How to Realize Efficient and Scalable Graph Embeddings via an Entropy-Driven Mechanism

Peng Fang, Fang Wang, Member, IEEE, Zhan Shi, Member, IEEE, Hong Jiang, Fellow, IEEE, Dan Feng, Member, IEEE, Xianghao Xu, and Wei Yin

Abstract—Graph embedding is becoming widely adopted as an efficient way to learn graph representations required to solve graph analytics problems. However, most existing graph embedding methods, owing to computation-efficiency challenges for large-scale graphs, generally employ a one-size-fits-all strategy to extract information, resulting in a large amount of redundant or inaccurate representations. In this work, we propose HuGE+, an efficient and scalable graph embedding method enabled by an entropy-driven mechanism. Specifically, HuGE+ leverages hybrid-property heuristic random walk to capture node features, which considers both information content of nodes and the number of common neighbors in each walking step. More importantly, to guarantee information effectiveness of sampling, HuGE+ adopts two heuristic methods to decide the random walk length and the number of walks per node, respectively. Extensive experiments on real-world graphs demonstrate that HuGE+ achieves both efficiency and performance advantages over recent popular graph embedding approaches. For three downstream graph tasks, our approach not only offers >10% average gains, but also exhibits \(23 \times 127\) x speedup over existing sampling-based methods. In addition, HuGE+ significantly reduces memory footprint by an average of 68.9%, facilitating training for billion-node-scale graph embeddings.

Index Terms—Graph embedding, efficiency, scalability, entropy-driven mechanism

1 INTRODUCTION

Graph is an important data structure for representing connected entities and relationships existing in a wide variety of real-world scenarios, such as social networks, biological graphs, traffic networks, semantic graphs, etc. Over the last few years, graph analytics has received significant attention for its ability to extract meaningful insights from large-scale graphs representing the above real-world scenarios. Graph embedding (a.k.a. network embedding), as an effective technique of graph analytics, has become a key means to learn node representations from a graph [1]. Specifically, it aims to embed the nodes of a graph into a low-dimensional vector space, while preserving the inherent structural properties of the graph, and the representation vectors can serve a wide range of downstream graph tasks through machine learning methods, such as link prediction [2], classification [3], and clustering [4].

In the current literature, existing approaches in graph embedding roughly fall into three categories: sampling-based techniques such as Deepwalk [6], Node2vec [7], LINE [8], SDNE [9], Struc2vec [10], VERSE [11], DiaRW [12] and HuGE [21]; matrix factorization-based (MF-based) techniques such as GraRep [13], HOPE [14], NetMF [15], NetSMF [16], STRAP [17], ProNE [18], and NRP [19]; and graph neural networks-based (GNN-based) techniques such as Graph Convolution [20], GraphSAGE [22], Graph Attention [23], GraphGAN [24], and ProGAN [25]. However, most of the existing methods face efficiency and scalability challenges in the embedding process, especially for large-scale graphs that are increasingly common nowadays. For example, sampling-based techniques, such as node2vec, need to sample a large number of node pairs to ensure the quality of embedding, and thus require substantial computational resources, taking months to learn embeddings for a graph with 100 million nodes and 500 million edges by 20 threads on a modern server [18]. For MF-based techniques, the performance of embedding depends heavily on the DRAM size due to expensive matrix factorization operations, some recent works [17], [18], [19] attempt to address this challenge, especially NRP [19] efficiently handle a billion-edge Twitter graph on 100GB level RAM. NRP also reveals that the random walk based methods are restricted by the sheering number of possible walks while increasing the walk length, so it is possible to implement a heuristic walk to improve sampling efficiency? For GNN-based techniques, training a neural network also incurs very high computational overhead and inevitably limits their scalability [1], [19]. Therefore, it is critically important to generate high-efficiency graph embeddings to enable downstream tasks to be executable on large-scale graphs.
Furthermore, current graph embedding techniques adopt a one-size-fits-all strategy for all nodes and fail to meet the unique requirements of different individual real-world graphs [11]. For the sampling-based approach, which generally samples node pairs from an input graph, defining the proximity of pairwise nodes is the key to achieving effectiveness for various downstream tasks in diverse graphs. In addition, the random walk length \( L \) and the number of walks per node \( r \) in existing techniques heavily rely upon an empirical value set (usually, \( L \approx 80 \) and \( r = 10 \)). These static configurations for the sampling procedure may introduce an excessive amount of low-quality information, limiting the efficiency and scalability on large-scale graphs.

To better understand the impact of \( L \) and \( r \) on learning node representations, we quantitatively examine the relationship between information entropy and the walk length, in Fig. 1, and between relative entropy and number of walks per node, in Fig. 2. The concept of information entropy was introduced by Claude Shannon [27] (a.k.a. Shannon entropy) as a measure of how much information is contained in a given source [28]. As mentioned in previous approaches, information entropy can be used to measure global or local information in random walk process on graphs [29], [30] and quantify the effectiveness of text data [31], [32]. Consider a path \( W^L_u \) generated by the random walk with length \( L \) from source node \( u \), suppose that the path \( W^L_u \) passes through the nodes \( v_1, v_2, v_3, \ldots, v_r \), then the probability of \( v_i \) occurring in path \( W^L_u \) is \( p(v_i) = \frac{n(v_i)}{n(W^L_u)} \), where \( n(v_i) \) represents the degree of nodes in graph and the counts of node occurrence in random walks.

The information entropy of all paths with length \( L \) generated by the walker is \( H(W^L_u) = -\sum p(v_i) \log p(v_i) \), indicating that the more evenly distributed the nodes in a path, the larger the \( H \) value will be. Fig. 1 shows how information entropy changes with an increasing walk length \( L \) on three real graphs: Karate [34], email-Eu-core [35], and Wiki-Vote [36]. For clarity of display, each curve plots the average information entropy of paths starting from all nodes in the corresponding graph and sampling procedure. Since each path of short random walks corresponds to a sentence from corpus [6], we can find that the commonly used \( L \) value of 80 in existing models cannot support a concise and comprehensive representation because of the diminishing returns on \( H \), considering the linearly increased size of walks, this will inevitably introduce a great amount of redundant information. From a macro perspective, the corpus is generated by multiple rounds \( r \) of walk paths for each node, and the study in [6] reveals that the distribution of frequency with which nodes appear in corpus \( q(v_i) \) is similar to the node degree distribution of a graph \( p(v_i) \) because they both follow a power-law distribution [26]. Inspired by this observation, we first leverage kernel density distribution [33] to estimate the two distributions and then use the Kullback-Leibler divergence (a.k.a. relative entropy) [37] to quantify the discrepancy between \( q(v_i) \) and \( p(v_i) \), denoted as \( KL = \sum p(v_i) \log \frac{p(v_i)}{q(v_i)} \). Visually indicated in Fig. 2, as \( r \) increases, the first peak of the blue shade (i.e., the tallest on the left in \( q(v_i) \)) becomes more leptokurtic, and its density distribution more similar to that of the red shade \( p(v_i) \). This means that the nodes appearing less frequently in the walk are more and more concentrated, similar to the low-degree nodes are concentrated in the graph, which is consistent with the similarity between the two distributions observed in [6]. It also can be found that the \( KL \) value labeled in Fig. 2 has a slight variation after \( r = 7 \), which is less than the empirical value for \( r = 10 \). Note that the corpus is obtained from the sampling procedure and then the representation vectors are generated by word embedding methods. The low-quality corpus will adversely affect the effectiveness of embedding training and ultimately interfere with the accuracy of downstream tasks [40], [41]. Consequently, the flexibility of strategy and effectiveness of walks should be fully considered in the generated graph embeddings.

