dimensions $(C_0; C_1, ..., C_K)$ for a graph database, C_0 is the unified dimensionality for the hidden embedding, and $\{C_1, C_2, ..., C_k\}$ is the candidate set of proper dimensionality for each graph's representation.

Table 1: The performance of the GNNs with supervised graph classification in terms of accuracy (%). Max, Min, and Avg denote the maximum, minimum, and average performance gain of GNNs resulting from the dimensions estimated by MGEDE compared to the selecting dimensions heuristically. The best results are highlighted in bold.

Data.		ENZYMES					D&D					
	Dim.	SOPOOL	DGCNN	GIN	DIFFP	GraSAG	Dim.	SOPOOL	DGCNN	GIN	DIFFP	GraSAG
Heuristic	16	51.0±4.4	38.6±5.8	42.8±5.0	58.5±4.3	50.8±6.6	48	77.1±5.1	77.8±3.5	76.0±5.3	77.9±2.3	74.7±3.8
	32	52.6±6.3	43.1±9.6	43.8±5.1	62.3±2.9	54.7±6.3	64	77.4±5.5	78.1±4.1	77.3±6.2	80.3±3.6	75.3±4.3
	48	53.0±6.2	45.8±5.3	45.8±5.4	63.2±8.6	60.0±7.9	80	76.3±4.8	77.9±3.1	76.8±5.3	79.3±3.7	75.1±3.0
	64	52.0±6.1	44.9±7.6	45.3±6.1	62.2±4.7	55.7±6.8	96	76.1±4.2	77.6±3.6	76.5±4.1	78.8±3.3	74.6±4.6
	80	51.2±9.9	43.8±7.0	44.3±6.7	61.5±4.4	56.4±5.7	112	75.7±4.7	77.4±3.7	76.3±4.9	78.7±4.6	74.5±5.1
	96	51.1±5.6	43.6±6.5	43.6±6.8	61.3±4.8	55.1±7.6	128	75.3±5.4	77.4±3.6	75.8±4.4	78.3±3.3	74.0±3.1
MGEDE	(42;37,46)	**54.0±6.3**	**47.2±4.9**	**46.2±6.6**	**64.1±9.0**	**60.9±5.1**	(66;59,72)	**78.1±3.6**	**78.6±1.9**	**78.0±2.2**	**80.7±2.4**	**75.8±2.9**
Max/Min/Avg		3/1/2.2	8.6/1.4/3.9	3.4/0.4/1.9	5.6/0.9/2.6	10.1/0.9/5.45		2.8/0.7/1.8	1.2/0.5/0.9	2.2/0.7/1.6	2.8/0.4/1.8	1.8/0.5/1.1

Data.		PROTEINS					COLLAB					
	Dim.	SOPOOL	DGCNN	GIN	DIFFP	GraSAG	Dim.	SOPOOL	DGCNN	GIN	DIFFP	GraSAG
Heuristic	16	74.4±3.9	73.1±5.4	74.7±4.6	75.5±4.3	74.6±4.4	48	74.0±2.2	72.5±2.2	79.2±1.5	73.7±1.6	68.9±2.4
	32	74.7±5.5	73.6±4.8	75.2±3.8	76.0±2.3	75.1±4.6	64	74.7±2.7	73.6±2.3	79.9±0.8	75.6±2.1	**71.1±2.2**
	48	74.9±2.5	74.0±4.7	75.3±4.2	76.2±2.8	75.3±5.4	80	74.5±2.1	73.2±1.7	79.6±1.3	74.5±1.3	70.0±1.4
	64	**75.0±4.5**	73.5±4.1	75.1±4.5	75.9±4.2	75.0±3.1	96	74.3±2.2	73.0±2.5	79.5±1.3	73.9±1.5	69.7±1.7
	80	74.9±4.1	73.2±5.2	74.6±2.1	75.8±3.3	73.2±3.2	112	74.3±1.6	72.7±1.9	79.3±1.6	73.8±1.9	68.1±1.1
	96	73.5±5.0	73.4±4.8	74.3±1.9	75.2±3.3	72.7±2.4	128	74.1±2.0	72.4±1.5	79.2±1.4	73.5±1.6	68.2±2.5
MGEDE	(41;35,50)	**75.0±4.2**	**74.4±5.2**	**75.8±3.0**	**76.7±3.7**	**75.9±5.2**	(57;53,68)	**77.1±1.7**	**74.6±1.4**	**80.4±1.2**	**76.4±1.5**	**71.1±0.9**
Max/Min/Avg		1.5/0.0/0.4	1.3/0.4/0.9	1.5/0.5/0.9	1.5/0.5/0.9	3.2/0.6/1.6		3.1/2.4/2.8	2.2/1/1.7	1.2/0.5/1.0	2.9/0.8/2.2	3.0/0.0/1.8

