Unsupervised Inductive Whole-Graph Embedding by Preserving Graph Proximity

Yunsheng Bai[1], Hao Ding[2], Yang Qiao[1], Agustin Marinovic[1], Ken Gu[1], Ting Chen[1],
Yizhou Sun[1], Wei Wang[1]
[1]University of California, Los Angeles
[2]Purdue University
yba@ucla.edu, ding209@purdue.edu, angelinana0408@gmail.com, amarinovic@ucla.edu,
ken.qgu@gmail.com, {tingchen, yzsun, weiwang}@cs.ucla.edu

1 LEARNING GRAPH EMBEDDINGS FROM GRAPH PROXIMITY METRICS

We introduce ***UGraphEmb*** (*U*nsupervised *Graph*-level *Emb*edding) for learning graph-level representations in an unsupervised and inductive way. Recent years we have witnessed the great popularity of graph representation learning with success in not only node-level tasks such as node classification [23] and link prediction [59] but also graph-level tasks such as graph classification [58] and graph similarity/distance computation [1].

There has been a rich body of work [2, 35] on node-level embeddings that turn each node in a graph into a vector preserving node-node proximity (similarity/distance). It is thus natural to raise the question: Can we embed an entire graph into a vector in an unsupervised way, and how? However, most existing methods for graph-level, i.e. whole-graph, embeddings assume a supervised model [61], with only a few exceptions such as Graph Kernels [57], which typically count subgraphs for a given graph and can be slow, and Graph2Vec [30], which is transductive.

A key challenge facing designing an unsupervised graph-level embedding model is the lack of graph-level signals in the training stage. Unlike node-level embedding which has a long history in utilizing the link structure of a graph to embed nodes, there lacks such natural proximity (similarity/distance) information between graphs. Supervised methods, therefore, typically resort to graph labels as guidance and use aggregation based methods, e.g. average of node embeddings, to generate graph-level embeddings, with the implicit assumption that good node-level embeddings would automatically lead to good graph-level embeddings using only "***intra*-graph** information" such as node attributes, link structure, etc.

However, this assumption is problematic, as simple aggregation of node embeddings can only preserve limited graph-level properties, which is, however, often insufficient in measuring graph-graph proximity ("***inter*-graph**" information). Inspired by the recent progress on graph proximity modeling [1, 25], we propose a novel framework, UGraphEmb that employs multi-scale aggregations of node-level embeddings, guided by the graph-graph proximity defined by well-accepted and domain-agnostic graph proximity metrics such as Graph Edit Distance (GED) [4], Maximum Common Subgraph (MCS) [5], etc.

The goal of UGraphEmb is to learn high-quality ***graph-level*** representations in a completely ***unsupervised*** and ***inductive*** fashion: During training, it learns a function that maps a graph into a universal embedding space best preserving graph-graph proximity, so that after training, any new graph can be mapped to this embedding space by applying the learned function. Inspired by the recent success of pre-training methods in the text domain, such as ELMO [34], Bert [7], and GPT [37], we further fine-tune the model via incorporating a supervised loss, to obtain better performance in downstream tasks, including ***graph classification***, ***graph similarity ranking***, ***graph visualization***, etc.

2 THE PROPOSED APPROACH: UGRAPHEMB

We present the overall architecture of our unsupervised inductive graph-level embedding framework UGraphEmb in Figure 3. The key novelty of UGraphEmb is the use of graph-graph proximity. To predict the proximity between two graphs, UGraphEmb generates one embedding per graph from node embeddings using a novel mechanism called **Multi-Scale Node Attention (MSNA)**, and computes the proximity using the two graph-level embeddings.

2.1 Inductive Whole-Graph Embedding

2.1.1 Node Embedding Generation. For each graph, UGraphEmb first generates a set of node embeddings using neighbor aggregation methods based on Graph Convolutional Networks (GCN) [23], which are permutation-invariant and inductive. We adopt the most recent and state-of-the-art method, Graph Isomorphism Network (GIN) [54], in our framework. GIN has been proven to be theoretically the most powerful GNN under the neighbor aggregation framework [54].

2.1.2 Graph Embedding Generation. After node embeddings are generated, UGraphEmb [1] generates one embedding per graph using these node embeddings. Existing methods are typically based on aggregating node embeddings, by either a simple sum or average, or some more sophisticated way to aggregate. Appendix B provides a more detailed survey and explanation on some intuitions described next.

[1]This paper is already published in IJCAI'19.

KDD '19, August 03–07, 2019, Anchorage, AK

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

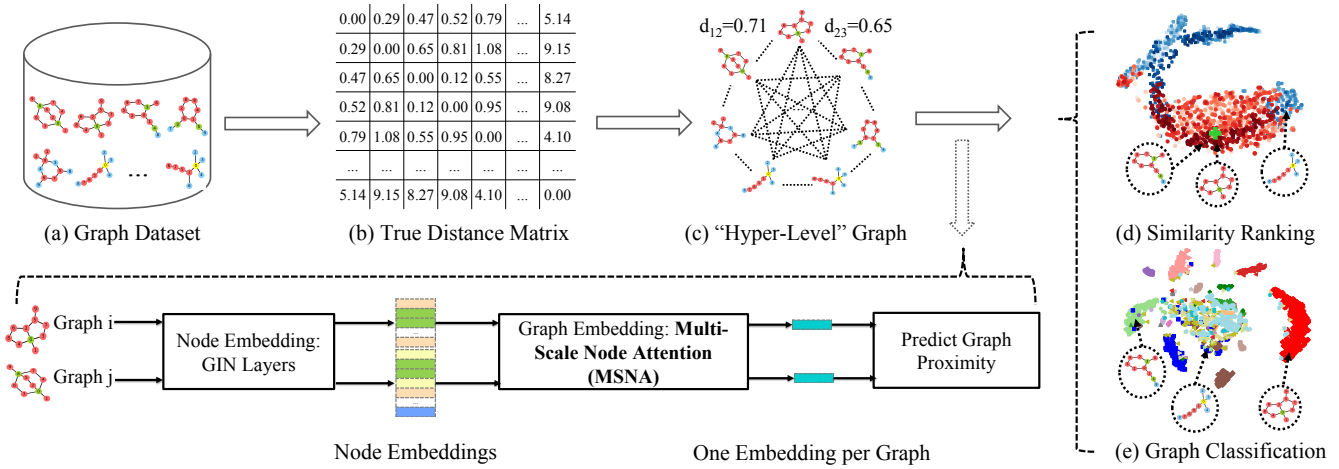

Figure 1: Overview of UGraphEmb. (a) Given a set of graphs, (b) UGraphEmb first computes the graph-graph proximity scores , (c) yielding a "hyper-level graph" where each node is a graph in the dataset, and each edge has a proximity score associated with it, representing its weight/strength. UGraphEmb then trains a function that maps each graph into an embedding which preserves the proximity score. (d) After embeddings are generated, similarity ranking can be performed. The green "+" sign denotes an example query graph. Red and blue colors indicate how similar and dissimilar a graph is to the query based on the ground truth. (e) Finally, UGraphEmb can perform fine-tuning on the proximity-preserving whole-graph embeddings, adjusting them for the task of graph classification.

However, since our goal is to embed each graph as a single point in the embedding space that preserves graph-graph proximity, the graph embedding generation model should capture structural difference at multiple scales [55] and be adaptive to different graph proximity metrics. Thus, we propose the following **Multi-Scale Node Attention (MSNA)** mechanism. Denote the input node embeddings of graph $\mathcal{G}$ as $\boldsymbol{U_G} \in \mathbb{R}^{N \times D}$, where the n-th row, $\boldsymbol{u_n} \in \mathbb{R}^D$ is the embedding of node n. The graph level embedding is obtained as follows:

$$\boldsymbol{h_G} = \text{MLP}_{\boldsymbol{W}} \left(\Big\Vert_{k=1}^{K} \text{ATT}_{\boldsymbol{\Theta}^{(k)}}(\boldsymbol{U_G}) \right) \tag{1}$$

where $\Vert$ denotes concatenation, K denotes the number of neighbor aggregation layers, ATT denotes the following multi-head attention mechanism that transforms node embeddings into a graph-level embedding, and $\text{MLP}_{\boldsymbol{W}}$ denotes multi-layer perceptrons with learnable weights $\boldsymbol{W}$ applied on the concatenated attention results.

The intuition behind Equation 1 is that, instead of only using the node embeddings generated by the last neighbor aggregation layer, we use the node embeddings generated by each of the K neighbor aggregation layers. ATT is defined as follows:

$$\text{ATT}_{\boldsymbol{\Theta}}(\boldsymbol{U_G}) = \sum_{n=1}^{N} \sigma(\boldsymbol{u_n}^T \text{ReLU}(\boldsymbol{\Theta}(\frac{1}{N} \sum_{m=1}^{N} \boldsymbol{u_m})))\boldsymbol{u_n}. \tag{2}$$

where N is the number of nodes, σ is the sigmoid function $\sigma(x) = \frac{1}{1+\exp(-x)}$, and $\boldsymbol{\Theta^{(k)}} \in \mathbb{R}^{D \times D}$ is the weight parameters for the k-th node embedding layer. During the generation of whole-graph embeddings, the attention weight assigned to each node should be adaptive to the graph proximity metric. To achieve that, the weight is determined by both the node embedding $\boldsymbol{u_n}$, and a learnable graph representation. The learnable graph representation is adaptive to a particular graph proximity via the learnable weight matrix $\boldsymbol{\Theta^{(k)}}$.