To address the above challenges, we propose HuGE+, an entropy-driven approach to efficient and scalable graph embeddings. Specifically, HuGE+ is designed on top of the sampling-based model due to its capability of automatically capturing useful representations from complex graph structures [1]. Different from HuGE [21], which only considers the structural and potential properties to differentiates the candidates in the sampling strategy, ignoring the effect of information content of the added node to the generated path, HuGE+ leverages hybrid-property heuristic random walk, which taking into consideration both common neighbors and the information content of the nodes to exploit the proximity of the pairwise nodes, in other word, if a candidate node has more common neighbors with the source
node and takes along more effective information to the generated path, then the walker moves to this node with a higher probability. In this fashion, it not only identifies nodes with similar roles, but also ensures the exploration of richer information in the walking. Moreover, from the perspective of flexible and effective sampling, we are inspired by the observation in Figs. 1 and 2 to present heuristic methods based on information theory [39]. Simply put, the solutions adopt the information entropy and relative entropy to measure the information effectiveness of the generated walk path and corpus to determine the random walk length and the number of walks per node, respectively, improving the computing and memory efficiency without sacrificing correctness.

In summary, our main contributions are as follows:

- We investigate the pressing problems of compute efficiency for large-scale graph feature learning, and quantitatively analyze why the flexibility and the effectiveness of sampling are crucial to the generated graph embeddings.
- We propose HuGE+, an entropy-driven model that takes into account both the number of common neighbors and the information content of node to measure the proximity of pairwise nodes, and adopts heuristic methods to decide the random walk length and number of walks per node, instead of the hyperparameter optimization in existing techniques. HuGE+ sheds new light on improving performance for generating embeddings on large-scale graphs.
- We conduct extensive experiments on real-world and synthetic graphs to confirm the advantages of our approach. The evaluation results show that HuGE+ achieves much better efficiency and effectiveness of embedding learning over recent popular graph embedding techniques, offering $>10\%$ average gains in downstream tasks. In particular, our solution exhibits $23 \times -127 \times$ speedup and reduces memory footprint by an average of 68.9% compared with sampling-based models.

## 2 Design and Analysis of HuGE+

In this section, we present HuGE+, an efficient and scalable two-phase model with a sampler phase and a learner phase. The overall workflow of HuGE+ is shown in Fig. 3. The sampling strategy in HuGE+ takes into account both the information content of nodes and the number of common neighbors to measure the proximity of pairwise nodes, and adopts heuristic methods to adjust the random walk length and the number of walks per node. The representation vectors are generated by the Skip-Gram model [47].

### 2.1 Sampling Strategy

#### 2.1.1 Hybrid-Property Heuristic Random Walks

Sampling-based graph embedding techniques extract node features through random walks, which have been used to approximately measure many properties in the graph. As popularly used in graph statistics, common neighbors can represent potential information between nodes, such as node similarity [8], [38]. Besides, most real-world graphs are reported to be scale-free because their degree distributions follow a power-law, at least asymptotically [26]. High-degree nodes with a number of edges that greatly exceeds the average usually play important roles in graph, for instance, information spreading in social networks [54] and epidemics control in disease networks [55]. For random walks, high-degree nodes tend to be revisited more and walks starting from them are likely to obtain richer information by traveling around the local neighbors [12], [56]. Deepwalk uses a uniform probability to choose the next-hop node, which will not adequately preserve the original graph properties, especially when the underlying information around nodes differs significantly due to node degree skewness. Although Node2vec introduces two hyperparameters to consider both homophily and structural equivalence for nodes in a biased random walk, it requires relatively high overheads for laborious parameter tuning. DiaRW presents high-degree biased backtracking in sampling strategy, yet the node representation only adapts to the scale-free feature of real-world graphs, ignoring the implicit properties of nodes, such as node similarity. Recently proposed HuGE [21] leverages a heuristic random walk mechanism to optimize the one-size-fit-all static configuration, it jointly considers the common neighbors and the degree of nodes to measure the proximity of pairwise nodes for the next-hop node selection. However, since the skewness and clustering in the real-world graphs, this strategy may easily induce the walker falls into local loops or be likely to walk back to the previously visited hub-nodes. Precisely, if the selected node has more common neighbors with previously visited nodes, then the walker may have a higher probability of walking back to those visited nodes via the common neighbors. Meanwhile, if the selected node is rejected under the walking-backtracking strategy used in HuGE, the walker should backtrack to the current node and re-select a neighbor to walk. Although these operations facilitate to identify the similarity and structural properties of a graph, from the aspect of content in the generated walking path, it is difficult to ensure the effectiveness and diversity of sampling.
with too much repeated or backtracking information. Therefore, for a concise and comprehensive representation, sampling procedure not only needs to capture a exact information, but also needs to consider the effect of the collected information to the generated content.

Based on the above considerations, we propose a hybrid-property heuristic random walk (HRW) for next-hop node selection. Different from the the pairwise node proximity measurement in HuGE, HRW differentiates the candidates by taking into account both the number of common neighbors and the information content of candidate nodes. More formally, common neighbors can be represented the potential proximity for pairwise nodes, and the information content definition not only includes the amount of information contained in the node itself, but also considers the effect of the candidate node to the content of generated walking path to reduce the redundancy information. Specifically, Fig. 4 shows an example of random walk, suppose that the walker is currently at \( v_0 \) and the task here is to select the next-hop node from among \( N(v_0) = \{v_1, v_2, v_3, v_4, v_5\} \), if a candidate node \( v \) \((v \in N(v_0))\) has more common neighbors with \( v_0 \) and takes along more information for the walking path, then the walker moves to \( v \) with a higher probability. In the example graph, the number of common neighbors between \( v_0 \) and \( \{v_1, v_2, v_3, v_4, v_5\} \) is \( 1, 2, 1, 0, 0 \), and the degrees of \( \{v_1, v_2, v_3, v_4, v_5\} \) are respectively \( 7, 4, 4, 4, 1 \). Intuitively, since \( v_2 \) has the most common neighbors with \( v_0 \), it has the highest similarity to \( v_0 \) among nodes in the candidate set. Nevertheless, the walker may have a high chance of walking back to previously visited nodes through common neighbors. On the other hand, since \( v_3 \) and \( v_5 \) have the same number of common neighbors with \( v_0 \), the walker will more likely choose \( v_1 \), leading to a higher probability of walking to unvisited nodes, however, when selecting the next-hop node from \( v_1 \), under the backtracking mechanism, the walker may have a high probability to backtrack to \( v_2 \), it is likely to fall into local loop like HuGE. Although it can capture the local structural features, a large amount of repetitive information produced in the content of the generated walking path due to these loops. To alleviate this situation, it is necessary to dynamically measure the effect of added node to the content of the walking path, if a added node has a lot of repetitions in the path, the probability of transferring to this node should be reduced, while the opposite case should be given a higher transition probability, and thus ensuring the effectiveness and diversity of sampling. Since the transition probability from \( v_0 \) to \( v_4 \) and \( v_5 \) will be the lowest due to the minimal similarity with the current node \( v_0 \), the probability of \( v_0 \) and \( v_4 \) and \( v_5 \) in the graph will likely be minimal. Therefore, by jointly considering common neighbors and the information content of nodes, HRW is arguably capable of both identifying similar properties and ensuring the exploration of richer information in the random walk.

To implement HRW, the key issue is to formulate the selection of the next-hop node with a mathematical model. Table 1 lists key notations used throughout the paper. Let \( G = (V, E) \), \( u, v \in V \), we first leverage the number of common neighbors \( Cm(u, v) \) to define the node similarity between \( u \) and \( v \) by the number of distinctive neighbors as

\[
\text{Sim}(u, v) = \frac{1}{\text{deg}(u) - Cm(u, v)}, \text{if}(u, v) \in E,
\]

where \( \text{deg}(u) \) is the degree of \( u \). From the above equation, it is obvious that \( \text{Sim}(u, v) \) grows with \( Cm(u, v) \) since \( \text{deg}(u) \) is fixed. Instead of the single property biased random walk, HRW also considers weights in node transition probabilities based on the information content of nodes.

