A Novel Global Prototype-Based Node Embedding Technique

ZYAD ALKAYEM1, RAMI ZEWAIL1, (Member, IEEE), AMIN SHOUKRY1,2, DAI SUKE KAWAHARA3, AND SAMIR A. ELSAGHEER MOHAMED1,4

1Department of Computer Science and Engineering, Egypt–Japan University of Science and Technology, New Borg El-Arab City 21934, Egypt
2Department of Computer and Systems Engineering, Alexandria University, Alexandria 21544, Egypt
3School of Fundamental Science and Engineering, Waseda University, Tokyo 169-8555, Japan
4Department of Electrical Engineering, Aswan University, Aswan 81542, Egypt

Corresponding author: Zyad Alkayem (zyad.shokry@ejust.edu.eg)

ABSTRACT
Node embedding refers to learning or generating low-dimensional representations for nodes in a given graph. In the era of big data and large graphs, there has been a growing interest in node embedding across a wide range of applications, ranging from social media to healthcare. Numerous research efforts have been invested in searching for node embeddings that maximally preserve the associated graph properties. However, each embedding technique has its own limitations. This paper presents a method for generating deep neural node embeddings that encode dissimilarity scores between pairs of nodes with the help of prototype nodes spread throughout the target graph. The proposed technique is adaptable to various notions of dissimilarity and yields efficient embeddings capable of estimating the dissimilarity between any two pairs of nodes in a graph. We compare our technique against relevant state-of-the-art similar embedding techniques. Superior results have been demonstrated in a number of experiments using several benchmark datasets.

INDEX TERMS
Deep neural networks, link prediction, node embedding, unsupervised node classification.

I. INTRODUCTION
A node embedding technique seeks to project nodes into a latent space where geometric relations in this latent space correspond to relationships (e.g., edges) in the original graph or network [1]. However, there is no embedding technique that "fully" preserves both the topological graph properties (adjacency relations between nodes) and the node features relations (problem-dependent information). Research came up with many embedding techniques that vary in complexity, performance, and the graph properties required to be preserved. In general, node embedding methods fall under one of three groups: (i) factorization-based, (ii) random walk-based, and (iii) deep learning-based [2], [3]. Factorization-based algorithms aim to reconstruct some polynomial function of the adjacency matrix from the node embeddings. On the other hand, random walk-based methods follow a different approach. Rather than directly reconstructing a deterministic function of the graph’s adjacency matrix, random walk-based methods tend to optimise node embeddings such that two nodes are more likely to have similar embeddings if they tend to co-occur on short random walks over the graph [1], [4]. Deep neural methods generate node representations that depend on the structure of the graph as well as the feature information available. Complex deep encoders address the non-linearity and irregularity in the structure of the input graph.

In this paper, we do not consider the factorization-based embedding methods because they are not applicable to large graphs as they usually involve costly eigenvalue decomposition operations, and therefore they are of limited use [5], [6], [7]. Instead, we focus on random walk-based methods, specifically the node2vec scheme, and deep neural methods, specifically the GraphSage [8] scheme. The two schemes are widely used, state-of-the-art embedding schemes. However, each scheme has its own limitations. For example, node2vec does not leverage node features in the embedding learning process and can not generate embeddings for unseen nodes.
during the training process, a property called transduction [1], [9]. On the other hand, GraphSage assumes that nodes in the same neighborhood should have similar embeddings and depends mainly on node features, so that missing or partly noisy node features may significantly drop downstream task’s performance [10], [11]. The following highlights the main contributions in this paper:

- A novel deep prototype-based node embedding technique is presented. The proposed technique aims at alleviating some of the shortfalls of relevant state-of-the-art node embedding techniques. These include node2vec, GraphSage, and relevant representatives of random walk-based techniques.
- The performance of the proposed technique has been evaluated in node classification and link prediction tasks. Competitive results have been demonstrated against the state-of-the-art node2vec and GraphSage techniques.
- A thorough discussion and analysis is provided to explain the obtained results.

The remaining of this paper is organized as follows: Section II briefly overviews key node embedding techniques, specifically the node2vec and GraphSage embedding methods. Section III gives a detailed description of the proposed embedding technique. Section IV provides eight benchmark datasets used in the experimental work conducted in this paper along with a comparison of the performance of the novel proposed technique against state-of-the-art methods on both node classification and link prediction tasks. Section V discusses the relative merits of the proposed technique. Finally, Section VI concludes the paper.