2.2 Unsupervised Loss via Inter-Graph Proximity Preservation

2.2.1 Definition of Graph Proximity. The key novelty of UGraphEmb is the use of graph-graph proximity. It is important to select an appropriate graph proximity (similarity/distance) metric.

In this paper, we use GED as an example metric to demonstrate UGraphEmb. GED measures the minimum number of edit operations to transform one graph to the other, where an edit operation on a graph is an insertion or deletion of a node/edge or relabelling of a node. Thus, the GED metric takes both the graph structure and the node labels/attributes into account. Appendix C contains more details on GED.

2.2.2 Prediction of Graph Proximity. Once the proximity metric is defined, and the whole-graph embeddings for $\mathcal{G}_i$ and $\mathcal{G}_j$ are obtained, denoted as $\boldsymbol{h_{\mathcal{G}_i}}$ and $\boldsymbol{h_{\mathcal{G}_j}}$, we can compute the similarity/distance between the two graphs.

Since GED is a well-defined graph distance metric, we can minimize the difference between the predicted distance and the ground-truth distance:

$$\mathcal{L} = \mathbb{E}_{(i,j)\sim\mathcal{D}}(\hat{d_{ij}} - d_{ij})^2 = \mathbb{E}_{(i,j)\sim\mathcal{D}}(||\boldsymbol{h_{\mathcal{G}_i}} - \boldsymbol{h_{\mathcal{G}_j}}||_2^2 - d_{ij})^2. \tag{3}$$

where (i, j) is a graph pair sampled from the training set and d_{ij} is the GED between them.

After training, the learned neural network can be appleid to any graph, and the graph-level embeddings can facilitate a series of downstream tasks, and can be fine-tuned for specific tasks. For

Table 1: Graph classification accuracy in percent. "-" indicates that the computation did not finish after 72 hours. We highlight the top 2 accuracy in bold.

Method	PTC	IMDBMULTI	WEB	NCI109	REDDIT12K
GK	57.26	43.89	21.37	62.06	31.82
DGK	57.32	44.55	23.65	62.69	32.22
SP	58.24	37.01	18.16	73.00	–
DSP	60.08	39.67	22.65	73.26	–
WL	66.97	49.33	26.44	**80.22**	39.03
DWL	70.17	49.95	34.56	**80.32**	39.21
GRAPH2VEC	60.17	47.33	**40.91**	74.26	35.24
NETMF	56.65	30.67	19.71	51.84	23.24
GRAPHSAGE	52.17	34.67	20.38	65.09	25.01
UGRAPHEMB	**72.54**	**50.06**	37.36	69.17	**39.97**
UGRAPHEMB-F	**73.56**	**50.97**	**45.03**	74.48	**41.84**

Table 2: Similarity ranking performance. BEAM, HUNGARIAN, and VJ are three approximate GED computation algorithms returning upper bounds of exact GEDs. We take the minimum GED computed by the three as ground-truth GEDs for training and evaluating all the methods on both Task 1 and 2. Their results are labeled with "$*$". HED is another GED solver yielding lower bounds. "-" indicates that the computation did not finish after 72 hours.

	PTC		IMDBMULTI		WEB		NCI109		REDDIT12K	
Method	τ	p@10	τ	p@10	τ	p@10	τ	p@10	τ	p@10
GK	0.291	0.135	0.329	0.421	0.147	0.101	0.445	0.012	0.007	0.009
DGK	0.222	0.103	0.304	0.410	0.126	0.009	0.441	0.010	0.011	0.012
SP	0.335	0.129	0.009	0.011	0.008	0.065	0.238	0.012	–	–
DSP	0.344	0.130	0.007	0.010	0.011	0.072	0.256	0.019	–	–
WL	0.129	0.074	0.034	0.038	0.014	0.246	0.042	0.006	0.089	0.017
DWL	0.131	0.072	0.039	0.041	0.017	0.262	0.049	0.009	0.095	0.023
GRAPH2VEC	0.128	0.188	0.697	0.624	0.014	0.068	0.033	0.127	0.008	0.017
NETMF	0.004	0.012	0.003	0.143	0.002	0.010	0.001	0.008	0.009	0.042
GRAPHSAGE	0.011	0.033	0.042	0.059	0.009	0.010	0.018	0.040	0.089	0.017
BEAM	0.992*	0.983*	0.892*	0.968*	0.963*	0.957*	0.615*	0.997*	0.657*	1.000*
HUNGARIAN	0.755*	0.465*	0.872*	0.825*	0.706*	0.160*	0.667*	0.164*	0.512*	0.808*
VJ	0.749*	0.403*	0.874*	0.815*	0.704*	0.151*	0.673*	0.097*	0.502*	0.867*
HED	0.788	0.433	0.627	0.801	**0.667**	0.291	0.199	0.174	0.199	0.237
UGRAPHEMB	**0.840**	**0.457**	**0.853**	**0.816**	0.618	**0.303**	**0.476**	**0.189**	**0.572**	**0.365**

example, for graph classification, a supervised loss function can be used to further enhance the performance.

3 EXPERIMENTS

We evaluate our model, UGRAPHEMB, against a number of state-of-the-art approaches designed for unsupervised node and graph embeddings on three tasks (see Task 3 in Appendix G), to answer the following questions:

- **Q1** How superb are the graph-level embeddings generated by UGRAPHEMB, when evaluated with downstream tasks including graph classification and similarity ranking?
- **Q2** Do the graph-level embeddings generated by UGRAPHEMB provide meaningful visualization for the graphs in a graph database?
- **Q3** Is the quality of the embeddings generated by UGRAPHEMB sensitive to choices of hyperparamters?

3.1 Task 1: Graph Classification

Intuitively, the higher the quality of the embeddings, the better the classification accuracy. Thus, we feed the graph-level embeddings generated by UGRAPHEMB and the baselines into a logistic regression classifier to evaluate the quality: (1) GRAPH KERNELS (GK, DGK, SP, DSP, WL, and DWL); (2) GRAPH2VEC [30]; (3) NETMF [35]; (4) GRAPHSAGE [16].

For GRAPH KERNELS, we also try using the kernel matrix and SVM classifier as it is the standard procedure outlined in [57], and report the better accuracy of the two. For (3) and (4), we try different averaging schemes on node embeddings to obtain the graph-level embeddings and report their best accuracy. Appendix F.1 gives more details on the experimental settings.

As shown in Table 1, UGRAPHEMB without fine-tuning, i.e. using only the unsupervised "inter-graph" information, can already achieve top 2 on 3 out of 5 datasets and demonstrates competitive accuracy on the other datasets. With fine-tuning (UGRAPHEMB-F),

our model can achieve the best result on 3 out of 5 datasets. Methods specifically designed for graph-level embeddings (Graph Kernels, Graph2Vec, and UGraphEmb) consistently outperform methods designed for node-level embeddings (NetMF and GraphSAGE), suggesting that *good node-level embeddings do* ***not*** *naturally imply good graph-level representations.*

3.2 Task 2: Similarity Ranking

For each dataset, we split it into training, validation, and testing sets by 6:2:2, and report the averaged *Mean Squared Error (mse), Kendall's Rank Correlation Coefficient (τ)* [20], and *Precision at 10 (p@10)* to test the ranking performance. Appendix F.2 shows more details on the setup.

Table 2 shows that UGraphEmb achieves state-of-the-art ranking performance under all settings except one. This should not be a surprise, because only UGraphEmb utilizes the ground-truth GED results collectively determined by Beam [31], Hungarian [39], and VJ [10]. UGraphEmb even outperforms HED [11], a state-of-the-art approximate GED computation algorithm, under most settings, further confirming its strong ability to generate proximity-preserving graph embeddings by learning from a specific graph proximity metric, which is GED in this case.

3.3 Parameter Sensitivity of UGraphEmb

We evaluate how the dimension of the graph-level embeddings and the percentage of graph pairs with ground-truth GEDs used to train the model can affect the results. We report the graph classification accuracy on ImdbMulti.

As can be seen in Figure 2, the performance becomes marginally better if larger dimensions are used. For the percentage of training pairs with ground-truth GEDs, the performance drops as less pairs are used. Note that the x-axis is in log-scale. When we only use 0.001% of all the training graph pairs (only 8 pairs with ground-truth GEDs), the performance is still better than many baseline methods, exhibiting impressive insensitivity to data sparsity.