Claude Shannon defined information entropy as a theoretical model of communication, i.e., the transmission of information of various kinds [27], which has been wildly adopted in many areas, such as neural networks [57], feature selection [58], and best basis selection [59]. For a given random event \( X \), with possible states \( (1, 2, \ldots, n) \), then \( p(X_i) \) can be seen as a probability function of a random variable, and the amount of information for event \( X \) is defined as

\[
I(X_i) = -\log(p(X_i)),
\]

and thus the information entropy \( H(X) \) is given by

\[
H(X) = -\sum_{i=1}^{n} p(X_i) \log(p(X_i)).
\]
By convention, \( p_{X_i} \log p_{X_i} = 0 \) if \( p_{X_i} = 0 \). The logarithm in Equation 3 provides a convenient unit for quantifying the amount of information, i.e., the unit is called a bit when the base of logarithm is 2. \( H(X) \) represents the average number of bits required to represent each item in \( X \).

For random walk, high-degree nodes tend to be revisited more and walk starting from them are likely to obtain richer information, then based on Equation 2 the amount of information of candidate node \( v \) can be defined as

\[
I(v) = -\log \left( \frac{1}{\deg(v)} + \varepsilon \right), \quad v \in N(u),
\]

where \( \varepsilon \) is the initial parameter to retain the information of node with \( \deg(v) = 1 \) in the logarithm. As analysis in Fig. 1, the information entropy of walk path with length \( L \) generated by the walker is \( H_{W^L} = -\sum p_i \log p_i \), where the \( p_i \) is the probability of \( i \) occurring in the path \( W^L \) from the source node \( u \). To ensure a concise and comprehensive node feature extraction in the sampling procedure, the variation of \( H_{W^L} \) (\( \Delta H_{W^L} \)) for the walk path after adding the candidate node should be considered in the weighted of transition probability, \( \Delta H_{W_{L+1}^L} \) is defined as

\[
\Delta H_{W_{L+1}^L} = H_{W_{L+1}^L} - H_{W_L}, \quad L = 1, 2, 3, \ldots, k.
\]

Based on Equations (4) and (5), suppose the source node is \( u \), the weight function can be constructed by \( I(v) \cdot \Delta H_{W^L} \) to define the information content for the candidate node \( v \), this function brings two benefits: (1) exerting the influence of high degree nodes to obtain richer information by traveling around the local neighbors; (2) discriminating the effective information for the sampled path to reduce redundancy. Accordingly, the weight of transition probability from \( u \) to \( v \) is

\[
\alpha_{(u,v)} = \frac{1}{\deg(u) - Cm(u,v)} \cdot \left\{ -\log \left( \frac{1}{\deg(v)} + \varepsilon \right) \frac{H_{W_{L+1}^L}}{H_{W_L}} \right\}.
\]

Take Fig. 4 as an example to illustrate the walking process in one step. Suppose that the walker is currently at \( v_{0} \), \( N(v_{0}) = \{v_1, v_2, v_3, v_4, v_5\} \), the weights of transition probabilities are \( \alpha_{(0,1)} = 0.702 \), \( \alpha_{(0,2)} = 0.667 \), \( \alpha_{(0,3)} = 0.5 \), \( \alpha_{(0,4)} = 0.4 \), \( \alpha_{(0,5)} = 0.028 \), suggesting that \( v_1 \) has a higher probability than other candidate nodes as the next-hop node, which is consistent with the intuition. Unlike HRW, Deepwalk uses equal probability 0.2 for all candidate nodes in \( N(v) \), which obviously fails to differentiate the implicit properties for nodes and ignores the importance of structural information.

To limit the computation overhead, HRW leverages a commonly used strategy, walking-backtracking [12], [56], to determine the next-hop in each walking step. Specifically, at a source node \( u \) in each step, HRW randomly chooses \( v \) from \( N(u) \) as a candidate node, the acceptance probability for \( v \) as the next-hop node is \( p_{(u,v)} \), and if \( v \) is rejected, which will happen with probability \( 1 - p_{(u,v)} \), the walker will backtrack to \( u \) and repeat a random selection again from \( N(u) \). In this way, we only access the accepted node and avoid traversing all nodes in \( N(u) \) in each step. Formally, the transition probability from \( u \) to \( v \) can be written as

\[
P_{(u,v)} = \begin{cases} 
Z(a_{(u,v)}), & \text{if } (u, v) \in E \\
0, & \text{other}
\end{cases}
\]

Here we do a normalization on \( a_{(u,v)} \) by \( Z = \frac{\mathbf{e}^{\mathbf{u} - \mathbf{v}}}{\mathbf{e}^{\mathbf{u} - \mathbf{v}} + 1} \), which is widely applied in machine learning and satisfies the walking-backtracking strategy.

### 2.1.2 Heuristic Walk Length

In the current literature, most sampling-based graph embedding techniques adopt a one-size-fits-all strategy for all nodes, setting as a fixed walk length that relies on an empirical value set (usually, \( L = 80 \)). As mentioned above, each path of short random walks corresponds to a sentence from the corpus in the Skip-Gram model. The representation of sentences needs to be concise and have good coverage. If \( L \) is too large, redundancy will become unavoidable; otherwise, too small an \( L \) will make it difficult to guarantee the coverage of graph properties. However, most existing techniques do not take this into consideration. Moreover, an over-sized \( L \) would directly increase the sampling overhead and the storage and computation consumption in the learning phase, thereby limiting the scalability. Fortunately, information entropy can be used as a measure of how much information is contained in a given source [28], and thus we propose a heuristic walk length strategy (HWL) to measure the effectiveness of information during walking based on information entropy (\( H \)).

The main idea of HWL is to observe the variation of \( H \) for short random walks as a node is selected to the path, and if it becomes stable, the random walk is simply stopped. In other words, as shown in Fig. 1, a stable \( H \) means that the newly added node has little contribution to representation. HWL characterizes the correlation between the variations of \( H \) and \( L \) by linear regression and calculates the coefficient of determination (\( R^2 \)) to determine the termination of a random walk process in the heuristic method. Mathematically, for any \( u, v \in G \), the random walk process of the source node \( u \) is defined as

\[
W_{L}^{u} = \{v_{1}^{u}, v_{2}^{u}, v_{3}^{u}, \ldots, v_{k}^{u}\}, \quad L = 1, 2, 3, \ldots, k,
\]

where \( v_{k}^{u} \) denotes \( k \)th node of the path, \( L \) is the walk length. With the \( L \) variation, the probability of \( v \), occurrence in the path is \( p_{v} = \frac{n(v)}{\sum_{i=1}^{k} n(v)} \). Accordingly, information entropy for the random walk is given as follows:

\[
H_{W_{L}^{u}} = -\sum_{i=1}^{k} \frac{n(v)}{\sum_{i=1}^{k} n(v)} \log \frac{n(v)}{\sum_{i=1}^{k} n(v)}.
\]

Based on the above equation, the Pearson correlation coefficient for \( H_{W_{L}^{u}} \) and \( L \) is computed by

\[
r_{(H,L)} = \frac{\sum_{i=1}^{n}(H_{W_{L}^{u}} - \overline{H}_{W_{L}^{u}})(L_{i} - \overline{L})}{\sqrt{\sum_{i=1}^{n}(H_{W_{L}^{u}} - \overline{H}_{W_{L}^{u}})^{2}} \sqrt{\sum_{i=1}^{n}(L_{i} - \overline{L})^{2}}}, \quad n = 1, 2, \ldots, L,
\]

where \( \overline{H}_{W_{L}^{u}} \) and \( \overline{L} \) are the mean of the series \( H_{W_{L}^{u}} \) and \( L_{i} \) (\( 1 \leq i \leq n \), respectively). In the case of a linear model for \( H \) and \( L \), \( R_{(H,L)}^{2} \) is simply the square of \( r_{(H,L)} \). The closer \( R_{(H,L)}^{2} \)
is to 1, the better linear relation between $H$ and $L$. Fig. 5a shows an example of the linear relation between $H$ and $L$ on Wiki-Vote graph, the dotted lines indicate linear regression between the $H$ and $L$ of each walking step, it can be found that the linear relation between the two variables gradually decreases with the newly-selected node adding to the walking path, while the commonly used $L = 80$ in existing models obviously cannot maintain a good linear relationship due to the diminishing returns on $H$. Here we set $R_{H,L}^2 \geq \mu$ as the judgment condition to determine whether the walk stops. To guarantee a linear relationship for two datasets in a linear model, usually, we let $\mu \geq 0.99$.