II. RELATED WORK

This section presents a brief overview for the baseline node embedding methods most relevant to the method proposed in this paper.

A. RANDOM WALK-BASED EMBEDDINGS

Node2vec is chosen as a representative of the random walk-based methods. Node2vec derives its idea from word2vec [12], an embedding method in the natural language processing domain. Word2vec places words having similar context close in the embedding space. In a similar fashion, node2vec takes short random walks in a graph and treats the sequence of nodes in a random walk as a sequence of words in a sentence. The random walks gathered are then forwarded to a single hidden-layer neural network, commonly known as the Skip-gram model [13]. The Skip-gram model is trained to predict the surrounding words given a current word, or in the case of node2vec, to predict the surrounding neighbor nodes given a current node. Node2vec has been preceded by DeepWalk [14], an embedding scheme that follows the same aforementioned approach yet with unbiased random walks. Node2vec extends DeepWalk by employing parameters to control the random walk behavior. Specifically, node2vec utilizes a combination of breadth-first and depth-first node sampling to explore nearby and far nodes in the graph, respectively [15], [16].

B. GRAPH NEURAL NETWORKS (GNNs)

Over the past decade, graph neural networks methods have been receiving much attention. This class of methods was pioneered by the seminal work of [17] which introduced the Graph Convolutional Networks (GCNs) node embedding technique. The GNNs architectures share the following steps [18], [19]: the node features are initially the attribute vectors, then each node detects its neighbors and, accordingly, the feature vectors of the aggregated neighbors are compressed to update the node feature.

Among various methods in this category, GraphSage is a prominent example of GNNs. The feature aggregation function of GraphSage is given in (1).

$$h^k_{N(v)} = \text{AGGREGATE}_k \left( \left\{ h^k_{u} \mid u \in N(v) \right\} \right)$$

$$h^k_v = \sigma \left( W^k \cdot \text{CONCAT} \left( h^{k-1}_v, h^k_{N(v)} \right) \right),$$

(1)

where $N(v)$ is the neighborhood of node $v$ (usually $N(v)$ does not exceed the 2-hop), $h^k_{N(v)}$ is the feature set within node $v$’s neighborhood at the $k^{th}$ iteration, $h^k_v$ is node $v$’s representation at the $k^{th}$ iteration, $W^k$ is a trainable parameter matrix and $\sigma$ is the sigmoid function. The aggregator functions include the mean, the max and the LSTM [20] (long short-term memory). GraphSage tries to maximize the similarity of the embeddings of $u$ and $v$, where $u$ and $v$ are neighbors, and minimizes that between $u$ and $n$, where $n$ is a sampled non-neighbor of $u$. The similarity between the embeddings of two nodes, according to GraphSage’s convention, is the dot product of their embeddings.

Position-aware Graph neural networks (PGNNs) is another relevant method introduced in [21] which aims to address some serious limitations of typical GNNs. Existing GNNs are partially capable of detecting the location of a node with respect to other nodes in a graph. In a formal sense, a typical GNN usually fails to reconstruct the shortest path between two nodes given their generated embeddings. A PGNN learns a non-linear distance-weighted aggregation strategy over carefully sampled random subsets of nodes called anchor-sets and computes the distance from a particular target node to each anchor-set. As a result, PGNNs can accurately estimate node locations relative to anchor nodes besides being inductive, scalable, and able to incorporate node features.

III. PROPOSED EMBEDDING TECHNIQUE

The section is divided into two parts, A and B. Part A presents the foundation on which our work is based. Part B describes in detail the proposed embedding technique.

A. BACKGROUND

1) RADIAL BASIS FUNCTION NETWORKS (RBFNs)

Our work draws some inspirations from the typical RBFNs [22]. By default, an RBF is a single hidden-layer
The network described in this paper is an RBFN-like network, as shown in Fig. 1, trained using a Siamese-like training technique. Note that a Siamese network having $b$ branches can be simulated by training $b$ dataset samples back-to-back using only one branch. This view is the one adopted in this paper. The proposed framework, the adopted dissimilarity metrics as well as the training algorithm are described next in part B.

B. PROPOSED METHODOLOGY

This section presents the details of the proposed deep prototype-based node embedding technique. The proposed method aims to encode relational topological node information as well as available node features. The proposed method is unsupervised which makes it well suited for large graphs as well as small graphs.