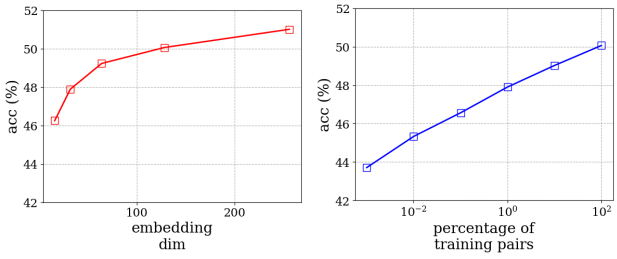

Figure 2: Classification accuracy on the ImdbMulti dataset w.r.t. the dimension of graph-level embeddings and the percentage of graph pairs used for training.

4 RELATED WORK

Unsupervised graph representation learning has a long history. Classic works including NetMF [35], LINE [47], DeepWalk [33], etc., which typically generate an embedding for each node in ***one*** graph. Theoretical analysis shows that many of these works cannot handle embeddings for multiple graphs in the sense that the node embeddings in one graph are not comparable to those in another graph in any straightforward way [18]. A simple permutation of node indices could cause the node embedding to be very different.

Table 3: A brief comparison of methods on node and graph representation learning. "G": Designed for graph-level, i.e. whole-graph, embeddings (✓) or not (×). "U": Unsupervised or supervised. "I": Inductive or transductive. For Graph Kernels, there are multiple citations.

Method	Citation	G	U	I
LLE	[2]	×	✓	×
GCN	[23]	×	×	✓
GIN	[54]	×	×	✓
DiffPool	[58]	✓	×	✓
Graph Kernels	-	✓	✓	✓
Graph2Vec	[30]	✓	✓	×
NetMF	[35]	×	✓	×
GraphSAGE	[16]	×	✓	✓
UGraphEmb	this paper	✓	✓	✓

More recently, some of the methods based on Graph Convolutional Networks (GCN) [6, 23], such as VGAE [24], satisfy the desired permutation-invariance property. Categorized as "graph autoencoders" [53], they also belong to the family of graph neural network methods. Although satisfying the permutation-invariance requirement, these autoencoders are still designed to generate unsuperised node embeddings.

Methods designed for unsupervised graph-level embeddings include Graph2Vec [30] and Graph Kernels [57], which however are either not based on learning or not inductive. Unlike node-level information which is reflected in the neighborhood of a node, graph-level information is much more limited. A large amount of graph neural network models resort to graph labels as a source of such information, making the models supervised aiming to improve graph classification accuracy specifically, such as DiffPool [58], CapsGNN [61], etc., while UGraphEmb learns a function that maps each graph into an embedding that can be used to facilitate many downstream tasks.

In Table 3, we can see that only UGraphEmb and Graph Kernels satisfy all the three properties. However, Graph Kernels cannot adapt to or learn from general graph proximity metrics like GED.

5 CONCLUSION

We present UGraphEmb, an end-to-end neural network based framework aiming to embed an entire graph into an embedding preserving the proximity between graphs in the dataset. A novel mechanism for generating graph-level embeddings is proposed. Experiments show that the produced graph-level embeddings achieve competitive performance on three downstream tasks.

ACKNOWLEDGEMENTS

This work is partially supported by NIH R01GM115833 and U01HG008488, NSF DBI-1565137, DGE-1829071, NSF III-1705169, NSF CAREER Award 1741634, and Amazon Research Award.

REFERENCES

[1] Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. 2019. SimGNN: A Neural Network Approach to Fast Graph Similarity Computation. *WSDM* (2019).

[2] Mikhail Belkin and Partha Niyogi. 2003. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation* 15, 6 (2003), 1373–1396.

[3] David B Blumenthal and Johann Gamper. 2018. On the exact computation of the graph edit distance. *Pattern Recognition Letters* (2018).

[4] H Bunke. 1983. What is the distance between graphs. *Bulletin of the EATCS* 20 (1983), 35–39.

[5] Horst Bunke and Kim Shearer. 1998. A graph distance metric based on the maximal common subgraph. *Pattern recognition letters* 19, 3-4 (1998), 255–259.

[6] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*. 3844–3852.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[8] Lun Du, Yun Wang, Guojie Song, Zhicong Lu, and Junshan Wang. 2018. Dynamic Network Embedding: An Extended Approach for Skip-gram based Network Embedding.. In *IJCAI*. 2086–2092.

[9] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *NIPS*. 2224–2232.

[10] Stefan Fankhauser, Kaspar Riesen, and Horst Bunke. 2011. Speeding up graph edit distance computation through fast bipartite matching. In *International Workshop on Graph-Based Representations in Pattern Recognition*. Springer, 102–111.

[11] Andreas Fischer, Ching Y Suen, Volkmar Frinken, Kaspar Riesen, and Horst Bunke. 2015. Approximation of graph edit distance based on Hausdorff matching. *Pattern Recognition* 48, 2 (2015), 331–343.

[12] Paolo Frasconi, Marco Gori, and Alessandro Sperduti. 1998. A general framework for adaptive processing of data structures. *IEEE transactions on Neural Networks* 9, 5 (1998), 768–786.

[13] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *ICML*. JMLR. org, 1263–1272.

[14] Sanjay Surendranath Girija. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. (2016).

[15] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *SIGKDD*. ACM, 855–864.

[16] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS*. 1024–1034.

[17] Peter E Hart, Nils J Nilsson, and Bertram Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics* 4, 2 (1968), 100–107.

[18] Mark Heimann and Danai Koutra. 2017. On generalizing neural node embedding methods to multi-network problems. In *KDD MLG Workshop*.

[19] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. 2016. Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design* 30, 8 (2016), 595–608.

[20] Maurice G Kendall. 1938. A new measure of rank correlation. *Biometrika* 30, 1/2 (1938), 81–93.

[21] Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *ICLR* (2015).

[22] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. 2018. Neural relational inference for interacting systems. *ICML* (2018).

[23] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *ICLR* (2016).

[24] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. *NIPS Workshop on Bayesian Deep Learning* (2016).

[25] Sofia Ira Ktena, Sarah Parisot, Enzo Ferrante, Martin Rajchl, Matthew Lee, Ben Glocker, and Daniel Rueckert. 2017. Distance metric learning using graph convolutional networks: Application to functional brain networks. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 469–477.

[26] Yongjiang Liang and Peixiang Zhao. 2017. Similarity search in graph databases: A multi-layered indexing approach. In *ICDE*. IEEE, 783–794.

[27] Tengfei Ma, Cao Xiao, Jiayu Zhou, and Fei Wang. 2018. Drug Similarity Integration Through Attentive Multi-view Graph Auto-Encoders. *IJCAI* (2018).

[28] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, Nov (2008), 2579–2605.

[29] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*. 3111–3119.

[30] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. 2017. graph2vec: Learning distributed representations of graphs. *KDD MLG Workshop* (2017).

[31] Michel Neuhaus, Kaspar Riesen, and Horst Bunke. 2006. Fast suboptimal algorithms for the computation of graph edit distance. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. Springer, 163–172.

[32] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning convolutional neural networks for graphs. In *ICML*. 2014–2023.

[33] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *SIGKDD*. ACM, 701–710.

[34] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *NAACL* (2018).

[35] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network Embedding as Matrix Factorization: UnifyingDeepWalk, LINE, PTE, and node2vec. *WSDM* (2018).

[36] Rashid Jalal Qureshi, Jean-Yves Ramel, and Hubert Cardot. 2007. Graph based shapes representation and recognition. In *International Workshop on Graph-Based Representations in Pattern Recognition*. Springer, 49–60.

[37] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. (2018).

[38] Kaspar Riesen and Horst Bunke. 2008. IAM graph database repository for graph based pattern recognition and machine learning. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. Springer, 287–297.

[39] Kaspar Riesen and Horst Bunke. 2009. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision computing* 27, 7 (2009), 950–959.

[40] Kaspar Riesen, Sandro Emmenegger, and Horst Bunke. 2013. A novel software toolkit for graph edit distance computation. In *International Workshop on Graph-Based Representations in Pattern Recognition*. Springer, 142–151.

[41] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The graph neural network model. *IEEE Transactions on Neural Networks* 20, 1 (2009), 61–80.

[42] Nino Shervashidze and Karsten Borgwardt. 2009. Fast subtree kernels on graphs. In *NIPS*. 1660–1668.

[43] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. 2011. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research* 12, Sep (2011), 2539–2561.

[44] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. 2009. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*. 488–495.

[45] Anshumali Shrivastava and Ping Li. 2014. A new space for comparing graphs. In *Proceedings of the 2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. IEEE Press, 62–71.

[46] Martin Simonovsky and Nikos Komodakis. 2017. Dynamic edgeconditioned filters in convolutional neural networks on graphs. In *Proc. CVPR*.

[47] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW*. International World Wide Web Conferences Steering Committee, 1067–1077.

[48] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. *ICLR* (2018).

[49] Nikil Wale, Ian A Watson, and George Karypis. 2008. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems* 14, 3 (2008), 347–375.

[50] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *SIGKDD*. ACM, 1225–1234.

[51] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. 2017. Community Preserving Network Embedding.. In *AAAI*. 203–209.

[52] Christopher KI Williams. 2001. On a connection between kernel PCA and metric multidimensional scaling. In *Advances in neural information processing systems*. 675–681.