3.4 Time Complexity Analysis

The time complexity of NDE is mainly composed of the Louvain algorithm (O($nlogn$)), matrix normalization (O(n^2)), and the structure entropy calculation (O($e + n$)), where n and e are the number of nodes and edges, respectively. Overall, the time complexity of NDE is O(n^2). The time complexity of GDE is O($N \cdot n^2$), where N is the number of graphs in the graph database.

4 EXPERIMENTS

In this section, we report the outcomes of an extensive set of experiments with nine benchmark datasets. The experiments were designed to test the effectiveness of the dimensionalities estimated by MGEDE with both node and graph classification tasks.

4.1 Experiment Setup

Datasets. For the graph classification task, we used three biological datasets (i.e., ENZYMES [3], PROTEINS [3], D&D [6]) and a social network dataset (i.e., COLLAB [45]). There were 6, 2, 2, and 3 classes in the four datasets, respectively. With the node classification tasks, we use three undirected graphs (i.e., Cora [30], Citeseer [30], and Pubmed [27]) and three directed graphs (i.e., Cora-ML [32], Directed Citeseer (Di-Citeseer) [30], and AM-Computer [24]). They have 7, 6, 3, 7, 6 and 10 classes, respectively.

Models. We evaluated the MGEDE's performance in terms of how the estimated dimensionalities were with three types of popular GNNs and network embedding algorithms, including **a).** GNNs for supervised graph classification (i.e., GIN [44], SOPOOL [42], DGCNN [50], GraSAG [8] and DIFFP [48]); **b)**. GNNs for semi-supervised node classification on undirected graphs (GEN [40], GCN [14], GAT [39], and DAGNN [19]); **c).** GNNs for semi-supervised node classification on directed graphs (DiGCN [38], and DiGCL [37]); and **d).** Unsupervised network embedding algorithms (DANE [7] and CAN [25]). To the best of our knowledge, MinGE [20] is the only framework that can estimate the suitable number of dimensions for node-level embeddings with undirected graphs. Therefore, we take the grid-search-based heuristic method and MinGE as the node-level comparison methods with undirected graphs. There are no current methods estimating embedding dimensions for node-level embedding with directed graphs and graph-level embedding. Hence, for these two tasks, we could only use the grid-search-based heuristic method as the comparison method.

Evaluation Metrics. We used accuracy as our evaluation metric for supervised graph classification and semi-supervised node classification. This is because these two tasks are relatively balanced in the experimental setting with this paper, and accuracy is a widely used evaluation metric [8, 42, 44, 48, 50]. For the unsupervised network embedding, following previous studies [7, 25], we adopted Micro-F1 and Macro-F1 as the evaluation metrics. All the experimental results reported are from 10-fold cross-validations.

Experimental Setting. MGEDE has only one hyper-parameter, being K, and it only applies to graph classification tasks. K controls the size of the candidate set of proper dimensionalities for graph embeddings. In our experiments, we set $K = 2$.

4.2 Performance of GNNs on Supervised Graph Classification

Tab. 1 illustrates the performance of the GNNs and algorithms with supervised graph classification tasks. Here, the heuristic method is compared with MGEDE. In terms of maximum (max), minimum (min), and average (avg) performance gain. From the results for minimum performance gain, we can see that MGEDE performed competitively on all datasets. These results verify that MGEDE can support practitioners in their use of GNNs for graph classification tasks.

Table 2: The performance of GNNs on semi-supervised node classification on undirected graphs in terms of accuracy (%).