Let $G$ be a given graph with set of vertices $V = \{v_1, v_2, \ldots, v_n\}$, $n = |V|$ and $F$ the matrix of nodes’ features of size $n \times f$ where $f$ is the size of a node’s feature vector. Several dissimilarity matrices $S^1, S^2, \ldots, S^k$ are extracted from $G$. In our basic formulation, the node-to-node shortest paths matrix and the pairwise cosine dissimilarity matrix based on $F$ are computed. Other dissimilarity matrices can be incorporated as well. Note that the network can adapt to either similarity or dissimilarity matrices, but we use the term “dissimilarity” throughout the paper to avoid confusion. Therefore, we convert any similarity measure, such as the cosine similarity, into a dissimilarity one. As we utilize multiple dissimilarity matrices, the hidden layer is preceded by a softmax layer of size $1 \times k$, where $k$ is the number of prototypes.
of dissimilarity matrices. Our target is to generate, for each node \( v_i \) in \( V \), a corresponding embedding vector \( e_i \) of size \( 1 \times e \) where \( e \) is a hyper-parameter that stands for the desired embedding size.

To generate an embedding \( e_i \) for node \( v_i \), the node index \( i \) is forward to the hidden layer to get the dissimilarity scores-against the hidden neurons prototypes-from the pre-computed dissimilarity matrices \( S^1, S^2, \ldots, S^k \). For a given dissimilarity matrix \( S \), a node \( v_i \) and a prototype node \( v_j \), the dissimilarity score is the element at index \((i, j)\) in \( S \). Let \( S^1, S^2, \ldots, S^k \) be the dissimilarity scores between node \( v_i \) and prototype node \( v_j \) for each \( S^1, S^2, \ldots, S^k \). These scores are multiplied by the normalized weights \( \hat{\alpha}_1, \hat{\alpha}_2, \ldots, \hat{\alpha}_k \). Each \( \hat{\alpha} \) is normalized according to the softmax rule:

\[
\hat{\alpha}_i = \frac{\exp(\alpha_i)}{\sum_j \exp(\alpha_j)},
\]

where each \( \alpha_j \) is a trainable parameter.

Now assume a hidden layer of \( z \) neurons and output layer of \( e \) neurons. Each output neuron acts as a linear summation unit for all the upcoming dissimilarity scores from the hidden layer yet weighted by the weights matrix \( z \times e \) between hidden and output layers. The embedding vector \( e_i \) of a node \( v_i \) is the result of the output layer.

To update the network’s weights, we follow the approach of Siamese Networks training: given two nodes \( v_a \) and \( v_b \), we forward them back-to-back to obtain respective embeddings \( e_a \) and \( e_b \). We find the cosine dissimilarity between \( e_a \) and \( e_b \):

\[
S_{\text{output}} = 1 - \frac{e_a \cdot e_b}{\|e_a\| \|e_b\|}.
\]

We minimize the L2-norm distance loss function:

\[
(S_{\text{output}} - S_{\text{true}})^2,
\]

where \( S_{\text{true}} \) is the equally weighted sum of the dissimilarity scores, between \( v_a \) and \( v_b \), defined in the dissimilarity matrices \( S^1, S^2, \ldots, S^k \).

IV. EXPERIMENTS

The proposed method is evaluated in two graph-related tasks, namely, unsupervised node classification and link prediction using eight benchmark graph datasets.

A. DATASETS DESCRIPTION

Our experiments are applied on the following graph datasets:

- Cora [24]: The Cora dataset contains 2708 scientific publications that are divided into seven categories. There are 5429 edges each indicating a citation of a paper in another one. A binary word vector describes each publication in the dataset, indicating the existence or absence of the associated word from a dictionary of 1433 unique words.
- BlogCatalog [25]: A social blog directory hosting the blogs of the users (bloggers). A node corresponds to a blogger in the network, and an edge indicates a friendship between two bloggers and the attributes vector of each node is based on recording occurrences of specific keywords in the blogger’s blog descriptions.
- LastFM [26]: The dataset is collected from LastFM social network where nodes are LastFM users from Asian countries and edges are mutual follower ties between nodes. The users’ favorite artists are used to extract the node features. Node labels represent the countries the users belong to.
- MIT [27]: Facebook friendship social network for Massachusetts Institute of Technology (MIT) in which seven properties are attached to each Facebook user: status flag, gender, major, second major, dorm, year, and high school. The ‘year’ property is taken as the label while the other six properties are set as features via one-hot encoding.
- Flickr [28]: Flickr is an online network where people can share photos and follow each other. Nodes and edges represent users and followers respectively. The feature vector of a node reflects a user’s interests based on a list of tags whereas the label of the node is the category of images the user is interested in.
- WikiCS [29]: The dataset is derived from Wikipedia articles and consists of nodes and edges corresponding to Computer Science articles and hyperlinks, respectively. The articles correspond to 10 classes representing different fields.
- Amazon [30]: A dataset of two networks: Amazon Computers (A. Computers) and Amazon Photos (A. Photos). Nodes represent goods, and edges indicate that two goods are frequently purchased together. The product reviews, in form of bag-of-words, node features may be used to map goods to their product categories.