[53] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. 2019. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596* (2019).

[54] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks? *ICLR* (2019).

[55] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation learning on graphs with jumping knowledge networks. *ICML* (2018).

[56] Xifeng Yan, Philip S Yu, and Jiawei Han. 2005. Substructure similarity search in graph databases. In *SIGMOD*. ACM, 766–777.

[57] Pinar Yanardag and SVN Vishwanathan. 2015. Deep graph kernels. In *SIGKDD*. ACM, 1365–1374.

[58] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. 2018. Hierarchical Graph Representation Learning with Differentiable Pooling. *NeurIPS* (2018).

[59] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. In *NeurIPS*. 5171–5181.

[60] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An end-to-end deep learning architecture for graph classification. In *AAAI*.

[61] Xinyi Zhang and Lihui Chen. 2019. Capsule Graph Neural Network. *ICLR* (2019).

[62] Xiaohan Zhao, Bo Zong, Ziyu Guan, Kai Zhang, and Wei Zhao. 2018. Substructure Assembling Network for Graph Classification. *AAAI* (2018).

APPENDICES

A COMPARISON WITH EXISTING FRAMEWORKS

To better see the novelty of our proposed framework, UGraphEmb, we present a detailed study on two related existing frameworks for node and graph embeddings. As shown in Figure 3, we summarize graph neural network architectures for learning graph representations into three frameworks:

- ***Framework 1***: Supervised/Unsupervised framework for node-level tasks, e.g. node classification, link prediction, etc.
- ***Framework 2***: Supervised end-to-end neural networks for graph-level tasks, typically graph classification.
- ***Framework 3***: UGraphEmb, unsupervised framework for multiple graph-level tasks with the key novelty in using graph-graph proximity.

The rest of this section describes the first two frameworks in detail, and compare UGraphEmb with various other related methods when appropriate. This section also serves as a more thorough survey of graph embedding methods, proving more background knowledge in the area of node and graph representation learning.

A.1 Framework 1: Node Embedding (Supervised and Unsupervised)

Since the goal is to perform node-level tasks, the key is the "Node Embedding Model" which produces one embedding per node for the input graph. As described in the main paper, there are many methods to obtain such node embeddings, such as:

- ***Matrix Factorization***:
  This category includes a vast amount of both early and recent works on network (node) embedding, such as Laplacian Eigenmaps (LLE) [2], M-NMF [51], NetMF [35], etc.
  Many interesting insights and theoretical analysis have been discovered and presented, but since this work focuses on neural network based methods, the reader is referred to [35] for a complete discussion.
- ***Direct Encoding (Free Encoder)***:
  This simple way of directly initializing one embedding per node randomly can be traced back to the Natural Language Processing domain – the classic Word2Vec [29] model indeed randomly initializes one embedding per word where gradients flow back during optimization.
  Node embedding methods such as LINE [47] ans DeepWalk [33] adopt this scheme. DeepWalk is also known as "skip-gram based methods" [8] due to its use of Word2Vec.
  However, such methods are intrinsically transductive – they cannot handle new node unseen in the training set in a straightforward way [16]. For Word2Vec though, it is typically not a concern since out-of-vocabulary words tend to be rare in a large text corpus.
  This also reveals a fundamental difference between the text domain and graph domain – words have their specific semantic meaning making them identifiable across different documents, yet nodes in a graph or graphs usually lack such identity. This calls for the need of ***inductive representation learning***, which is addressed more recently by the next type of node embedding model.
- ***Graph Convolution (Neighbor Aggregation)***:
  As discussed in the main paper, Graph Convolutional Network (GCN) [6] boils down to the aggregation operation that is applied to every node in a graph. This essentially allows the neural network model to ***learn a function that maps input graph to output node embeddings***:

$$\phi(\mathcal{G}) = \phi(\boldsymbol{A}_{\mathcal{G}}, \boldsymbol{F}_{\mathcal{G}}) = \boldsymbol{U}_{\mathcal{G}}. \tag{4}$$

  where $\boldsymbol{A}_{\mathcal{G}}$ and $\boldsymbol{F}_{\mathcal{G}}$ denote the adjacency matrix (link structure) and the node and/or edge features/attributes, and $\boldsymbol{U}_{\mathcal{G}} \in \mathbb{R}^{N \times D}$ is the node embedding matrix.
  The importance of such function ϕ is evident – for any new node i outside the training set of nodes, the neighbor aggregation models can simply apply the learned ϕ to obtain $\boldsymbol{u}_i$; for any new graph outside the training set of graphs, the same procedure also works, achieving ***inductivity***. ***Permutation-invariance*** can also be achieved as discussed in the main paper.
  Methods including GraphSAGE [16], GAT [48], GIN [54], etc. are all under this category, with different aggregators proposed.

So far we have discussed about the node embedding generation step. In order to make the node embeddings high-quality, additional components are usually necessary as auxiliaries/guidance in the architectures of methods belong to Framework 1, including:

- ***Predict Node Context***:
  The goal is to use the node embeddings to reconstruct certain "node local context" – in other words, to force the node embeddings to preserver certain local structure. We highlight three popular types of definitions of such context:
  - ***1st order neighbor***:
    The model encourages directly connected nodes to have similar embeddings. In LINE-1st, the loss function is similar to the Skip-gram objective proposed in Word2Vec. In SDNE [50], an auto-encoder framework, the loss function is phrased as the reconstruction loss, the typical name in auto-encoders.
  - ***Higher order context***:
    An example is LINE-2nd, which assumes that nodes sharing many connections to other nodes are similar to each other. In practice, such incorporation of higher order neighbors typically gives better performance in node-level tasks.
  - ***Random walk context***:
    "Context nodes" are defined as the nodes that co-occur on random walks on a graph. By this definition, for a given node, its context can include both its close-by neighbors and distant node. Equipped with various techniques of tuning and improving upon random walks, this type of methods seems promising.
    Example methods include DeepWalk, Node2Vec [15], GraphSAGE, etc. Notice that the former two use direct encoding as its node embedding model as described previously, while GraphSAGE uses neighbor aggregation. From

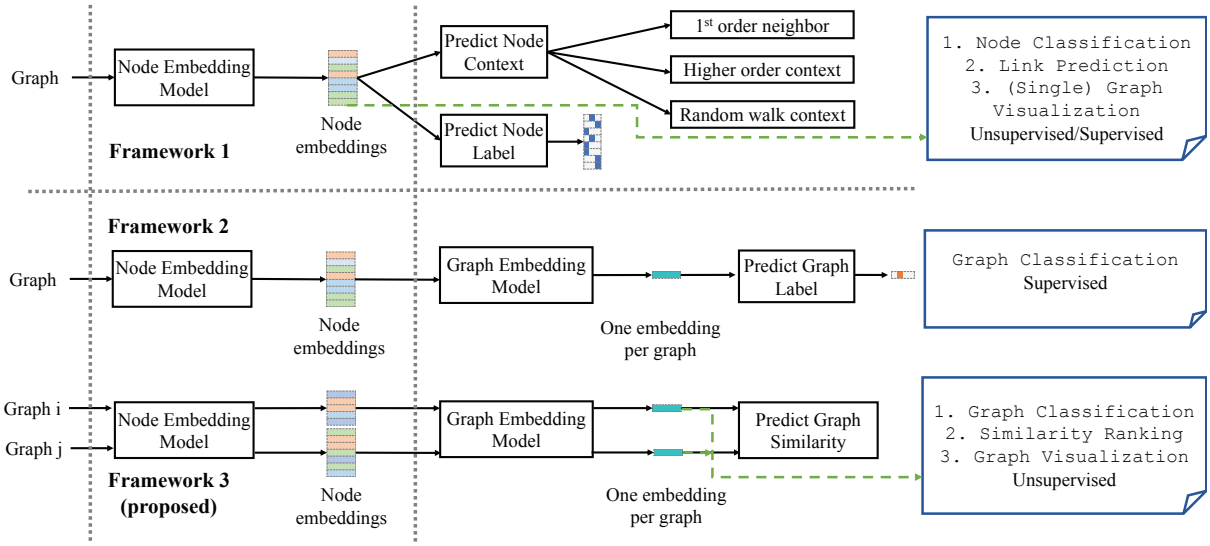

Figure 3: Architecture 1 and 2 are typical neural network architectures for graph representation learning. Architecture 3 is our proposed UGraphEmb.

this, we can also see that Framework 1 indeed includes a vast amount of models and architectures.

- ***Predict Node Label***:
  So far all the methods we have discussed about are unsupervised node embedding methods. As said in the main paper, to evaluate these unsupervised node embeddings, a second stage is needed, which can be viewed as a series of downstream tasks as listed in Figure 3.
  However, a large amount of existing works incorporate a supervised loss function into the model, making the entire model trainable end-to-end.
  Examples include GCN as in [24], as well as a series of improvements upon GCN, such as GraphSAGE [16], GAT [48], GIN [54], etc. as mentioned in the main paper.