Data.		Cora					Citeseer					Pubmed			
	Dim.	GEN	GCN	GAT	DAGNN	Dim.	GEN	GCN	GAT	DAGNN	Dim.	GEN	GCN	GAT	DAGNN
	16	81.6±0.3	81.0±0.9	83.0±0.3	84.2±0.5	16	73.0±0.4	70.1±0.7	71.8±0.4	72.9±0.7	48	79.2±0.4	78.6±0.3	78.3±0.4	80.0±0.5
	32	81.5±0.4	81.1±0.4	83.5±0.3	84.2±0.5	32	73.1±0.5	70.0±0.6	72.0±0.4	73.0±0.6	64	79.3±0.6	78.7±0.2	78.8±0.5	80.1±0.4
	48	81.8±0.3	81.1±0.4	83.8±0.4	84.3±0.5	48	73.3±0.9	70.2±0.3	72.1±0.5	73.1±0.5	80	79.5±0.5	78.9±0.3	79.2±0.4	80.2±0.4
	64	81.8±0.5	81.2±0.6	84.1±0.4	84.4±0.5	64	73.6±0.4	70.3±0.4	72.3±0.5	73.1±0.6	96	79.8±0.4	78.9±0.3	79.3±0.3	80.2±0.5
Heur-	80	81.9±0.1	81.3±0.3	84.3±0.4	84.5±0.4	80	73.7±0.4	70.4±0.4	72.3±0.3	73.2±0.5	112	80.2±0.4	79.0±0.3	79.4±0.5	80.3±0.4
istic	96	82.1±0.2	81.4±0.3	84.5±0.5	84.6±0.5	96	73.9±0.3	70.6±0.5	72.4±0.3	73.3±0.5	128	80.7±0.3	79.2±0.5	79.5±0.4	80.4±0.4
	112	81.4±0.2	81.3±0.5	84.2±0.3	84.3±0.5	112	73.6±0.5	70.4±0.8	72.2±0.3	73.2±0.5	144	80.4±0.5	78.9±0.2	79.3±0.4	80.2±0.5
	128	81.5±0.2	81.2±0.4	83.9±0.4	84.1±0.6	128	73.4±0.5	70.5±0.4	72.0±0.5	73.1±0.5	160	80.3±0.3	78.8±0.3	79.0±0.5	80.2±0.4
	144	81.0±0.4	81.1±0.5	84.0±0.3	84.1±0.6	144	73.4±0.4	70.1±0.6	71.7±0.4	73.1±0.6	176	80.1±0.4	78.7±0.3	78.9±0.2	80.1±0.4
	160	80.9±0.1	81.2±0.4	83.7±0.4	84.0±0.5	160	73.2±0.9	70.0±0.5	71.5±0.3	72.8±0.7	192	80.2±0.5	78.7±0.2	78.6±0.3	80.2±0.5
MinGE	98	82.4±0.3	81.4±0.4	84.5±0.4	84.7±0.6	101	73.8±0.3	70.6±0.4	72.3±0.4	73.4±0.5	123	80.6±0.3	79.2±0.4	79.4±0.4	80.3±0.4
MGEDE	101	**82.6±0.4**	**81.6±0.5**	**84.6±0.3**	**84.9±0.5**	99	**74.1±0.4**	**70.7±0.5**	**72.5±0.3**	**73.5±0.5**	130	**81.0±0.4**	**79.4±0.3**	**79.7±0.3**	**80.6±0.3**
Max/Min/Avg		1.7/0.2/1.0	0.6/0.2/0.4	1.6/0.1/0.6	0.9/0.2/0.6		1.1/0.2/0.6	0.7/0.1/0.4	1.0/0.1/0.4	0.7/0.1/0.4		1.8/0.3/1.0	0.8/0.2/0.5	1.4/0.2/0.6	0.6/0.2/0.4

Table 3: The performance of GNNs on semi-supervised node classification on directed graphs in terms of accuracy (%).