Datasets statistics are summarized in Table 1. The experiments show the performance of our model against node2vec and GraphSage over node classification and link prediction tasks. Note that we apply unsupervised node classification across all experiments and all embedding schemes.

B. UNSUPERVISED NODE CLASSIFICATION TASK

For a fair comparison between the different node embedding methods, we tried to standardize the factors involved in each experiment when possible. We conducted our experiments for 16 length embedding vector, 20 epochs with batch size of 50, train-test dataset split of 50% and Adam optimizer with learning rate \( = 0.005 \). For the GraphSage method, for each sampled node, we sampled up to the 2\textsuperscript{nd} neighborhood (i.e., 2 hops) with 25 and 10 nodes for the first and second hops respectively. GraphSage using mean aggregators is the variant considered in this experiment. Table 2 shows node classification results.

C. LINK PREDICTION TASK

The results for the link prediction are shown in Table 3. The reported results are using the AUC [31] (area under the ROC curve) metric. The link prediction experiment was
conducted as follows: we randomly remove 10% edges from each dataset’s graph as positive testing data. We sample the same number of nonexistent edges (unconnected node pairs) at random as the negative testing data. To create the training data, we use the remaining 90% of existing links and also the same number of additional sampled nonexistent links. To obtain the nodes embeddings for each training graph, all embedding methods are trained on the training graph (after removing the positive edges) of each dataset.

Later, we compute the embedding of existing edges (positive) or nonexistent (negative) between pairs of nodes in a graph by applying one of four operations: Hadamard or element-wise multiplication, averaging, L1 norm or L2 norm on the embeddings of the considered node pairs. A logistic regression classifier is applied on the edges embeddings to classify the respective edges into two classes: positive (or likely there is a link between the two nodes for which this edge embedding is computed) and negative.

We transform the task into a supervised one by defining a node-to-node edge indicator with value 1 for no edge and value 0 for an existing edge. The following term is added to the loss objective function given in (4).

\[(1 − \hat{y})(D(\mathbf{z}_u, \mathbf{z}_v)) + (\hat{y})(\max(m − D(\mathbf{z}_u, \mathbf{z}_v), 0)),\] (5)

where \(\hat{y}\) is a binary indicator whether an edge exists between two nodes \(u\) and \(v\), \(D()\) is the Euclidean distance between the two (normalized) embeddings \(\mathbf{z}_u\) and \(\mathbf{z}_v\) of \(u\) and \(v\) respectively and \(m = 0.33\). Equation (5) tends to minimize the Euclidean distance between two connected nodes embeddings.

### V. DISCUSSION

The method proposed in this paper can be viewed more as a global rather than a local embedding technique. This means that during nodes embeddings generation process, we capture the relation(s) among nodes no matter how distant these nodes are within the graph. This has an advantage against local embedding techniques, which may fail to look beyond a node’s direct neighborhood. Regarding our method, the embedding for any node in the graph is acquired after matching the similarities between this node and a set of fixed reference nodes (prototypes) spread across the entire graph. In that sense, the direct neighborhood of a node is not our focus, as opposed to the strategies adopted in GraphSage and node2vec. Although GraphSage and node2vec, in the light of our presentation, would be considered local embedding methods, extra costly measures can be adopted to model more global views of the input graph. For example, in node2vec, random walks can model higher-order relations among nodes yet for long path lengths. Hence, GraphSage and node2vec can be considered as “inherently” local. Being a global embedding technique is an add-on feature when dealing with complex graph datasets.

The performance of node classification, link prediction and many other downstream tasks depends on the nature of the presented graph and how informative the external information given in the form of feature vectors is. The results in Table 2 show the advantage our method against the rest of the methods over 50% of the datasets in the node classification task. Let us consider node classification over the LastFM dataset in particular. In comparison with GraphSage and node2vec, our method outperforms due to the LastFM dataset nature. Feature vectors contain information about the favorite artists each user listens to, which is, at least, as important as user-to-user connections in determining a user’s country. As a consequence, node2vec performance would suffer since it takes into account only the structural information. Moreover, higher-order connections among a group of users or between a user and other users spread across the network can provide richer information about an individual’s identity than low-order connections with immediate neighbors. This may explain why our method performs better than GraphSage over the social networks LastFM and BlogCatalog.