Before finishing presenting Framework 1, we highlight important distinctions between the proposed framework and the following baseline methods:

A.1.1 UGraphEmb vs NetMF. NetMF is among the state-of-the-art matrix factorization based methods for node embeddings. It performs eigen-decomposition, one-side bounding, and rank-d approximation by Singular Value Decomposition, etc. for a graph, and is transductive. UGraphEmb is graph-level and inductive. Section F.1.2 gives more details on how we obtain graph-level embeddings out of node-level embeddings for NetMF.

A.1.2 UGraphEmb vs GraphSAGE. GraphSAGE belongs to neighbor aggregation based methods. Although being unsupervised and inductive, by design GraphSAGE performs node-level embeddings via an unsupervised loss based on context nodes on random walks (denoted as ***"Random walk context"*** as in Figure 3), while UGraphEmb performs graph-level embeddings via the MSNA mechanism, capturing structural difference at multiple scales and adaptive to a given graph similarity/distance metric.

A.2 Framework 2: Supervised Graph Embedding

The second framework we identify as supervised graph embedding. So far graph classification is the dominating and perhaps the only important task for Frmaework 2.

Here we highlight some existing works to demonstrate its popularity, including PatchySan [32], ECC [46], Set2Set [13], GraphSAGE, DGCNN/SortPool [60], SAN [62], DiffPool [58], CapsGNN [61], etc.

Notice that most of these models adopt the neighbor aggregation based node embedding methods described previously, which are inductive so that for new graphs outside the training set of graphs, their graph-level embeddings can be generated, ans graph classification can be performed.

A.2.1 UGraphEmb vs Graph Kernels. Although Graph Kernels [57] are not supervised methods, we still make a comparison here, because Graph Kernels are a family of methods designed for graph classification, the same task as Framework 2.

Different graph kernels extract different types of substructures in a graph, e.g. graphlets [44], subtree patterns [42], etc., and the resulting vector representation for each graph is typically called "feature vector" [57], encoding the count/frequency of substructures. These feature vectors are analogous to graph-level embeddings, but the end goal is to create a kernel matrix encoding the similarity between all the graph pairs in the dataset fed into a kerkenl SVM classifier for graph classification.

Compared with graph kernels, UGraphEmb ***learns*** a function that preserves a general graph similarity/distance metric such as Graph Edit Distance (GED), and as a result, yields a graph-level embedding for each graph that can be used to facilitate a series of downstream tasks. It is inductive, i.e. handles unseen graphs due to the learned function. In contrast, although Graph Kernels can be considered as inductive [43], graph kernels have to perform the

subgraph extraction for every graph, which can be slow and cannot adapt to different graph proximity metrics.

A.2.2 UGraphEmb vs Graph2Vec. Similar to DeepWalk, Graph2Vec is also inspired by the classic Word2Vec paper, but instead of generating node embeddings, it is designed to generate graph-level embeddings, by treating each graph as a bag of rooted subgraphs, and adopting Doc2Vec [29] instead of Word2Vec. The difference between Graph2Vec and UGraphEmb is that, Graph2Vec is transductive (similar to Graph Kernels), as explained in Section A.1, while UGraphEmb is inductive.

A.3 Framework 3: UGraphEmb

This is our proposed framework, which is the key novelty of the paper. Now since we have introduced Framework 1 and Framework 2, it can be clearly seen that the use of graph-graph proximity is a very different and new perspective of performing graph-level embeddings. UGraphEmb satifies all the following properties: ***graph-level***, ***unsupervised***, and ***inductive***.

B GRAPH-LEVEL EMBEDDING GENERATION

B.1 A Brief Survey

Across the years, many methods have been proposed for the generation of graph-level embeddings from node-level embeddings, with the most popular ones including:

- ***Sum/Average***:
  The graph embedding is simply the summation or average of the node embeddings [9].
- ***Supersource***:
  Dated back to early works [12, 41], the idea is to introduce a "dummy/super node" connecting to every node in the graph, so that during aggregation, this supersource node absorbs information from all the nodes. Its embedding is treated as the graph-level embedding.
  The edge between the supersource node and every other node is directed, so that information flows from other nodes to the supersource node. Otherwise, two far-away nodes would affect each other due to the supersource node.
- ***Coarsening/Pooling***:
  Graph coarsening/pooling reduces the original graph into smaller and smaller graphs by clustering nodes/subgraphs recursively into a hierarchy [6, 58].
- ***Others***:
  Other methods include Capsule Graph Neural Network [61],aggregation using "fuzzy" histograms [19], Convolutional Neural Networks [32], etc.

B.2 Design of Whole-Graph Embedding Generation Model

However, since our goal is to embed each graph as a single point in the embedding space that preserves graph-graph proximity, the graph embedding generation model should:

- ***Capture structural difference at multiple scales***.
  Applying a neighbor aggregation layer on nodes such as GIN cause the information to flow from a node to its direct neighbors, so sequentially stacking K layers would cause the final representation of a node to include information from its K-th order neighbors.
  However, after many neighbor aggregation layers, the learned embeddings could be too coarse to capture the structural difference in small local regions between two similar graphs. Capturing structural difference at multiple scales is therefore important for UGraphEmb to generate high-quality graph-level embeddings.
- ***Be adaptive to different graph proximity metrics***.
  UGraphEmb is a general framework that should be able to preserve the graph-graph proximity under any graph proximity metric, such as GED and MCS. A simple aggregation of node embeddings without any learnable parameters limits the expressive power of existing graph-level embedding models.

These existing methods are based on aggregation of node embeddings without flexibility/learnability to adapt to different graph similarity/distance metrics, and lack the capability to capture structural difference at multiple scales. We therefore propose a **Multi-Scale Node Attention (MSNA)** mechanism to address both issues for UGraphEmb.

C GRAPH PROXIMITY METRICS

C.1 Graph Proximity Metric Selection

Since the key novelty of UGraphEmb is the use of graph-graph proximity, it is important to select an appropriate graph proximity (similarity/distance) metric. We identify three categories of candidates:

- ***Proximity defined by graph labels***.
  For graphs that come with labels, we may treat graphs of the same label to be similar to each other. However, such proximity metric may be too coarse, unable to distinguish between graphs of the same label.
- ***Proximity given by domain knowledge or human experts***.
  For example, in drug-drug interaction detection [27], a domain-specific metric to encode compound chemical structure can be used to compute the similarities between chemical graphs. However, such metrics do not generalize to graphs in other domains. Sometimes, this information may be very expensive to obtain. For example, to measure brain network similarities, a domain-specific preprocessing pipeline involving skull striping, band-pass filtering, etc. is needed. The final dataset only contains networks from 871 humans [25].
- ***Proximity defined by domain-agnostic and well-accepted metrics***.
  Metrics such as Graph Edit Distance (GED) [4] and Maximum Common Subgraph (MCS) [5] have been widely adopted in graph database search [26, 56], are well-defined and general to any domain.

In this paper, we use GED as an example metric to demonstrate UGraphEmb.

C.2 Graph Edit Distance (GED)

The edit distance between two graphs [4] $\mathcal{G}_1$ and $\mathcal{G}_2$ is the number of edit operations in the optimal alignments that transform $\mathcal{G}_1$ into $\mathcal{G}_2$, where an edit operation on a graph $\mathcal{G}$ is an insertion or deletion of a node/edge or relabelling of a node. Note that other variants of GED definitions exist [40], and we adopt the most basic version in this work. Fig. 4 shows an example of GED between two simple graphs.

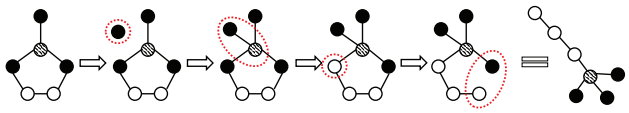

Figure 4: The GED between the graph to the left and the graph to the right is 4, involving the following edit operation sequence: A node addition, an edge addition, a node label substitution, and an edge deletion.

Notice that although UGraphEmb currently does not handle edge types, UGraphEmb is a general framework and can be extended to handle edge types, e.g. adapting the graph neural network described in [22].

C.3 Graph Proximity Metric and Loss Function

Multidimensional scaling (MDS) is a classic form of dimensionality reduction [52]. The idea is to embed data points in a low dimensional space so that their pairwise distances are preserved, e.g. via minimizing the loss function

$$\mathcal{L}(\boldsymbol{h_i}, \boldsymbol{h_j}, d_{ij}) = (||\boldsymbol{h_i} - \boldsymbol{h_j}||_2^2 - d_{ij})^2 \tag{5}$$

where $\boldsymbol{h_i}$ and $\boldsymbol{h_j}$ are the embeddings of points i and j, and d_{ij} is their distance.

Denote the whole-graph embeddings for $\mathcal{G}_i$ and $\mathcal{G}_j$ as $\boldsymbol{h}_{\mathcal{G}_i}$ and $\boldsymbol{h}_{\mathcal{G}_j}$. Since GED is a well-defined graph distance metric, we can minimize the difference between the predicted distance and the ground-truth distance:

$$\mathcal{L} = \mathbb{E}_{(i,j)\sim\mathcal{D}}(\hat{d_{ij}} - d_{ij})^2 = \mathbb{E}_{(i,j)\sim\mathcal{D}}(||\boldsymbol{h}_{\mathcal{G}_i} - \boldsymbol{h}_{\mathcal{G}_j}||_2^2 - d_{ij})^2. \tag{6}$$

where (i, j) is a graph pair sampled from the training set and d_{ij} is the GED between them.