Data.		Cora-ML			Di-Citeseer				AM-Computer			
	Dim.	GCN	DiGCN	DiGCL	Dim.	GCN	DiGCN	DiGCL	Dim.	GCN	DiGCN	DiGCL
	16	70.2±0.6	76.5±0.1	72.6±2.1	16	62.6±0.8	63.0±0.2	61.3±1.7	48	82.4±0.6	84.5±0.6	66.9±1.7
	32	70.4±0.3	76.6±0.5	75.3±1.3	32	62.8±0.4	64.3±0.6	62.0±1.4	64	82.9±0.5	85.3±0.2	67.6±1.3
	48	70.5±0.4	76.9±0.4	75.7±1.8	48	62.9±0.3	64.5±0.6	64.2±1.8	80	82.9±0.7	85.3±0.3	68.5±1.3
	64	70.7±0.3	77.0±0.4	76.7±1.3	64	62.9±0.4	64.6±0.5	66.2±1.8	96	83.0±0.4	85.4±0.3	71.2±1.8
Heur-	80	70.9±0.3	77.0±0.4	77.1±1.4	80	63.0±0.3	64.7±0.4	66.5±1.4	112	83.1±0.6	85.5±0.3	72.5±2.1
istic	96	71.1±0.3	77.1±0.3	77.5±1.1	96	63.1±0.4	64.8±0.5	66.1±1.4	128	**83.4±0.5**	**85.8±0.3**	**73.2±2.2**
	112	71.0±0.3	77.0±0.3	77.3±1.2	112	62.8±0.4	64.7±0.5	61.8±1.3	144	83.0±0.5	85.6±0.3	67.6±1.7
	128	70.8±0.2	76.9±0.4	76.7±2.1	128	62.9±0.4	64.5±0.4	60.9±0.7	160	82.9±0.3	85.5±0.3	65.7±2.0
	144	70.9±0.2	76.7±0.3	76.7±1.7	144	62.7±0.5	64.6±0.8	60.2±1.9	176	82.6±0.5	85.4±0.2	62.6±2.0
	160	70.7±0.2	76.6±0.4	75.6±1.1	160	62.4±0.3	64.4±0.5	57.8±0.8	192	82.2±0.5	84.8±0.3	60.5±2.1
MGEDE	101	**71.3±0.3**	**77.3±0.4**	**77.8±1.2**	91	**63.2±0.3**	**64.9±0.5**	**66.8±1.1**	128	**83.4±0.5**	**85.8±0.3**	**73.2±2.2**
Max/Min/Avg		1.1/0.2/0.6	0.8/0.2/0.5	5.2/0.3/1.7		0.8/0.1/0.4	1.9/0.1/0.5	9/0.3/4.1		1.2/0/0.6	1.3/0/0.5	12.7/0/5.6

4.3 Performance of GNNs on Semi-supervised Node Classification

Tab. 2 provides the results with a semi-supervised node classification task on undirected graphs. MGEDE helps the GNNs to deliver promising performance on all undirected graphs with an average of 0.4% greater accuracy. Tab. 3 reports the results with directed graphs. Likewise, on all directed graphs, the GNNs were more accurate when using the number of dimensions estimated by MGEDE.

4.4 Performance on Unsupervised Network Embedding

Tab. 4 shows the performance of the two network embedding models DANE and CAN. With a node classification task, in this experiment, MGEDE helped the two models reach competitive performance in terms of Micro-F1 and Macro-F1 scores.

4.5 Time Efficiency Analysis

Beyond improving the classification accuracy of GNNs, one of MGEDE's is that it saves practitioners time. First, it estimates the proper number of dimensions for graph embeddings in a much shorter time than a grid search method would take. Second, the GNNs themselves operate at a higher running efficiency because the dimensions chosen for the embeddings are appropriate. As shown in Tab. 5 and Fig. 5, MGEDE took 1347 seconds (22.45 minutes) to estimate the proper number of dimensionalities for the graph-level embeddings for the COLLAB dataset. The grid search method

Table 4: The performance of unsupervised network embedding in terms of Micro-F1 and Macro-F1.