The global property of an embedding technique may not always be a favorable aspect. In some datasets, the close connections between nodes are sufficient to produce embeddings that effectively encode relevant graph information. Further work to draw higher-order relations among nodes may be redundant and sometimes disturb the performance of the downstream task being considered. Supporting this point is the Amazon Computers dataset. It is a reasonable conclusion that products that are bought together (as indicated by the graph links) usually belong to the same category. Hence, and in similar contexts, a local embedding technique may outperform a global one in predicting the nodes’ classes.

### TABLE 1. Datasets characteristics.

|          | # nodes | # edges | # classes | # attributes |
|----------|---------|---------|-----------|--------------|
| Cora     | 2708    | 5278    | 7         | 1433         |
| BlogCatalog | 5196   | 171743  | 6         | 8189         |
| MIT      | 5928    | 217118  | 32        | 2804         |
| Flickr   | 7575    | 239738  | 9         | 12047        |
| LastFM   | 7624    | 27806   | 18        | 7842         |
| WikiCS   | 11701   | 216123  | 10        | 300          |
| A. Photos | 7650   | 119081  | 8         | 745          |
| A. Computers | 13752 | 245861  | 10        | 767          |

### TABLE 2. Node classification accuracy results. The best results are highlighted.

|          | Node2vec | GraphSage | Our Method |
|----------|----------|-----------|------------|
| Cora     | 0.7356   | 0.7563    | 0.7579     |
| BlogCatalog | 0.6506 | 0.6200    | 0.8357     |
| MIT      | 0.7503   | 0.4162    | 0.5829     |
| Flickr   | 0.5231   | 0.5850    | 0.5222     |
| LastFM   | 0.1737   | 0.2098    | 0.4295     |
| WikiCS   | 0.7596   | 0.7259    | 0.6241     |
| A. Photos | 0.8838   | 0.7020    | 0.8988     |
| A. Computers | 0.8306 | 0.7081    | 0.7745     |
TABLE 3. Link prediction results in AUC metric. The best results are highlighted.

|       | Node2vec | GraphSage | Our Method |
|-------|----------|-----------|------------|
|       | Hadamard | Average   | L1         | L2         | Hadamard | Average | L1 | L2         | Hadamard | Average | L1     | L2     |
| Cora  | 0.6615   | 0.5139    | 0.8654     | 0.8722     | 0.9103   | 0.7019   | 0.9428 | 0.9535     | 0.7992   | 0.4869  | 0.7210 | 0.5923  |
| BlogCatalog | 0.7745   | 0.5603    | 0.8729     | 0.8801     | 0.5266   | 0.4389   | 0.8036 | 0.7782     | 0.7551   | 0.5084  | 0.7616 | 0.7356  |
| MIT   | 0.8191   | 0.6260    | 0.9251     | 0.9316     | 0.8067   | 0.7646   | 0.8240 | 0.8655     | 0.8475   | 0.3823  | 0.7958 | 0.7801  |
| Flickr| 0.5894   | 0.6294    | 0.861      | 0.8701     | 0.9251   | 0.9222   | 0.9224 | 0.9201     | 0.4439   | 0.5668  | 0.6229 | 0.5966  |
| LastFM| 0.74    | 0.5732    | 0.9385     | 0.9398     | 0.5001   | 0.6358   | 0.8301 | 0.6492     | 0.7922   | 0.5635  | 0.8011 | 0.8076  |
| WikiCS| 0.8572   | 0.6688    | 0.9569     | 0.9608     | 0.8285   | 0.7984   | 0.8208 | 0.9098     | 0.8696   | 0.6290  | 0.8184 | 0.8032  |
| A. Photos | 0.9056   | 0.5729    | 0.9791     | 0.9805     | 0.6288   | 0.6972   | 0.9182 | 0.8360     | 0.9186   | 0.5288  | 0.9256 | 0.9116  |
| A. Computers | 0.8131 | 0.5511    | 0.9738     | 0.9756     | 0.6955   | 0.4948   | 0.8436 | 0.3186     | 0.8857   | 0.6433  | 0.9119 | 0.8944  