2.1.3 Heuristic Number of Walks Per Node

Sampling-based techniques usually perform multiple walks for each node to guarantee the quality of training in the Skip-Gram model. The number of walks per node determines the size of corpus generated by walk paths. As with the fixed walk length that depends on an empirical value set, the number of walks per node (usually set $r = 10$) is an urgent problem that needs to be solved. The conciseness of corpus can not only ensure the effectiveness of information but also reduce the overhead of the sampling phase and improve efficiency in the learning phase. As shown in Fig. 2, with $r$ increasing, the probability distribution of node occurrence in the generated corpus will be similar to the degree distribution of the graph and gradually stabilize. Based on this observation, we try to quantify the quality of corpus through the difference between the node occurrence distribution in corpus and the degree distribution in graph. One of the most popular means of measuring the discrepancy between two distributions is to use relative entropy [37], which has been fruitfully used for machine learning metrics [60], quantum computation [61], image thresholding [62] and hot spot identification [63]. Accordingly, we present the heuristic number of walks per node (HWN) strategy, focusing on heuristically exploring the number of walks based on relative entropy ($D(p||q)$).

Relative entropy (a.k.a. Kullback-Leibler divergence) is the general case of entropy that measures the distance between two distributions or a measure of the inefficiency of an assumption that the distribution is $q(x)$ when the true distribution is $p(x)$ [64]. In other words, the divergence of $p(x)$ with respect to $q(x)$, $D(p||q)$, is defined as relative entropy [65]

$$D(p||q) = \sum_x p(x) \log \frac{p(x)}{q(x)}. \quad (11)$$

$D(p||q)$ is always nonnegative, a convex function of $(p,q)$, and zero if and only if the two distributions are the same, $p = q$. Moreover, it is asymmetric, i.e., $D(p||q) \neq D(q||p)$.

For a given graph $G$, where any $v_i \in G$, the degree of a node is denoted as $\text{deg}(v_i), i = 1, 2, 3, \ldots, n$, and $n$ is the number of nodes in $G$, the degree distribution is given by

$$p(v_i) = \frac{\text{deg}(v_i)}{\sum_i \text{deg}(v_i)}.$$

(12)

The occurrence counts of $v_i$ in the generated corpus is $n(v_i), i = 1, 2, 3, \ldots, n$, and the probability for $n(v_i)$ is defined as

$$q(v_i) = \frac{n(v_i)}{\sum_i n(v_i)}$$

(13)

Then the relative entropy for $p(v_i)$ and $q(v_i)$ is denoted as

$$D(q(v_i)||p(v_i)) = \sum_i \frac{\text{deg}(v_i)}{\sum_i \text{deg}(v_i)} \log \frac{\text{deg}(v_i)}{n(v_i) \sum_i \text{deg}(v_i)}.$$  

(14)

With $r$ increasing, we can calculate the difference of $D(q||p)$ as

$$\Delta D_r(q||p) = |D_r(q(v_i)||p(v_i)) - D_{r-1}(q(v_i)||p(v_i))|, r > 1.$$  

(15)

As shown in Fig. 5b, $\Delta D_r(q||p)$ will gradually converge with $r$ variation on Wiki-Vote graph, and convergence is found to be achieved at less than the commonly used empirical value for $r = 10$, further confirming the observation in Fig. 2. HWN leverages $\Delta D_r(q||p) \leq \delta$ as the termination condition for the sampling phase, utilizing a heuristic method to explore the number of walks per node. Taking the link prediction on Wiki-Vote graph as an example, where $\delta = 0.001$, the sampling phase needs 7 iterations per node, less than the traditional scheme $r = 10$, it does not reduce the accuracy of tasks. In other cases, HWN will introduce $r$ that is greater than 10 for some graphs, such as CA-AstroPh. Interestingly, since HWN aims to optimize the effectiveness of corpus, it also benefits downstream tasks, which will be discussed in the evaluation section.

2.2 Feature Learning Strategy

Feature learning in graphs can be treated as a maximum likelihood optimization problem [6], [7]. In general, feature learning can be modeled as a mapping function $\varphi : V \rightarrow \mathbb{R}^d$ from nodes to feature representations, where $d$ is the number of dimensions for feature representations and $\varphi(u)$ is the embedding vector of node $u$. For each source node $u \in V$, we define $N_S(u)$ as the neighbors of node $u$ generated by a sampling strategy $S$. Since our scheme captures node representations based on the Skip-Gram model [47], the optimization for the objective function is given as

$$\max_{\varphi} \sum_{u \in V} \log Pr(N_S(u) | \varphi(u)).$$  

(16)

Skip-Gram is a natural language processing model that can maximize the co-occurrence probability between words within a window $w$ in a sentence. Assuming that the predicting nodes in a context are independent of one another, the conditional probability in Equation (16) can be approximated by

Fig. 5. For sampling on Wiki-Vote graph, (a) shows a schematic diagram of the linear relation between $H$ and $L$, the dotted lines indicate linear regression of each step. (b) depicts the difference in $D(q||p)$ as $r$ increases.

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on March 01,2023 at 21:15:25 UTC from IEEE Xplore. Restrictions apply.
\[ \Pr(N_S(u) \mid \varphi(u)) = \prod_{n_i \in N_S(u)} \Pr(n_i \mid \varphi(u)). \]

(17)

Algorithm 2 provides pseudocode for the generation of representation vector \( \varphi(u) \) of each node \( u \) from walk sequence that appears within window \( w \) based on the Skip-Gram model. We leverage stochastic gradient descent (SGD) to iteratively update the representation of \( u \) to maximize the probability of its neighbors (line 3,4).

Algorithm 1. The HRW Algorithm

**Input:** Graph \( G(V,E) \), Current node \( u \), HWL parameter \( \mu \), Minimum walk length \( L_{min} \)

**Output:** Node walk path \( walk \)

1: Initialize walk to [u], \( R^2_{(i,L)} = 1 \)
2: while \( R^2_{(i,L)} \geq \mu \) do
3: \( curr = \textrm{walk[-1]} \)
4: Select \( v \) uniformly at random from \( curr \)'s neighbors
5: Append \( v \) to \( walk \)
6: Generate a random number \( p \in [0,1] \)
7: if \( p > Z_{\text{softmax}} \left( \frac{1}{\max(n_v \mid \varphi(v))} \right) \) then
8: Append \( curr \) to \( walk \)
9: end if
10: \( L = \text{length of walk} \)
11: if \( L > L_{min} \) then
12: \( H_{W \cdot L} = \text{Information entropy of the current walk} \)
13: \( R^2_{(i,L)} = \text{Squared of } r_{(i,u \cdot L)} \)
14: end if
15: end while
16: return \( walk \)

Since neighboring nodes are symmetrical to each other in the feature space, we use the softmax unit to model the conditional likelihood for each source-neighbor node pair as

\[ \Pr(n_i \mid \varphi(u)) = \frac{\exp(\varphi(n_i) \cdot \varphi(u))}{\sum_{v \in V} \exp(\varphi(v) \cdot \varphi(u))} \]

The function \( \sum_{v \in V} \exp(\varphi(v) \cdot \varphi(u)) \) is expensive to compute for large graphs. To reduce the compute overhead, we speed up the training time by approximating it via negative sampling [33]. The time complexity of Skip-Gram with negative sampling is \( O(C \cdot \cdot (K + 1) \cdot o) \), where \( K \) is the number of negative nodes that are sampled, \( C \) means the corpus size, \( w \) denotes the window size and the unit operation of predicting and updating one word’s embedding is \( o \), and the space complexity is \( O(2 |V| d) \), where \( d \) is representation dimensions.