Data.		Cora					Pubmed			
	Dim.	DANE		CAN		Dim.	DANE		CAN	
		Ma_F1	Mi_F1	Ma_F1	Mi_F1		Ma_F1	Mi_F1	Ma_F1	Mi_F1
	16	0.780	0.805	0.795	0.816	48	0.827	0.828	0.833	0.836
	32	0.804	0.817	0.853	0.866	64	0.835	0.834	0.834	0.842
	48	0.810	0.823	0.857	0.870	80	0.839	0.840	0.834	0.845
	64	0.813	0.827	0.859	0.871	96	0.841	0.840	0.837	0.845
Heur-	80	0.816	0.832	0.861	0.873	112	0.842	0.844	0.838	0.846
istic	96	0.824	0.834	0.862	0.876	128	0.847	0.846	0.842	0.848
	112	0.818	0.834	0.866	0.876	144	0.846	0.845	0.839	0.844
	128	0.816	0.831	0.863	0.875	160	0.845	0.843	0.786	0.789
	144	0.809	0.826	0.864	0.870	176	0.843	0.840	0.765	0.770
	160	0.804	0.824	0.860	0.867	192	0.841	0.837	0.766	0.765
MinGE	98	0.826	0.837	0.865	0.877	123	0.844	0.845	0.840	0.848
MGEDE	101	**0.828**	**0.842**	**0.869**	**0.881**	130	**0.849**	**0.848**	**0.843**	**0.850**
Max		0.048	0.037	0.074	0.065		0.022	0.020	0.078	0.085
Min		0.002	0.005	0.003	0.004		0.002	0.002	0.001	0.002
Avg		0.017	0.016	0.014	0.014		0.008	0.008	0.024	0.025

Table 5: Times taken for MGEDE to estimate the proper number of dimensions for graph-level embeddings with four graph databases.

Data.	ENZYMES	PROTEINS	D&D	COLLAB
Graphs	600	1113	1178	5000
AvgNode	32.6	39.1	284.3	74.49
AvgEdge	62.1	72.8	715.6	2457.7
Time(s)	13	26	373	1347

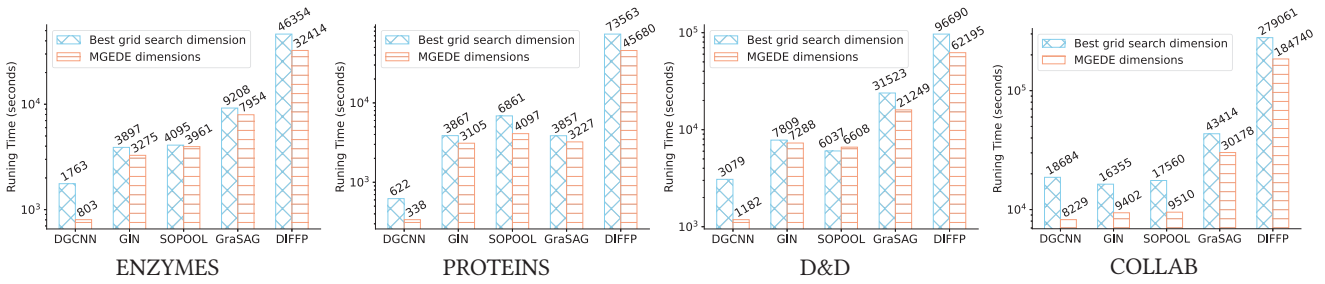

Figure 5: Running time(s) of GNNs with the best grid search dimension and the appropriate dimensions estimated by MGEDE.

Table 6: Graph classification accuracy with different value of the hyper-parameter K.

Dataset	ENZYMES					D&D				
K	2	3	4	5	6	2	3	4	5	6
Dims.	37,46	33,41,48	31,39,44,49	24,34,40,45,50	24,34,40,44,48,54	59,72	57,66,75	54,62,69,77	52,59,65,71,78	52,59,64,69,74,81
SOPOOL	54.0±6.3	33.3±6.8	30.4±4.5	27.7±4.2	27.2±5.7	78.1±3.6	76.3±4.1	74.8±3.9	75.1±4.0	75.0±3.7
Dataset	PROTEINS					COLLAB				
K	2	3	4	5	6	2	3	4	5	6
Dims.	35,50	30,41,55	28,38,47,58	27,36,43,51,61	25,33,39,46,54,63	53,68	52,62,78	51,57,65,80	50,55,60,67,80	49,53,57,62,68,80
SOPOOL	75.0±4.2	74.8±2.7	74.8±3.6	73.6±3.1	73.3±3.0	77.1±1.7	74.7±1.9	74.7±2.1	74.5±2.4	74.2±2.1

needed to parse the GNNs many times with each parse taking over 10000 seconds (2.8 hours). Further, as shown in Fig. 5, the blue bar denotes the running time of each GNN with the best grid search dimensions while the orange bar demonstrates the running times of the GNNs with dimensions estimated by MGEDE. It is clear that the running times for MGEDE were faster on almost all graph datasets.