TABLE 4. Node classification accuracy over Flickr dataset after introducing noise to 25%, 50% and 75% of the node features respectively.

| Method      | 25% | 50% | 75% |
|-------------|-----|-----|-----|
| GraphSage   | 0.3042 | 0.2445 | 0.2260 |
| Our Method  | 0.4317 | 0.4252 | 0.4297 |

![FIGURE 2. Node classification accuracy against percentage of prototype nodes over Cora dataset.](image)

TABLE 5. Node classification accuracy over LastFM dataset for different dissimilarity measures.

| Dissimilarity Measure                        | Accuracy  |
|----------------------------------------------|-----------|
| Shortest Path                               | 0.1918    |
| Shortest Path + Cosine Dissimilarity         | 0.4295    |
| Shortest Path + Cosine Dissimilarity + SimRank | 0.4028    |
| Shortest Path + Cosine Dissimilarity + Labels Dissimilarity | 0.8410    |

Useful embeddings of nodes by aggregating (taking average in our case) the features of their neighborhoods whereas our method maintains the cosine similarity (sum of Hadamard product yet unnormalized) between two nodes embeddings to be directly proportional to the similarity defined between the two nodes. On the other hand, node2vec is the clear winner in particular over L1 and L2 operators and in general if performance, regardless of the edge operator, is the only choice. However, the last finding contradicts with [15] where it implies that node2vec performs best using Hadamard.

The choice of prototype nodes in the graph is made at random. This factor, along with several other factors, bears the entry of noise into the node features compared to GraphSage. In general, GNNs are sensitive to even slight perturbations to graph properties such as modifying graph topology or introducing noise to node features [32], [33]. Table 4 shows the node classification accuracy of our method against GraphSage after adding noise to the node features of Flickr dataset. Noise introduced is random binary vectors added to specific ratios of node features added to specific ratios of node features.

The results in Table 4 show nearly 10% drop in node classification accuracy using our method after introducing noise to 75% of node features while GraphSage experienced about 34% drop in accuracy. Also, the results showed that our method remained resilient as compared to GraphSage that suffered after each increase in the percentage of noise infected feature vectors.

There are many alternatives rather than choosing prototype nodes at random. One scheme is to consider the celebrity nodes, or the nodes with the highest number of connections. Aside from prototypes assignment, setting the percentage of prototype nodes remains an important hyper-parameter. In our experiments, we find setting 30% of graph nodes as prototypes to be the best choice as demonstrated by Fig. 2.

Our method supports multiple dissimilarity measures if each measure can be defined in matrix form. We tested node classification accuracies, reported in Table 5 after applying a mixture of the dissimilarity measures over the LastFM dataset. We use the (i) shortest path, (ii) node features cosine dissimilarity, (iii) SimRank [34] and (iv) pairwise label dissimilarity (score of 0 for two nodes of same class and 1 for two nodes of different classes) measures.

The results show that combining both shortest path and node features cosine dissimilarity metrics has superior performance than that of the shortest path metric only. However, combining different dissimilarity metrics not always boost the performance as indicated by the third row in Table 5. The last result indicates that including pairwise label dissimilarity significantly increased performance. Note that utilizing labels dissimilarity among nodes is illogical or unfair when applying node classification task. Nevertheless, the last result is only to demonstrate that a carefully selected dissimilarity metric can greatly affect the performance of the applied downstream task. A last note to mention is that some dissimilarity metrics can be costly to compute. For instance, the computation of the shortest path distances has computational complexity of $O(n^3)$, for $n$ vertices in graph. Fortunately, we may resort to some relaxations such as limiting computation of the shortest path distances only to some $k$-hop neighborhood around the target nodes.
Finally, it is worth mentioning that PGNNs and our proposed method share the idea of employing anchor nodes (we call them prototypes) across the graph in the process of embeddings generation. However, the details of the two schemes and their purposes are different. For instance, the strategy of anchor sets sampling and the architectures are not the same. Another example is that, unlike our embedding framework, PGNNs are mainly dependent on noise-free node features as PGNNs are a variant of GNNs.

VI. CONCLUSION

In this paper, we propose a new method for generating node representations in graphs. The proposed method is evaluated on the two most famous graph related tasks: node classification and link prediction. Results are presented using a number of benchmark datasets. As a conclusion, we recommend using our method when dealing with complex, multi-dissimilarity measures or noise-inflicted networks. Regarding our future work, we plan to support supervised downstream tasks and enhance the performance of the proposed technique.