### 2.3 Implementation of HuGE+

The pseudocode of HuGE+ is given in Algorithm 3. HuGE+ consists of two main parts: a sampler and a learner, shown as Algorithms 1 and 2, respectively. In random walk implementations, there is an implicit bias due to the choice of the starting node \( u \), which is a common practice in sampling-based techniques [6], [7], [8], [9], [10], [11], [12]. Since we learn representations for all nodes, we handle this bias by simulating heuristic random walks starting from every node. The sampling procedure is realized in the HRW algorithm with the two heuristic methods HWL and HWN to adjust the walk length and number of walk per node, respectively. The HWL strategy for random walk length is implemented in lines 10 to 14 of Algorithm 1, and the HWN strategy for the number of walks per node is exhibited in lines 8 to 10 of Algorithm 3. To ensure that the words in corpus can be gained from the sliding windows \( w \) in the Skip-Gram model, we set a minimum walk length \( L_{min} \). Each phase in HuGE+ can be parallelized and executed asynchronously. We will show the performance of the parallel version HuGE+ P in the evaluation elaborated next. The time complexity of HuGE+ sampling procedure is denoted as \( O(L' \cdot r' \cdot |V|) \), where the actual walk length \( L' \) and rounds of walks per node \( r' \) are inversely proportional to the threshold value \( \mu \) and \( \delta \), respectively.

Algorithm 2. The HuGE+ Algorithm

**Input:** Graph \( G(V,E) \), Current node \( u \), HWL parameter \( \mu \), Minimum walk length \( L_{min} \), HWN parameter \( \delta \), Dimensions \( d \), Window size \( w \)

**Output:** node representations \( \varphi \in \mathbb{R}^{|V| \times d} \)

1: Initialize \( \text{walks to Empty} \), \( D_i(q|p) = 1 \)
2: for each node \( v \) in \( G \) do
3: \( p(v_i) = \text{Degree distribution of } v_i \) in \( G \)
4: \( \Delta D_i(q|p) > \delta \) do
5: for all nodes \( u \in V \) do
6: \( \text{walk } = \text{HRW}(G, u, \mu, L_{min}) \)
7: \( \text{Append walk to walks} \)
8: end for
9: \( q(v_i) = \text{Occurrence probability of } v_i \) in \( \text{walks} \)
10: \( D_i(q|p) = \text{Relative entropy for } q(v_i) \) and \( p(v_i) \)
11: \( \Delta D_i(q|p) = |D_i(q|p) - D_{i-1}(q|p)| \)
12: end while
13: Skip-Gram(\( \varphi \), \( \text{Walks} \), \( w \))
14: return \( \varphi \)

### 3 Evaluation

In this section, we will evaluate the efficacy of our proposed HuGE+ approach through extensive experiments driven by real-world and synthetic graph datasets. We compare HuGE+ against ten state-of-the-art graph embedding techniques for three downstream tasks, link prediction [2], multi-label classification [3] and graph clustering [4] that are commonly used for graph embedding evaluation [6], [7], [8], [10], [11], [12], [17], [18], [19]. We also assess the scalability and parameter sensitivity of HuGE+, along with the performance enhancement of sampling-based benchmarks attributed to the heuristic methods in HuGE+.

#### 3.1 Experimental Setup

**Experiment Environment.** We conduct evaluations on a 2.10 GHz Intel Xeon E7-4830 server equipped with 1T RAM and 4.5 T disk, running Ubuntu 18.04. HuGE+ is implemented by Python 2.7.6.
Eight widely-used real-world graph datasets are employed in our experiments. Table 2 lists the key properties of these datasets. According to the requirements of evaluation tasks, four graphs are selected for the multi-label classification and graph clustering task: PPI [68], Wiki [69], Flickr [71] and Youtube [71], and five graphs are chosen for the link prediction task: Wiki-Vote [36], CA-AstroPh [72], Youtube [71], LiveJournal [70], and Twitter [70]. Further more, we also generate a set of synthetic graphs [74] for evaluating the scalability of HuGE+.

Baselines. We compare HuGE+ against ten state-of-the-art graph embedding methods, including five sampling-based techniques (Deepwalk [6], Node2vec [7], LINE [8], VERSE [11] DiaRW [12] and HuGE [21]), two matrix factorization-based techniques (HOPE [14] and ProNE [18]), and two recent graph neural networks-based techniques (GraphGAN [24] and GraphSAGE [22]). In all the experiments, we set the dimensionality of embedding $d$ to 128, and choose the best embedding by parameter tuning for each method. For HuGE+, we set the HWL parameter $\mu$ to 0.995, HWN parameter $\delta$ to 0.001, and window size $w$ to 10. To ensure that the words in corpus can be gained from the sliding windows in the Skip-gram model, we set a minimum walk length $L_{min}$ to 10, to be the same as $w$. To cope with the multiple threads/processes in competitors, we also implement a 10-process HuGE+ and evaluate its performance based on the above default parameters.

### 3.2 Link Prediction

To perform the link prediction task on a given graph $G$, we first randomly remove 50% of its edges as a positive edge set, and the rest as a training set. We also provide a negative edge set where the randomly selected edges are not in $G$. It should be noted that the size of the negative set is the same as that of the positive set, and thus the two sets form the task testing set. For a pair of nodes $(u, v)$, $\phi(u)$ and $\phi(v)$ are the node representation vectors learned by embedding methods. Here the similarity score for $u$ and $v$ is measured in terms of the inner product $\phi(u) \cdot \phi(v)$, along with the

| Graph   | $|V|$ | $|E|$ | Type         | $|\text{Label}|$ | Density |
|---------|------|------|--------------|----------------|---------|
| PPI     | 3.89K | 76.6K | undirected   | 50              | $5.0 \times 10^{-4}$ |
| Wiki    | 4.78K | 185K  | directed     | 40              | $1.6 \times 10^{-2}$ |
| Flickr  | 80.5K | 5.90M | undirected   | 195             | $1.8 \times 10^{-3}$ |
| Youtube | 1.14M | 2.99M | undirected   | 47              | $4.6 \times 10^{-6}$ |
| Wiki-Vote | 7.12K | 104K  | directed     | –               | $2.9 \times 10^{-3}$ |
| CA-AstroPh | 18.72K | 198K | undirected   | –               | $1.2 \times 10^{-3}$ |
| LiveJournal | 2.24M | 14.6M | directed     | –               | $5.1 \times 10^{-6}$ |
| Twitter | 11.3M | 85.3M | directed     | –               | $9.9 \times 10^{-7}$ |

### 3.3 Multi-Label Classification

Multi label classification task aims to predict one or more labels for each graph node and has been widely applied in

---

1. https://github.com/phanein/deepwalk
2. https://github.com/aditya-grover/node2vec
5. https://github.com/RcmFang/HuGE
7. https://github.com/THUDM/ProNE
8. https://github.com/hwwang55/GraphGAN

---

**TABLE 2**

The Real-World Graph Datasets Used in Experiments ($K = 10^3$, $M = 10^6$)

**TABLE 3**

AUC of HuGE+ and Baselines for Link Prediction on Different Graphs, Where a “-” Signifies the Failure of the Corresponding Method Due to Compute-Resource or Running-Time (>5 Days) Constrains