4.6 Hyper-parameter Analysis

Using SOPOOL for a graph classification task, we undertook a sensitivity study of the hyper-parameter K as shown in Tab. 6. SOPOOL delivered the best performance on all graph datasets when $K = 2$, and its performance witnessed a steady decrease as K increased. One possible reason for this is that a larger K may lead the GNNs to overfit, especially with small-sized datasets, such as ENZYMES.

5 RELATED WORK

This literature review surveys work related to entropy and dimension estimation.

Entropy. A representative way to measure a system's uncertainty is Shannon's entropy [31], which regards entropy as the distribution of basic unit events. Therefore, defining basic unit events is critical for calculating Shannon's entropy. Numerous researchers have tried to find a reasonable basic unit for events on graphs. Dehmer [5] defined the basic unit event in a graph by transforming the node into a positive integer. The structure encoding tree [15, 46] captures the entropy of the graph structure information. Recently, there has also been some work on defining entropy with text data. Term Frequency —Inverse Document Frequency [13] takes the frequency of words in corpus and the inverse document frequency of this word as the basic unit. Su's [35] idea is that the inner product of two-word embeddings could be regarded as the basic unit.

Dimension Estimation. There are some algorithms that estimate the proper dimensionality through a metric —for example, with a loss function that evaluates different numbers of dimensions. Yin and Shen [47] defined the pairwise inner product as a loss function for estimating dimension. Their work focuses on measuring the change in bias and variance resulting from different dimensionalities. The dimension option that provides the most balanced bias and variance is then selected as the best choice. Inspired by this work, Wang [41] presented a score function to gauge performance with different numbers of dimensions, ranging from 2 to a predefined maximum number. Su [35] built an association between entropy and the dimensions of the embeddings based on the semantic distribution hypothesis [29]. Also motivated by the semantic distribution hypothesis, Luo et al. [20] proposed a dimension estimation method for node-level embeddings.

6 CONCLUSION

In this paper, we proposed a novel node and graph representation dimension estimation framework called MGEDE, designed to support GNNs produce embeddings for downstream tasks. Based on the minimum entropy principle, we presented a new method of calculating graph entropy. Composed of attribute entropy and structure entropy, these novel measures can be used to estimate the uncertainty in a graph. Additionally, the framework automatically chooses the appropriate dimensionality for node- and graph-level embeddings, with the choice being the one that minimizes the uncertainty in the graph. Moreover, we also devised a new GNN training architecture that can encode graphs into different dimension spaces. Extensive experiments demonstrate the effectiveness of MGEDE in guiding GNNs on both node- and graph-level tasks.

ACKNOWLEDGMENTS

This work was supported by the Australian Research Council (ARC) Projects Nos. DE200100964, LP210301259, and DP230100899. J. Wu and H. Peng are the corresponding authors.

REFERENCES

[1] Adriano Azevedo-Filho and Ross D Shachter. 1994. Laplace's Method Approximations for Probabilistic Inference in Belief Networks with Continuous Variables. In *Uncertainty Proc. 1994*. 28–36.

[2] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast Unfolding of Communities in Large Networks. *J STAT MECH-THEORY E* (2008), 10008–10021.

[3] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. 2005. Protein Function Prediction via Graph Kernels. *Bioinformatics* (2005), 47–56.

[4] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-gcn: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks. In *Proc. KDD*. 257–266.

[5] Matthias Dehmer. 2008. Information Processing in Complex Networks: Graph Entropy and Information Functionals. *Appl. Math. Comput.* (2008), 82–94.

[6] Paul D Dobson and Andrew J Doig. 2003. Distinguishing Enzyme Structures from Non-enzymes Without Alignments. *J. Mol. Biol.* (2003), 771–783.