Alternatively, if the graph proximity metric is about similarity, such as in the case of MCS, we can use the following loss function:

$$\mathcal{L} = \mathbb{E}_{(i,j)\sim\mathcal{D}}(\hat{s_{ij}} - s_{ij})^2 = \mathbb{E}_{(i,j)\sim\mathcal{D}}(\boldsymbol{h}_{\mathcal{G}_i}^T \boldsymbol{h}_{\mathcal{G}_j} - s_{ij})^2. \tag{7}$$

D DATASETS

D.1 Detailed Description of Datasets

Five real graph datasets are used for the experiments. A concise summary can be found in Table 4.

- Ptc [45] is a collection of 344 chemical compounds which report the carcinogenicity for rats. There are 19 node labels for each node.
- ImdbMulti [57] consists of 1500 ego-networks of movie actors/actresses with unlabeled nodes representing the people and edges representing the collaboration relationship. The nodes are unlabeled, but there could be 3 graph labels for each graph.
- Web [38] is a collection of 2340 documents from 20 categories. Each node is a word, and there is an edge between two words if one word precedes the other. Since one word can appear in multiple sentences, the entire document is represented as a graph. Only the most frequent words are used to construct the graph, and there are 15507 words in total, thus 15507 node types associated with the dataset.
- Nci109 [49] is another bioinformatics dataset. It contains 4127 chemical compounds tested for their ability to suppress or inhibit human tumor cell growth.
- Reddit12k [57] contains 11929 graphs each corresponding to an online discussion thread where nodes represent users, and an edge represents the fact that one of the two users responded to the comment of the other user. There is 1 of 11 graph labels associated with each of these 11929 discussion graphs, representing the category of the community.

D.2 Additional Notes on Web

Since each graph node in Web represents a word, it is natural to consider incorporating the semantic similarity between two words, e.g. using Word2Vec, into the GED definition, and even the broader topic of text matching and retrieval.

In fact, the definition of GED does not specify that node labels must be discrete. There exists some variant of GED definition that can define node label difference in a more complicated way [40], which is a promising direction to explore in future. It is also promising to explore document embedding based on graph representation of text.

E DATA PREPROCESSING

For each dataset, we randomly split 60%, 20%, and 20% of all the graphs as training set, validation set, and testing set, respectively. For each graph in the testing set, we treat it as a query graph, and let the model compute the distance between the query graph and every graph in the training and validation sets.

E.1 Ground-Truth GED Computation

To compute ground-truth GED for training pair generation as well as similarity ranking evaluation, we have the following candidate GED computation algorithms:

- A* [17]:
  It is an exact GED solver, but due to the NP-hard nature of GED, it runs in exponential time complexity. What is worse, a recent study shows that no currently available algorithm can reliably compute GED within reasonable time between graphs with more than 16 nodes [3].
- Beam [31], Hungarian [39], and VJ [10]:
  They are approximate GED computation algorithms with sub-exponential time complexity, quadratic time complexity, and quadratic time complexity, respectively. They are all guaranteed to return upper bounds of the exact GEDs, i.e. their computed GEDs are always greater than or equal to the actual exact GEDs.
- HED [11]:

Table 4: Statistics of datasets. "Min", "Max", "Mean", and "Std" refer to the minimum, maximum, mean, and standard deviation of the graph sizes (number of nodes), respectively.

Dataset	Meaning	#Node Labels	#Graphs	#Graph Labels	Min	Max	Mean	Std
PTC	Chemical Compounds	19	344	2	2	109	25.5	16.2
IMDBMULTI	Social Networks	1	1500	3	7	89	13.0	8.5
WEB	Text Documents	15507	2340	20	3	404	35.5	37.0
NCI109	Chemical Compounds	38	4127	2	4	106	29.6	13.5
REDDIT12K	Social Networks	1	11929	11	2	3782	391.4	428.7

It is another approximate GED solver running in quadratic time, but instead yields lower bounds of exact GEDs.

We take the minimum distance computed by BEAM [31], HUNGARIAN [39], and VJ [10]. The minimum is taken because their returned GEDs are guaranteed to be upper bounds of the true GEDs. In fact, the ICPR 2016 Graph Distance Contest [2] also adopts this approach to handle large graphs.

We normalize the GEDs according to the following formula: $\text{nGED}(\mathcal{G}_1, \mathcal{G}_2) = \frac{\text{GED}(\mathcal{G}_1, \mathcal{G}_2)}{(|\mathcal{G}_1|+|\mathcal{G}_2|)/2}$, where $|\mathcal{G}_i|$ denotes the number of nodes of $\mathcal{G}_i$ [36].

For the smaller datasets PTC, IMDBMULTI, and WEB, we compute the ground-truth GEDs for all the pairs in the training set. For the larger datasets NCI109 and REDDIT12K, we do not compute all the pairs, and instead cap the computation at around 10 hours.

We run the ground-truth GED solvers on a CPU server with 32 cores, and utilize at most 20 cores at the same time, using code from [40]. The details are shown in Table 5.

Table 5: Number of graph pairs used to train UGRAPHEMB on each dataset, along with the total wall time to compute the ground-truth GEDs for these pairs.

Dataset	#Total Pairs	#Comp. Pairs	Time
PTC	118336	42436	9.23 Mins
IMDBMULTI	2250000	810000	4.72 Hours
WEB	5475600	1971216	8.23 Hours
NCI109	17032129	2084272	10.05 Hours
REDDIT12K	142301041	2124992	10.42 Hours

E.2 "Hyper-Level" Graph

At this point, it is worth mentioning that the training procedure of UGRAPHEMB is stochastic, i.e. UGRAPHEMB is trained on a subset of graph pairs in each iteration. Moreover, UGRAPHEMB does not require the computation all the graph pairs in the training set, so the notion of "hyper-level" graph as mentioned in the main paper does *not* imply that UGRAPHEMB constructs a fully connected graph where each node is a graph in the dataset.

In future, it would be promising to explore other techniques to construct such "hyper-level" graph beyond the current way of random selection of graph pairs in the training set.

E.3 Node Label Encoding

For PTC, WEB, and NCI109, the original node representations are one-hot encoded according to the node labels. For graphs with unlabeled nodes, i.e., IMDBMULTI and REDDIT12K, we treat every node to have the same label, resulting in the same constant number as the initialize representation.

In future, it would be interesting to consider more sophisticated ways to encode these node labels, because node labels help identifying different nodes across different graph datasets, which is an important component for a successful pre-training method for graphs. Consider that we want to combine multiple different graph datasets of different domains for large-scale pre-training of graph neural networks. Then how to handle different node labels in different datasets and domains becomes an important issue.

F PARAMETER SETTINGS AND EXPERIMENTAL DETAILS

For the proposed model, to make a fair comparison with baselines, we use a single network architecture on all the datasets, and run the model using exactly the same test graphs as used in the baselines.

We set the number of GIN layers to 3, and use ReLU as the activation function. The output dimensions for the 1st, 2nd, and 3rd layers of GIN are 256, 128, and 64, respectively. Following the original paper of GIN [54], we fix ϵ to 0.

Then we transform the concreted embeddings into graph-level embeddings of dimension 256 by using two fully connected (dense) layers, which are denoted as MLP in the main paper.

The model is written in TensorFlow [14]. We conduct all the experiments on a single machine with an Intel i7-6800K CPU and one Nvidia Titan GPU. As for training, we set the batch size to 256, i.e. 256 graph pairs (512 graphs) per mini-batch, use the Adam algorithm for optimization [21], and fix the initial learning rate to 0.001. We set the number of iterations to 20000, and select the best model based on the lowest validation loss.

F.1 Task 1: Graph Classification

F.1.1 Evaluation Procedure. Since UGRAPHEMB is unsupervised, we evaluate all the methods following the standard strategy for evluating unsupervised node embeddings [47, 50]. It has three stages: (1) Train a model using the training set with validation set for parameter tuning; (2) Train a standard logistic regression classifier using the embeddings as features as well as their ground-truth graph labels; (3) Run the model on the graphs in the testing set and feed their embeddings into the classifier for label prediction.