REFERENCES

[1] W. L. Hamilton, “Graph representation learning,” Synth. Lectures Artif. Intell. Mach. Learn., vol. 14, no. 3, pp. 1–159, 2020.

[2] P. Goyal and E. Ferrara, “Graph embedding techniques, applications, and performance: A survey,” Knowl.-Based Syst., vol. 151, pp. 78–94, Jul. 2018.

[3] M. Xu, “Understanding graph embedding methods and their applications,” 2020, arXiv:2012.08819.

[4] F. Xia, J. Liu, H. Nie, Y. Fu, L. Wan, and X. Kong, “Random walks: A review of algorithms and applications,” IEEE Trans. Emerg. Topics Comput. Intell., vol. 4, no. 2, pp. 95–107, Apr. 2020, doi: 10.1109/TELCI.2019.2952908.

[5] J. Luo, S. Xiao, S. Jiang, H. Gao, and Y. Xiao, “Ripple2vec: Node embedding with ripple distance of structures,” Data Sci. Eng., vol. 7, no. 2, pp. 156–174, Jun. 2022, doi: 10.1007/s41019-022-00184-6.

[6] H. Cai, V. W. Zheng, and K. C.-C. Chang, “A comprehensive survey of graph embedding: Problems, techniques and applications,” 2017, arXiv:1709.07604.

[7] J. Zhou, L. Liu, W. Wei, and J. Fan, “Network representation learning: From preprocessing, feature extraction to node embedding,” 2021, arXiv:2110.07582.

[8] W. L. Hamilton, R. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” 2017, arXiv:1702.03722.

[9] I. Chami, S. Abu-El-Haija, B. Perozzi, C. Ré, and K. Murphy, “Machine learning on graphs: A model and comprehensive taxonomy,” J. Mach. Learn. Res., vol. 23, no. 89, pp. 1–64, 2022. [Online]. Available: http://jmlr.org/papers/v23/20-852.html.

[10] D. Zügner, A. Akbarnejad, and S. Günnemann, “Adversarial attacks on neural networks for graph data,” in Proc. 5th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining. Jul. 2018, pp. 2847–2856, doi: 10.1145/3198189.3220078.

[11] T. Schumacher, H. Wolf, M. Ritzert, F. Lemmerich, J. Bachmann, D. Zügner, A. Akbarnejad, and S. Günnemann, “Adversarial attacks on neural networks for node and graph classification,” 2019, arXiv:1906.02319.

[12] S. Hochreiter and J. J. Schmidhuber, “Long short-term memory,” Neural Comput., vol. 9, no. 8, pp. 80–1735, 1997.

[13] J. You, R. Ying, and J. Leskovec, “Position-aware graph neural networks,” 2019, arXiv:1906.08171.

[14] J. Mark, “Introduction to radial basis function networks,” Centre Cogn. Sci., Univ. Edinburgh, Edinburgh, Scotland, Tech. Rep., 1996. [Online]. Available: https://faculty.cc.gatech.edu/~isbell/tutorials/rbf-intro.pdf.

[15] G. R. Koch, “Siamese neural networks for one-shot image recognition,” Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2015. [Online]. Available: https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf.

[16] B. Perozzi, R. Al-Rfou, and S. Skiena, “DeepWalk: Online learning of social representations,” in Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining. New York, NY, USA: Association for Computing Machinery, Aug. 2014, pp. 701–710, doi: 10.1145/2623330.2623732.

[17] A. Grover and J. Leskovec, “Node2vec: Scalable feature learning for networks,” 2016, arXiv:1607.06553.

[18] J. Scholler, M. Wehking, F. S. Rizi, and M. Granitzer, “Investigating extensions to random walk based graph embedding,” in Proc. IEEE Int. Conf. Cognitive Comput. (ICCC), Jul. 2019, pp. 81–89, doi: 10.1109/ICCC.2019.00026.

[19] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” 2016, arXiv:1609.02907.

[20] S. Zhang, H. Tong, J. Xu, and R. Maciejewski, “Graph convolutional networks: A comprehensive review,” Comput. Social Netw., vol. 6, no. 1, pp. 1–23, Dec. 2019.

[21] J. Wu, J. He, and J. Xu, “DEMO-Net: Degree-specific graph neural networks for node and graph classification,” 2019, arXiv:1906.02319.

[22] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore, “Automating the construction of internet portals with machine learning,” Inf. Retr., vol. 3, no. 2, pp. 127–163, 2000, doi: 10.1023/A:1009583148834.