[7] Hongchang Gao and Heng Huang. 2018. Deep Attributed Network Embedding.. In *Proc. IJCAL*. 3364–3370.

[8] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Proc. NeurIPS*. 1025–1035.

[9] David W. Henderson and Eduarda Moura. 1995. Experiencing Geometry: On Plane and Sphere. *P.H.*, 1–193.

[10] Pham Thuc Hung and Kenji Yamanishi. 2021. Word2vec Skip-gram Dimensionality Selection via Sequential Normalized Maximum Likelihood. *Entropy* (2021), 997–1010.

[11] Edwin T Jaynes. 1980. The Minimum Entropy Production Principle. *Annu. Rev. Phys. Chem.* (1980), 579–601.

[12] Junteng Jia and Austion R. Benson. 2020. Residual Correlation in Graph Neural Network Regression. In *Proc. KDD*. 588–598.

[13] Karen Sparck Jones. 1972. A Statistical Interpretation of Term Specificity and Its Application in Retrieval. *J Doc* (1972), 11–21.

[14] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *Proc. ICLR*. 1–14.

[15] Angsheng Li and Yicheng Pan. 2016. Structural Information and Dynamical Complexity of Networks. *IEEE TIT* (2016), 3290–3339.

[16] Qian Li, Jianxin Li, Jiawei Sheng, Shiyao Cui, Jia Wu, Yiming Hei, Hao Peng, Shu Guo, Lihong Wang, Amin Beheshti, and Philip S. Yu. 2022. A Survey on Deep Learning Event Extraction: Approaches and Applications. *IEEE TNNLS* (2022), 1–21.

[17] Chuang Liu, Jia Wu, Weiwei Liu, and Wenbin Hu. 2021. Enhancing Graph Neural Networks by A High-quality Aggregation of Beneficial Information. *Neural Netw.* (2021), 20–33.

[18] Fanzhen Liu, Shan Xue, Jia Wu, Chuan Zhou, Wenbin Hu, Cecile Paris, Surya Nepal, Jian Yang, and Philip S Yu. 2020. Deep Learning for Community Detection: Progress, Challenges and Opportunities. In *Proc. IJCAL*. 4981–4987.

[19] Meng Liu, Hongyang Gao, and Shuiwang Ji. 2020. Towards Deeper Graph Neural Networks. In *Proc. KDD*. 338–348.

[20] Gongxu Luo, Jianxin Li, Jianlin Su, Hao Peng, Carl Yang, Lichao Sun, Philip S Yu, and Lifang He. 2021. Graph Entropy Guided Node Embedding Dimension Selection for Graph Neural Networks. *arXiv:2105.03178* (2021).

[21] James MacQueen et al. 1967. Some Methods for Classification and Analysis of Multivariate Observations. In *Proc. BSMSP*. 281–297.

[22] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. 2019. Provably Powerful Graph Networks. In *Proc. NeurIPS*. 1–12.

[23] Avner May, Jian Zhang, Tri Dao, and Christopher Ré. 2019. On the Downstream Performance of Compressed Word Embeddings. In *Proc. NeurIPS*. 1–12.

[24] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. 2015. Image-based Recommendations on Styles and Substitutes. In *Proc. SIGIR*. 43–52.

[25] Zaiqiao Meng, Shangsong Liang, Hongyan Bao, and Xiangliang Zhang. 2019. Co-embedding Attributed Networks. In *Proc. WSDM*. 393–401.

[26] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2019. Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks. In *Proc. AAAI*. 4602–4609.

[27] Galileo Namata, Ben London, Lise Getoor, Bert Huang, and U Edu. 2012. Query-driven Active Surveying for Collective Classification. In *Proc. MLG-KDD*. 1–8.

[28] Robert Peach, Alexis Arnaudon, and Mauricio Barahona. 2022. Relative, Local and Global Dimension in Complex Networks. *Nature Communications* 13, 1 (2022), 1–11.

[29] Magnus Sahlgren. 2008. The Distributional Hypothesis. *Ital. J. Disabil. Stud.* (2008), 33–53.