F.1.2 Baseline Setup. By default, we use the results reported in the original work for baseline comparison. However, in cases where the results are not available, we use the code released by the original authors, performing a hyperparameter search based on

[2] https://gdc2016.greyc.fr/

the original author's guidelines. Notice that our baselines include a variety of methods of different flavors:

- Graph Kernels:
  For the Graph Kernels baselines, there are two schemes to evaluate: (1) Treat the features extracted by each kernel method as the graph-level embeddings, and perform the second and third stages described previously; (2) Feed the SVM kernels generated by each method into a kernel SVM classifier as in [57]. The second scheme typically yields better accuracy and is more typical. We perform both schemes and report the better of the two accuracy scores for each baseline kernel. All the six versions of the Graph Kernels are described in detail in [57].
- Graph2Vec:
  Similar to Graph Kernels, Graph2Vec is also transductive, meaning it has to see all the graphs in both the training set and the testing set, and generates a graph-level embedding for each.
- NetMF and GraphSAGE:
  Since they generate node-level embeddings, we take the average of node embeddings as the graph-level embedding. We also try various types of averaging schemes, including weighted by the node degree, weighted by the inverse of node degree, as well as summation. We report the best accuracy achieved by these schemes.
  There is no training needed to be done for NetMF, since it is based on matrix factorization. For GraphSAGE, we combine all the graphs in the training set, resulting in one single graph to train GraphSAGE, which is consistent with its original design for inductive node-level embeddings. After training, we use the trained GraphSAGE model to generate graph-level embeddings for each individual graph in the test set, consistent with how UGraphEmb handles graphs in the test set.

F.1.3 Embedding Dimension. For all the baseline methods, we ensure that the dimension of the graph-level embeddings is the same as our UGraphEmb by setting hyperparameters for each properly. For Graph Kernels, however, they extract and count subgraphs, and for a given dataset, the number of unique subgraphs extracted depend on the dataset, which determines the dimension of the feature vector for each graph in the dataset. Thus, we do not limit the number of subgraphs they extract, giving them advantage, and follow the guidelines in their original papers for hyperparameter tuning.

F.1.4 Fine-Tuning. To incorporate the supervised loss function (cross-entropy loss) into our model, we use multiple fully connected layers to reduce the dimension of graph-level embeddings to the number of graph labels. When the fine-tuning procedure starts, we switch to using the supervised loss function to train the model with the same learning rate and batch size as before.

After fine-tuning, the graph label information is integrated into the graph-level embeddings. We still feed the embeddings into the logistic regression classifier for evaluation to ensure it is consistent for all the configurations of all the models. The accuracy based on the prediction of the model is typically much higher because it utilizes supervised information for graph label prediction.

F.2 Task 2: Similarity Ranking

F.2.1 Evaluation Procedure. For all the methods, we adopt the procedure outlined in Section F.1.2 to obtain graph-level embeddings. For each graph in the test set, we treat it as a graph query, compute the similarity/distance score between the query graph and every graph in the training set, and rank the results, compared with the ground-truth ranking results by the ground-truth GED solvers.

We compute both the similarity score (inner product of two graph-level embeddings) and distance score (squared L-2 distance between two graph-level embeddings) for every graph pair when doing the query, and report the better of the two in the paper. To verify that the actual ranking of the graphs makes sense, we perform several case studies. As shown in Figure 5, UGraphEmb computes the distance score between the query and every graph in the training set. Although the exact distance score is not exactly the same as the ground-truth normalized GEDs, the relatively position and ranking are quite reasonable.

Notice that for Graph Kernels, the three deep versions, i.e. DGK, DSP, and WDL, generate the same graph-level embeddings as the non-deep versions, i.e. GK, SP, and DL, but use the idea of Word2Vec to model the relation between subgraphs [57]. Consequently, the non-deep versions simply compute the dot products between embeddings to generate the kernel matrices, but the deep versions further modify the kernel matrices, resulting in different graph-graph similarity scores. We thus evaluate the deep versions using their modified kernel matrices.

G TASK 3: EMBEDDING VISUALIZATION

Visualizing the embeddings on a two-dimensional space is a popular way to evaluate node embedding methods [47]. However, we are among the first to investigate the question: Are the graph-level embeddings generated by a model like UGraphEmb provide meaningful visualization?

We feed the graph emebddings learned by all the methods into the visualization tool t-SNE [28]. The three deep graph kernels, i.e. DGK, DSP, and WDL, generate the same embeddings as the non-deep versions, but use additional techniques [57] to modify the similarity kernel matrices, resulting in different classification and ranking performance.

From Figure 6, we can see that UGraphEmb can separate the graphs in ImdbMulti into multiple clusters, where graphs in each cluster share some common substructures.

Such clustering effect is likely due to our use of graph-graph proximity scores, and is thus not observed in NetMF or GraphSAGE. For Graph Kernels and Graph2Vec though, there are indeed clustering effects, but by examining the actual graphs, we can see that graph-graph proximity is not well-preserved by their clusters (e.g. for WL graph 1, 2 and 9 should be close to each other; for Graph2Vec, graph 1, 2, and 12 should be close to each other), explaining their worse similarity ranking performance compared to UGraphEmb.

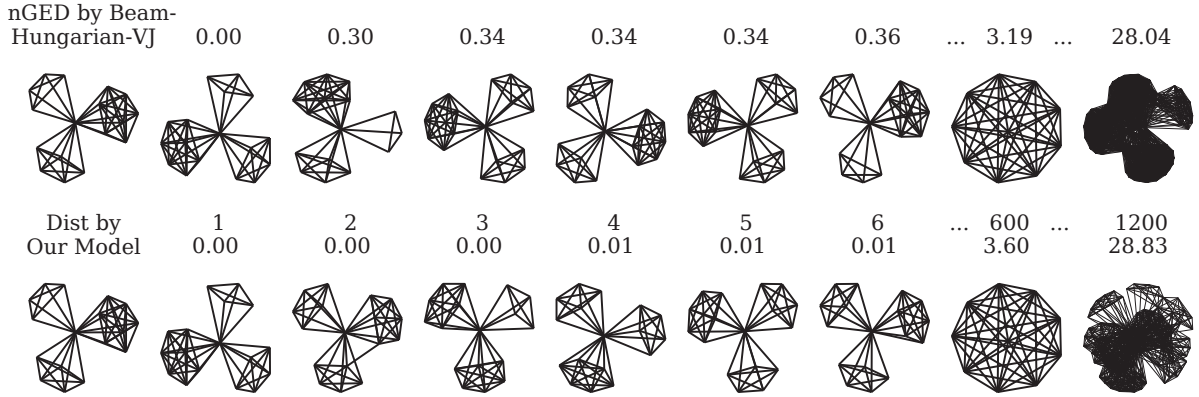

Figure 5: Visualization of the ranking results on a query in IMDBMULTI. The top row depicts the ground-truth ranking, labeled with normalized ground-truth GEDs, and the bottom row depicts ranking by UGRAPHEMB, depicts the distance score computed by UGRAPHEMB.

G.1 Detailed Evaluation Procedure

We feed the graph-level embeddings into the t-SNE [28] tool to project them into a 2-D space. We then do the following linear interpolation: (1) Select two points in the 2-D space; (2) Form a line segment between the two selected points; (3) Split the line into 11 equal-length line segments, resulting in 12 points on the line segment in total; (4) Go through these 12 points: For each point, find an embedding point in the 2-D space that is closest to it; (5) Label the 12 embedding points on the embedding plot and draw the actual graph to the right of the embedding plot. This yields the visualization of the IMDBMULTI dataset.

H ANALYSIS AND DISCUSSION OF EXPERIMENTAL RESULTS

On graph classification, UGRAPHEMB does not achieve top 2 on WEB and NCI109, which can be attributed to the fact that there are many node labels associated with the two datasets, as shown in Table 4. Combined with the fact that we use one-hot encoding for the initial node representations as described in Section E.3, UGRAPHEMB has limited capacity to capture the wide variety of node labels. In future, it is promising to explore other node encoding techniques, such as encoding based on node degrees and clustering coefficients [58].

Another possible reason is that the current definition of GED cannot capture the subtle difference between different node labels. For example, a Carbon atom may be chemically more similar to a Nitrogen atom than a Hydrogen atom, which should be reflected in the graph proximity metric. As mentioned in Section D.2, there are other definitions of GED that can handle such cases.

I ANALYSIS OF THE MULTI-SCALE NODE ATTENTION (MSNA) MECHANISM

Table 6 shows how much performance gain our proposed **Multi-Scale Node Attention (MSNA)** mechanism brings to our model, UGRAPHEMB. As can be seen in the table, a simple averaging scheme to generate graph-level embeddings cannot yields much worse performance, compared with the other three mechanisms. The supersource approach is not very bad, but still worse than the attention mechanism which brings learnable components into the model. The **MSNA** mechanism combines the node embeddings from different GCN layers, capturing structural difference at different scales and yielding the best performance.

Table 6: UGRAPHEMB-AVG uses the average of node embeddings to generate graph-level embeddings. UGRAPHEMB-SSRC uses supersource machanism. UGRAPHEMB-NA uses only the node embeddings of the last GCN layer to generate graph-level embeddings, but still uses the node attention mechanism. UGRAPHEMB-MSNA is our full model using three layers of GCN to generate graph-level embeddings. The results are on the IMDBMULTI dataset.

Method	acc	τ	p@10
UGRAPHEMB-AVG	34.73	0.243	0.647
UGRAPHEMB-SSRC	46.02	0.810	0.796
UGRAPHEMB-NA	49.51	0.851	0.810
UGRAPHEMB-MSNA	**50.06**	**0.853**	**0.816**

Please note that under all the four settings, the node embeddings layers are exactly the same, i.e. three sequential GIN layers. From UGRAPHEMB-AVG, we can see that the learnable components in the node embedding model are not enough for good graph-level embeddings.