<table>
<thead>
<tr>
<th>Method</th>
<th>Wiki-Vote</th>
<th>CA-AstroPh</th>
<th>Youtube</th>
<th>LiveJournal</th>
<th>Twitter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deepwalk</td>
<td>0.801</td>
<td>0.887</td>
<td>0.707</td>
<td>0.867</td>
<td>–</td>
</tr>
<tr>
<td>Node2vec</td>
<td>0.794</td>
<td>0.876</td>
<td>0.728</td>
<td>0.870</td>
<td>–</td>
</tr>
<tr>
<td>LINE</td>
<td>0.788</td>
<td>0.872</td>
<td>0.805</td>
<td>0.861</td>
<td>–</td>
</tr>
<tr>
<td>VERSE</td>
<td>0.854</td>
<td>0.924</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>ProNE</td>
<td>0.867</td>
<td>0.937</td>
<td>0.827</td>
<td>0.859</td>
<td>–</td>
</tr>
<tr>
<td>GraphGAN</td>
<td>0.853</td>
<td>0.949</td>
<td>0.766</td>
<td>0.881</td>
<td>0.881</td>
</tr>
<tr>
<td>GraphSAGE</td>
<td>0.892</td>
<td>0.913</td>
<td>0.772</td>
<td>0.891</td>
<td>–</td>
</tr>
<tr>
<td>HuGE</td>
<td>0.938</td>
<td>0.951</td>
<td>0.812</td>
<td>0.901</td>
<td>0.897</td>
</tr>
<tr>
<td>HuGE+</td>
<td>0.943</td>
<td>0.952</td>
<td>0.819</td>
<td>0.912</td>
<td>0.899</td>
</tr>
</tbody>
</table>
modern applications ranging from text categorization to bioinformatics. To perform this task, we use embedding vector and a one-vs-rest logistic regression classifier with L2 regularization [73] (using the LIBLINEAR library), and evaluate accuracy by micro-averaged F1 (Macro – F1) and micro-averaged F1 (Macro – F1) [9], where Macro – F1 considers equal weights to each instance and Macro – F1 gives equal weights to each category.

To train a classifier, nodes are randomly split into a training set and a test set, respectively. In the experiments, for each input graph \( G \), we select 10% to 90% training ratio on PPI, Wiki and Flickr, and 1% to 9% training ratio on Youtube, and the remaining nodes for testing. We report the averaged Macro – F1 and Micro – F1 scores from 100 trials. Figures 6 shows the Macro – F1 and Micro – F1 scores achieved by each method as a function of the training ratio variation, respectively. We observe that HuGE+ is comparable and in most cases better than existing popular methods on four real-world graphs. In particular, thanks to its effective heuristic sampling strategies, HuGE+ consistently outperforms the sampling-based models on all graphs in Macro – F1 and Micro – F1 scores, gaining 21.1% and 7.0% average improvements, respectively. Note that on the largest graph Youtube, HOPE, GraphGAN and GraphSAGE cannot efficiently handle this graph due to their complex computation operations (described in 3.2). On directed graph Wiki, VERSE achieves unsatisfactory performance relative to the comparable gains on undirected graphs, which is consistent with the observation in the evaluation of the link prediction task. ProNE outperforms all competitors in Micro – F1 score, but the performance in Macro – F1 score is less than impressive. This is because ProNE is specifically designed for node classification by leveraging the spectral propagation technique, and its effectiveness is lower for other tasks, such as the link prediction task.

3.4 Graph Clustering

To further verify the performance of our proposed HuGE+, we conduct graph clustering task on the learned node representation which detects groups of nodes with similar characteristics. We use the sklearn library K-means with the default K-means++ initialization [75] to cluster the embedded vectors in a d-dimensional space. To measure the clustering performance, clustering accuracy (ACC) and Normalized Mutual Information (NMI) score are used in our evaluations.

Table 4 reports the results on four real-world graphs. From it, we observe that HuGE+ can outperform the other baseline methods significantly in most cases in terms of both ACC and NMI, except for the NMI score of ProNE on the Wiki graph, but the performance of ProNE is not competitive in other cases compared to the baselines. Especially for the sampling-based methods: Deepwalk, Node2vec, LINE, VERSE, DiaRW and HuGE, HuGE+ exhibits clear performance advantages on all graphs, gaining 26.9% and 1.8 \times \) average improvements on ACC and NMI, respectively, further verifying the effectiveness of our proposed heuristic sampling strategies. Similar to the exhibitions in link prediction and multi-label classification task, graph neural networks-based technique: GraphGAN and GraphSAGE have limited scalability on the largest evaluated graph Youtube and their effectiveness is not comparable to the proposed method.

3.5 Efficiency and Scalability

Fig. 7 shows the time required by each method to generate node embedding. We observe that HuGE+ strikes the best balance between effectiveness and efficiency, and significantly outperforms the sampling-based methods on all graphs, by an average acceleration of 13.1 \times \) and the parallel version—HuGE+P is up to 23 \times 127 \times \) faster than all competitors.

<table>
<thead>
<tr>
<th>Method</th>
<th>PPI ACC</th>
<th>PPI NMI</th>
<th>Wiki ACC</th>
<th>Wiki NMI</th>
<th>Flickr ACC</th>
<th>Flickr NMI</th>
<th>Youtube ACC</th>
<th>Youtube NMI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deepwalk</td>
<td>0.206</td>
<td>0.042</td>
<td>0.278</td>
<td>0.011</td>
<td>0.217</td>
<td>0.060</td>
<td>0.194</td>
<td>0.005</td>
</tr>
<tr>
<td>Node2vec</td>
<td>0.190</td>
<td>0.050</td>
<td>0.252</td>
<td>0.007</td>
<td>0.223</td>
<td>0.061</td>
<td>0.189</td>
<td>0.003</td>
</tr>
<tr>
<td>LINE</td>
<td>0.148</td>
<td>0.045</td>
<td>0.219</td>
<td>0.014</td>
<td>0.164</td>
<td>0.059</td>
<td>0.104</td>
<td>0.001</td>
</tr>
<tr>
<td>HOPE</td>
<td>0.172</td>
<td>0.059</td>
<td>0.238</td>
<td>0.038</td>
<td>0.209</td>
<td>0.078</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>VERSE</td>
<td>0.216</td>
<td>0.036</td>
<td>0.269</td>
<td>0.002</td>
<td>0.166</td>
<td>0.066</td>
<td>0.205</td>
<td>0.004</td>
</tr>
<tr>
<td>ProNE</td>
<td>0.210</td>
<td>0.047</td>
<td>0.283</td>
<td>0.067</td>
<td>0.219</td>
<td>0.076</td>
<td>0.176</td>
<td>0.003</td>
</tr>
<tr>
<td>DiaRW</td>
<td>0.307</td>
<td>0.085</td>
<td>0.258</td>
<td>0.058</td>
<td>0.233</td>
<td>0.075</td>
<td>0.202</td>
<td>0.004</td>
</tr>
<tr>
<td>GraphGAN</td>
<td>0.234</td>
<td>0.085</td>
<td>0.264</td>
<td>0.042</td>
<td>0.178</td>
<td>0.052</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>GraphSAGE</td>
<td>0.209</td>
<td>0.044</td>
<td>0.245</td>
<td>0.065</td>
<td>0.164</td>
<td>0.057</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>HuGE</td>
<td>0.293</td>
<td>0.083</td>
<td>0.286</td>
<td>0.039</td>
<td>0.227</td>
<td>0.063</td>
<td>0.197</td>
<td>0.005</td>
</tr>
<tr>
<td>HuGE+</td>
<td>0.308</td>
<td>0.087</td>
<td>0.309</td>
<td>0.044</td>
<td>0.242</td>
<td>0.082</td>
<td>0.208</td>
<td>0.006</td>
</tr>
</tbody>
</table>