[23] R. Rossi and N. Ahmed, “The network data repository with interactive graph analytics and visualization,” in Proc. AAAI Conf. Artif. Intell., Mar. 2015, vol. 29, no. 1, pp. 1–2. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/9277.

[24] B. Rozenbergczki and R. Sarkar, “Characteristic functions on graphs: Birds of a feather, from statistical descriptors to parametric models,” in Proc. 29th ACM Int. Conf. Inf. Knowl. Manage., Oct. 2020, pp. 1325–1334.

[25] C. Hou, S. He, and K. Tang, “RoSANE: Robust and scalable attributed network embedding for sparse networks,” Neurocomputing, vol. 409, pp. 231–243, Oct. 2020, doi: 10.1016/j.neucom.2020.05.080.

[26] X. Huang, J. Li, and X. Hu, “Label informed attributed network embedding,” in Proc. 10th ACM Int. Conf. Web Search Data Mining, Feb. 2017, pp. 1–9, doi: 10.1145/3018661.3018667.

[27] L. Hou, S. He, and K. Tang, “RoSANE: A wiki-based benchmark for graph neural networks,” 2020, arXiv:2007.02901.

[28] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, “Pitfalls of graph neural network evaluation,” 2018, arXiv:1811.05868.

[29] A. P. Bradley, “The use of the area under the ROC curve in the evaluation of machine learning algorithms,” Pattern Recognit., vol. 30, pp. 1145–1159, Jul. 1997. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320396001422.

[30] C. Agarwal, M. Zitnik, and H. Lakkaraju, “Probing GNN explainers: A rigorous theoretical and empirical analysis of GNN explanation methods,” 2021, arXiv:2106.09078.

[31] Z. Alkayem et al.: Novel Global Prototype-Based Node Embedding Technique
RAMI ZEWAIL (Member, IEEE) received the B.Sc. and M.Sc. degrees from the Arab Academy for Science and Technology, Egypt, and the Ph.D. degree from the University of Alberta, Canada. He has over 15 years of academic and industrial research and development experience in areas of computer vision, machine learning, and embedded intelligence. He currently works as an Assistant Professor with the Department of Computer Science and Engineering, Egypt–Japan University of Science and Technology, Alexandria, Egypt. His research experience spans different fields, such as healthcare, oil and gas industry, and biometrics authentication. His research interests include computer vision, machine learning, and embedded intelligence with focus on learning in resource-constrained settings. He is a member of the Canadian Association for Artificial Intelligence. He serves as a reviewer for a number of scientific journals.

AMIN SHOUKRY received the B.Sc. and M.Sc. degrees from the Department of Computer Science and Automatic Control, Faculty of Engineering, Alexandria University, in 1973 and 1978, respectively, the Diploma degree in computer science from Pierre and Marie Curie University, Paris, France, in 1979, and the Ph.D. degree from the AI Group, Informatics Research Center, Nancy, France, in 1983. Next, he joined the AI Group, Informatics Research Center. He is currently a Professor with the Department of Computer Science and Engineering, Egypt–Japan University of Science and Technology, Alexandria, Egypt. He has published 70 papers in well-known journals and attended international as well as national conferences. His research interests include computational intelligence, machine learning, and pattern recognition. He was a recipient of the 1997 Egyptian National Research Academy Award in the area of computer and operations research.

DAISUKE KAWAHARA received the B.Sc. and M.Sc. degrees in electronic science and engineering and the Ph.D. degree in informatics from Kyoto University, in 1997, 1999, and 2005, respectively. He is currently a Professor with the Department of Communications and Computer Engineering, Waseda University, Tokyo, Japan. His research interests include natural language processing, particularly knowledge acquisition and text understanding.

SAMIR A. EL-SAGHEER MOHAMED received the B.Sc. degree in computer and control systems from the Faculty of Engineering, University of Assiut, Egypt, in May 1994, and the M.Sc. and Ph.D. degrees in computer science and engineering from the University of Rennes I, France, in 1998 and 2003, respectively. He joined the Faculty of Engineering, Aswan University, as an Assistant Professor, from 2006 to 2013. He was promoted to an Associate Professor and worked with Qassim University, Saudi Arabia, until 2019. Thereafter, he joined the Egypt–Japan University of Science and Technology. He has got 13 grants for research projects in different areas and from different grantors. His research interests include network security, cyber security, security in the Internet of Vehicles (IoV), and connected vehicles.