It is also worth mentioning that the supersource idea works reasonably well, which can be attributed to the fact that the supersource node is connected to every other node in the graph, so that every node passes information to the supersource node, contributing to a relatively informative graph-level embedding. Compared with the averaging scheme, there is additional MLP transformation on the node embedding of the supersource node after the aggregation of other node embeddings, as indicated by the Equation for GIN in the main paper.

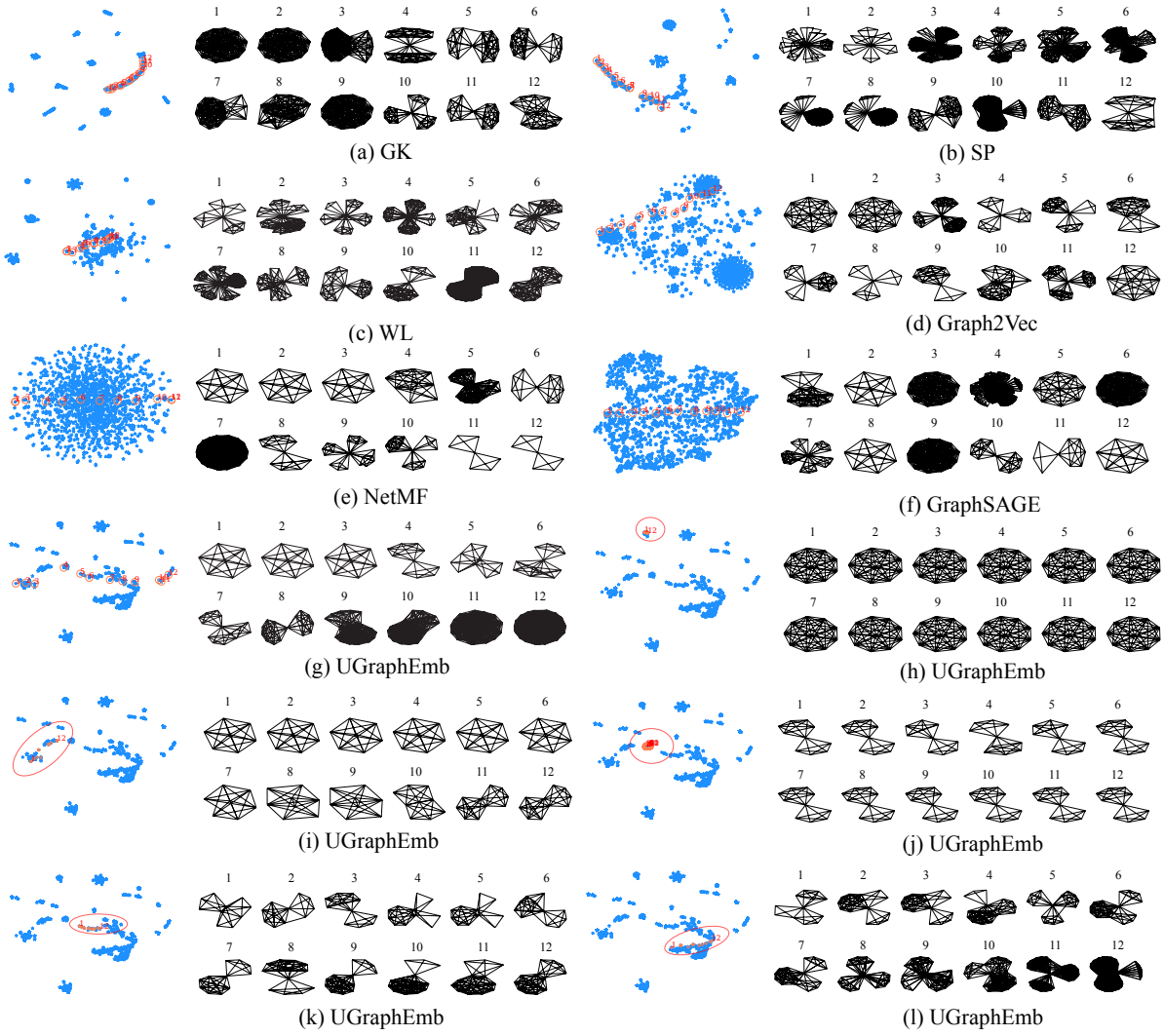

Figure 6: Visualization of the IMDBMULTI dataset. From (a) to (g), for each method, 12 graphs are plotted. For (h) to (l), we focus on UGRAPHEMB: 5 clusters are highlighted in red circles. 12 graphs are sampled from each cluster and plotted to the right.

J EXTRA VISUALIZATIONS

A few more visualizations are included in Fig. 7, 8, 9, and 10, for the PTC, WEB, NCI109, and REDDIT12K datasets used in the main paper.

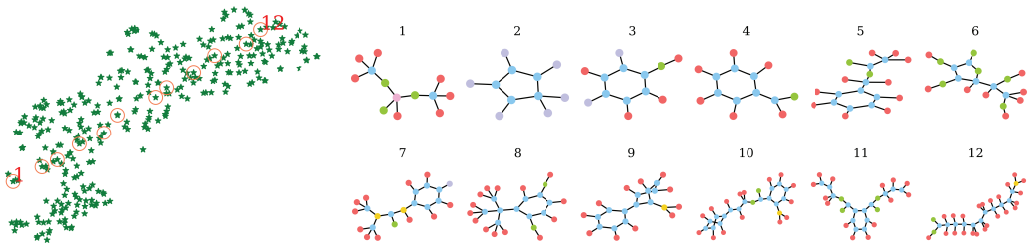

Figure 7: Visualization of the PTC dataset.

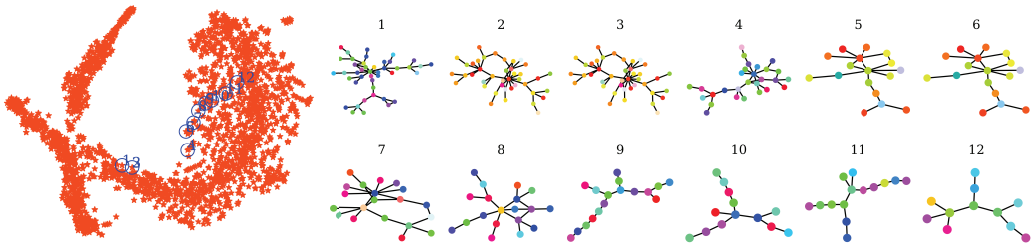

Figure 8: Visualization of the WEB dataset.

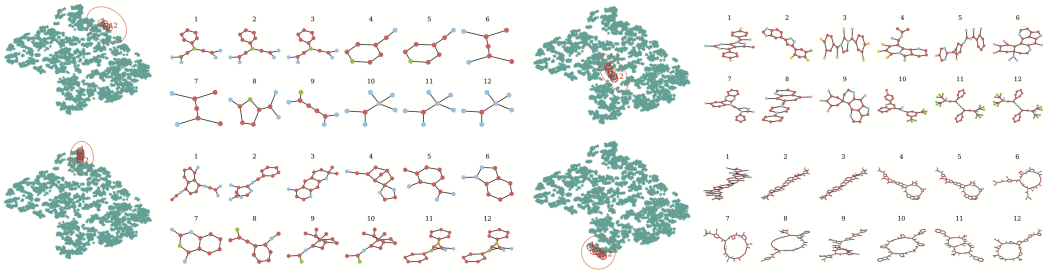

Figure 9: Visualization of the NCI109 dataset.

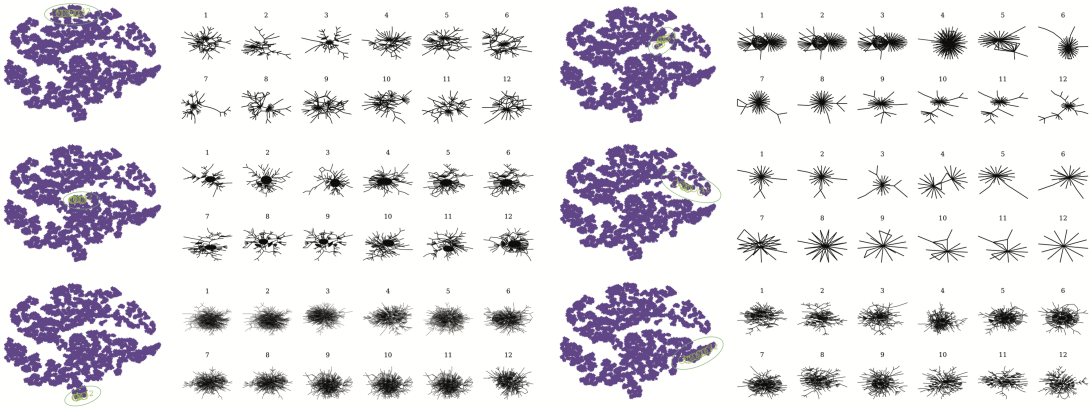

Figure 10: Visualization of the REDDIT12K dataset.