**TABLE 4**

\( \text{ACC} \) and \( \text{NMI} \) of HuGE+ and Baselines for Graph Clustering on Different Graphs, Where a “…” Signifies the Failure of the Corresponding Method Due to Compute-Resource or Running-Time (>5 Days) Constrains

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on March 01,2023 at 21:15:25 UTC from IEEE Xplore. Restrictions apply.
except ProNE. As illustrated in Table 3 and Fig. 6, ProNE is less effective compared to HuGE+ for the AUC and Macro – F1 score. While ProNE optimizes matrix factorization operation by graph partition techniques to meet the scalability for large-scale graphs, it still exerts a serious burden on DRAM. Note that GraphGAN, GraphSAGE, and HOPE cannot efficiently handle large graphs, such as Youtube, LiveJournal, and Twitter. In particular, GraphGAN and GraphSAGE are up to 2–3 orders of magnitude slower than HuGE+. We also note that in all methods, only HuGE+, HuGE and ProNE can successfully execute on the largest graph Twitter. To further test the scalability of HuGE+, we generate synthetic graphs [74] with a fixed node degree of 10 and the number of nodes from $10^4$ to $10^6$. Fig. 8 presents the running time for sampling and feature learning on the randomly generated graphs with different sizes, suggesting that the running time increases linearly with the size of a graph, and HuGE+ has the capability of handling billion-node-scale graphs. Moreover, the running time for five real-world graphs is inserted into the plot, which is consistent with the trend of synthetic data. Considering the resource consumption that affects scalability, we also evaluate the memory footprint of HuGE+ in Table 5. For fair comparison, we select the competitors that all leverage the Skip-Gram model to generate embedding vectors: Deepwalk, Node2vec, and DiaRW. Since HuGE+ inherits the advantage of heuristic random walk in HuGE and exhibits close performance in this case, there is no need to compare the two separately. The memory footprint here refers to the size of corpus generated by the sampler that will be trained in the Skip-Gram model. The results demonstrate that the memory footprint of HuGE+ is much smaller than Deepwalk, Node2vec and DiaRW on all graphs, by significant reductions of 80.2%, 80.7% and 46% on average, respectively. Although DiaRW is expected to have better efficiency and scalability by the variable-length strategy in random walk, it adopts a fixed number of walks per node as Deepwalk and Node2vec, ignoring the relationship between the size and the effectiveness in generated corpus.

### 3.6 Heuristic Methods for Embedding Enhancement

Recall that HuGE+ consists of three steps in its sampler: (1) hybrid-property heuristic random walk for exploiting the proximity of the pairwise nodes, (2) heuristic walk length for providing the variable-length walk, and (3) heuristic number of walks per node for determining the size of the generated corpus. The latter two heuristic methods help improve computing efficiency. Can they also work for the sampling-based baseline methods?

For the sampling procedure of Deepwalk, Node2vec, and DiaRW, we leverage the heuristic walk length (HWL) and the heuristic number of walks per node (HWN) instead of the fixed walk strategies (hyper-parameters $L$ and $r$). Figs. 10a and 10b exhibit both the original and enhanced results (denoted as Baseline-H) on real-world graphs for link prediction and multi-label classification, respectively, illustrating significant improvement achieved by the heuristic methods for the three competitors. Specifically, our scheme offers 11.7% average relative gains on the link prediction task, and provides 6.1% and 3.7% average improvement on Micro – F1 and Macro – F1 for the multi-label classification task, respectively. Moreover, all enhanced versions reduce running time by 58.3% and 39.4% on average for the two experiment tasks. The results demonstrate that the heuristic methods in HuGE+ are not only effective for HuGE+, but also present a general insight for improving graph embedding efficiency.

### 3.7 Parameter Sensitivity

We explore how different choices of parameters affect HuGE+ performance, including dimensionality $d$, context size $w$, HWL parameter $\mu$ and HWN parameter $\delta$. We leverage two tasks multi-label classification and link prediction to drive these evaluations, Fig. 9 depict the Macro – F1 scores

---

**TABLE 5**

<table>
<thead>
<tr>
<th>Graph</th>
<th>Deepwalk</th>
<th>Node2vec</th>
<th>DiaRW</th>
<th>HuGE+</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wiki-Vote</td>
<td>22MB</td>
<td>22MB</td>
<td>8.37MB</td>
<td>2.15MB</td>
</tr>
<tr>
<td>CA-AstroPh</td>
<td>80MB</td>
<td>82MB</td>
<td>17MB</td>
<td>9.52MB</td>
</tr>
<tr>
<td>Youtube</td>
<td>3.80GB</td>
<td>5.66GB</td>
<td>1.35GB</td>
<td>399MB</td>
</tr>
<tr>
<td>LiveJournal</td>
<td>8.32GB</td>
<td>8.34GB</td>
<td>1.53GB</td>
<td>902MB</td>
</tr>
<tr>
<td>PPI</td>
<td>14.93MB</td>
<td>15.12MB</td>
<td>4.35MB</td>
<td>1.55MB</td>
</tr>
<tr>
<td>Wiki</td>
<td>15.28MB</td>
<td>15.19MB</td>
<td>4.75MB</td>
<td>3.78MB</td>
</tr>
<tr>
<td>BlogCatalog</td>
<td>38.71MB</td>
<td>39.0MB</td>
<td>18MB</td>
<td>11.22MB</td>
</tr>
<tr>
<td>Flickr</td>
<td>355.52MB</td>
<td>355.61MB</td>
<td>213MB</td>
<td>183MB</td>
</tr>
<tr>
<td>Twitter</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>3.92GB</td>
</tr>
</tbody>
</table>

---

Fig. 7. Node embedding learning time required for each method on eight real-world graphs, where the y axis is in log-scale.

Fig. 8. Scalability of HuGE+ on synthetic graphs. The lines depict the running time required for sampling (blue line) and both sampling and feature learning (red line), respectively. Pentagrams show the time cost of feature learning (red line), respectively.
and AUC scores gained by HuGE+ for the multi-label classification task on PPI graph (Figs. 9a1-9d1) and link prediction task on Wiki-Vote graph (Figs. 9a2-9d2), respectively, both tasks are trained using 50% of the nodes. Except for the tested parameters, all the other parameters are set as default in the experiments. The dimensionality $d$ of the output vector representations determines the size of embedding and also affects the accuracy of downstream tasks. As shown in Figs. 9a1 and 9a2, there is plenty of room for improvement with small $d$, which means that the vectors need to improve the ability of representations, while continuously increasing the number of dimensions will risk over-fitting for neural networks. Meanwhile, we observe in Figs. 9b1 and 9b2 that increasing the context size $w$ in the Skip-Gram model will help improve the accuracy of the downstream tasks, which is not surprising since larger $w$ can exploit longer-range relation in the sentence and facilitate the network inference. However, if the context size $w$ is set too large, it will attenuate the impact of closer neighborhoods due to the introduced noise, resulting in compromised performance. In our approach, we introduce two parameters $\mu$ and $\delta$ as the judgment estimation for heuristic walk length and the number of walks per node, respectively.

Figs. 9c1 and 9c2 show that the Macro $F_1$ and AUC scores for both tasks grow linearly as $\mu$ approaches 0.99, explaining the linear correlation between walk length and the effectiveness of information representation. Similarly shown in Figs. 9d1 and 9d2, since the quality of generated corpus is related to $\delta$, the accuracy score trends to remain stable as $\delta$ decreases, indicating that the current corpus size is sufficient for concise and comprehensive representation. Overall, the parameter sensitivities of HuGE+ exhibits similar trend trajectories across different evaluation tasks on different graphs.

4 RELATED WORK

Graph embedding techniques, which focus on projecting the nodes of a graph into a low-dimensional vector space, while preserving structural and inherent properties of the graph, have received significant attention over the last few years [1], [42], [43]. In the current literature, graph embedding approaches roughly fall into three categories: sampling-based, matrix factorization-based and graph neural networks-based techniques.