[30] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective Classification in Network Data. *AI Mag.* (2008), 1–14.

[31] Claude Shannon. 1953. The Lattice Theory of Information. *Transactions of the IRE professional Group on Information Theory* (1953), 105–107.

[32] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of Graph Neural Network Evaluation. In *Proc. RRL-NeurIPS*. 1–11.

[33] Jonathan M Stokes, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, Nina M Donghia, Craig R MacNair, Shawn French, Lindsey A Carfrae, Zohar Bloom-Ackermann, et al. 2020. A Deep Learning Approach to Antibiotic Discovery. *Cell* 180, 4 (2020), 688–702.

[34] Jianlin Su. 2019. Angle Distribution of Two Random Vectors in n-dimensional Space. [EB/OL]. https://kexue.fm/archives/7076.

[35] Jianlin Su. 2020. How to Choose the Dimension of Word Embedding. [EB/OL]. https://kexue.fm/archives/7695.

[36] Xing Su, Shan Xue, Fanzhen Liu, Jia Wu, Jian Yang, Chuan Zhou, Wenbin Hu, Cecile Paris, Surya Nepal, Di Jin, et al. 2022. A Comprehensive Survey on Community Detection with Deep Learning. *IEEE TNNLS* (2022), 1–21.

[37] Zekun Tong, Yuxuan Liang, Henghui Ding, Yongxing Dai, Xinke Li, and Changhu Wang. 2021. Directed Graph Contrastive Learning. In *Proc. NeurIPS*. 19580–19593.

[38] Zekun Tong, Yuxuan Liang, Changsheng Sun, Xinke Li, David Rosenblum, and Andrew Lim. 2020. Digraph Inception Convolutional Networks. In *Proc. NeurIPS*. 17907–17918.

[39] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *Proc. ICLR*. 1–12.

[40] Ruijia Wang, Shuai Mou, Xiao Wang, Wanpeng Xiao, Qi Ju, Chuan Shi, and Xing Xie. 2021. Graph Structure Estimation Neural Networks. In *Proc. WWW*. 342–353.

[41] Yu Wang. 2019. Single Training Dimension Selection for Word Embedding with PCA. In *Proc. EMNLP-IJCNLP*. 3597–3602.

[42] Zhengyang Wang and Shuiwang Ji. 2020. Second-Order Pooling for Graph Neural Networks. *IEEE TPAMI* (2020), 1–12.

[43] Libing Wu, Min Wang, Dan Wu, and Jia Wu. 2021. DynSTGAT: Dynamic Spatial-Temporal Graph Attention Network for Traffic Signal Control. In *Proc. CIKM*. 2150–2159.

[44] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *Proc. ICLR*. 1–17.

[45] Pinar Yanardag and S.V.N. Vishwanathan. 2015. Deep Graph Kernels. In *Proc. KDD*. 1365–1374.

[46] Runze Yang, Hao Peng, and Angsheng Li. 2022. Dynamic Measurement of Structural Entropy for Dynamic Graphs. *arXiv preprint arXiv:2207.12653* (2022).

[47] Zi Yin and Yuanyuan Shen. 2018. On the Dimensionality of Word Embedding. In *Proc. NeurIPS*. 1–12.

[48] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical Graph Representation Learning with Differentiable Pooling. In *Proc. NeurIPS*. 1–11.

[49] Ge Zhang, Zhenyu Yang, Jia Wu, Jian Yang, Xue Shan, Hao Peng, Jianlin Su, Chuan Zhou, Quan Z Sheng, Leman Akoglu, and Charu C Aggarwal. 2022. Dual-discriminative Graph Neural Network for Imbalanced Graph-level Anomaly Detection. In *Proc. NeurIPS*. 1–12.

[50] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An End-to-End Deep Learning Architecture for Graph Classification. In *Proc. AAAI*. 1–8.

[51] Sheng Zhou, Hongjia Xu, Zhuonan Zheng, Jiawei Chen, Zhao Li, Jiajun Bu, Jia Wu, Xin Wang, Wenwu Zhu, and Martin Ester. 2022. A Comprehensive Survey on Deep Clustering: Taxonomy, Challenges, and Future Directions. *CoRR* abs/2206.07579 (2022).