Sampling-based techniques, inspired by the well-known natural language processing model word2vec [47], transform a graph into a set of random walk paths through sampling methods and then adopt word2vec (Skip-Gram) to generate graph embeddings from the sampling paths. Deepwalk [6] is a pioneer work extending word2vec to graph feature learning. It generates sequences of nodes from a graph by uniform random walk, which is commonly used in sampling methods but limited in preserving properties for complex graph structures. LINE [8] proposes a carefully designed objective function for preserving 1st- and 2nd-order node proximity and uses edge sampling instead of random walks. Nevertheless, the proximity preserved by LINE is restricted by the immediate neighbors of each node, which is insufficient for capturing the complete palette of node features [7], [10], [11]. Node2vec [7] introduces two hyperparameters to consider both homophily and structural equivalences in a biased random walk. Despite the fact that it facilitates flexibility in exploring node representations, it incurs high time and space overheads to laboriously tune hyperparameters. VERSE [11] proposes a generalized and efficient graph embedding framework that preserves the node proximity...
measured by Personalized Pagerank, but it ignores the asymmetries of similarity in the context graph and attempts to encode the similarities between vertices in a single embedding space [19], [67]. Some recently proposed approaches aim to use the heuristic-guide methods to generate the embeddings for some specific domain. Walklets [44] extends the DeepWalk to learn multiscale representations using random walks that skip or hop over multiple nodes at each step in the sampling procedure to generate multiscale relationships. JUST [45] proposes a heterogeneous graph embedding technique using random walks with Jump and Stay strategies to overcome the requirement for meta-path, flexibly choosing to change or maintain the type of the next node in the random walk. LBSN2Vec [46] designs a hypergraph embedding approach to break traditional hand-crafted features setting that performs random-walk-with-stay to sample the user mobility patterns and social relationships from the Location-Based Social Network for automatic feature learning. DiaRW [12] is proposed as a scalable graph embedding method based on a degree-biased random walk with variable length to sample node information. Although, these schemes achieve well performance for scenarios served by the heuristic methods, most of them leverage a routine random walk for sampling, ignoring the effectiveness and the efficiency in the generated walking path.


GNN-based techniques belong to another significant area that focuses on generalizing graph spectra into semi-supervised or supervised graph learning. Graph Convolution [20] proposes a first-order approximation to the K-localized graph convolution which extends convolution operation to non-euclidean spaces. GraphSAGE [22] introduces inductive representation learning, in which the node representation is learned by aggregating the representations of all neighbors. GraphGAN [24] learns node representations by modeling the connectivity behavior through an adversarial learning framework. ProGAN [25] presents a proximity generative adversarial network to address the problem of discovering the proximity between different nodes. Nevertheless, the compute overhead of most of GNN-based techniques is usually quite high [1], and the traditional deep learning architectures assume the input data on 1D or 2D grid to take advantage of GPU, which is not adaptable to graph [52].

5 Conclusion
In this work, we proposed HuGE+, an entropy-driven graph embedding approach with improved efficiency and scalability, which takes into account both the information content of node and the number of common neighbors to measure the proximity of pairwise nodes, and adopts heuristic methods for walk length and the number of walks per node to provide concise and comprehensive representation in the sampling procedure. HuGE+ is experimentally shown to be more efficient and effective than recent popular graph embedding benchmarks. Evaluation results demonstrate that HuGE+ is dozens of times faster in node embedding learning, and reduces memory footprint by an average of 68.9%. As future work, we would like to design a novel processing architecture based on either distributed or hybrid storage to accelerate HuGE+ further and consider dedicated application scenarios, such as dynamic and heterogeneous graphs. In addition, we are also interested in exploring the efficiency and scalability of graph neural network-based models.

References


Fang Fang received the BE and ME degrees in software engineering from Henan Polytechnic University, Jiaozuo, and Guangxi University, Nanjing, China, in 2013 and 2016, respectively. He is currently working toward the PhD degree majoring in computer architecture with the Huazhong University of Science and Technology, Wuhan, China. His current research interests include network, and graph processing, graph representation learning and big data analysis. He has several publications in major international conferences and journals, including ICDE, IWoQoS BESC, SBP-BRIMS, FCS and Pyhicsia A.

Fang Wang (Member, IEEE) received the BE and master’s degree in computer science, and the PhD degree in computer architecture from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 1994, 1997, and 2001, respectively. She is currently a professor of computer science and engineering in Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan, China. Her current research interests include computer architecture, parallel computing, and cloud computing. She has more than 80 publications in major international conferences and journals, including IEEE Transactions on Computers, IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Architecture and Code Optimization, ICDE, INFOCOM, SC, IODC, HPDC, ICPP, ICCD, MSST, DATE, and ICPP.

Zhan Shi (Member, IEEE) received the BE and ME, and PhD degree in computer science and technology from the Huazhong University of Science and Technology (HUST), China, in 1998, 2001 and 2011, respectively. He was postdoc research fellow with the University of Nebraska-Lincoln, Lincoln, Nebraska, in 2013-2014. He is currently an associate professor in the Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, China. His research interests include distributed storage, cloud storage and graph storage. His papers have been published in major journals and conferences, including ACM Transactions on Storage, IEEE Transactions on Services Computing, Journal of Parallel and Distributed Computing, Future Generation Computer Systems, ICDE, SC, EuroSys, ICPP and ICDCS.

Hong Jiang (Fellow, IEEE) received the BE degree from the Huazhong University of Science and Technology, Wuhan, China, in 1982, the MSc degree from the University of Toronto, Toronto, Canada, in 1987, and the PhD degree from the Texas A&M University, College Station, Texas, in 1991. He is Wendell H. Nedderman endowed professor and chair of the Department of Computer Science and Engineering, University of Texas at Arlington, Arlington, Texas. His research interests include computer architecture, computer storage systems, and parallel/distributed computing. He serves as an associate editor of IEEE Transactions on Parallel and Distributed Systems. He has more than 200 publications in major journals and international conferences in these areas, including IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers, ACM Transactions on Storage, ACM Transactions on Architecture and Code Optimization, Journal of Parallel and Distributed Computing, ISCA, MICRO, FAST, ICDE, USENIX ATC, USENIX LISA, SIGMETRICS, Middleware, IODC, IPDPS, OOPSLA, ECOOP, SC, ICS, HPDC, and ICPP.

Dan Feng (Member, IEEE) received the BE, ME, and PhD degrees in computer science and technology from the Huazhong University of Science and Technology (HUST), China, in 1991, 1994, and 1997, respectively. She is currently a professor in the School of Computer Science and Technology, Huazhong University of Science and Technology, China. Her research interests include computer architecture, massive storage systems, and parallel file systems. She has more than 100 publications in major journals and international conferences, including IEEE Transactions on Computers, IEEE Transactions on Parallel and Distributed Systems, ACM Transactions on Storage, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on Big Data, FAST, USENIX ATC, ISCA, SC, DAC, IODC, HPDC, ICS, ICCAD, IPDPS, MSST, DATE and ICPP.

Xianghao Xu received the BE degree in computer science and technology from the Liaoning University, Shenyang, China, in 2015, and the PhD degree in computer architecture from the Huazhong University of Science and Technology, Wuhan, China, 2021. He is currently an assistant professor in the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China. His current research interests include big data processing, computer architecture, and parallel and distributed computing. He has several publications in major international conferences and journals, including IEEE Transactions on Parallel and Distributed Systems, ICPP and IWoQoS.

Wei Yin received the BE degree in software engineering from the China University of Geosciences, Wuhan, China, in 2020. He is currently working toward the master’s degree majoring computer architecture with the Huazhong University of Science and Technology, Wuhan, China. His current research interests include high performance and parallel graph processing system.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/csdl.