Convolutional Set Matching for Graph Similarity

Yunsheng Bai[*,1] **Hao Ding**[*,2] **Yizhou Sun**[1] **Wei Wang**[1]
[1]University of California, Los Angeles, [2]Purdue University
yba@ucla.edu, ding209@purdue.edu, yzsun@cs.ucla.edu, weiwang@cs.ucla.edu

1 A Multi-Scale Convolutional Model for Pairwise Graph Similarity

We introduce GSimCNN (*G*raph *Sim*ilarity Computation via *C*onvolutional *N*eural *N*etworks) for predicting the similarity score between two graphs. As the core operation of graph similarity search, pairwise graph similarity computation is a challenging problem due to the NP-hard nature of computing many graph distance/similarity metrics.

We demonstrate our model using the Graph Edit Distance (GED) [2] as the example metric. It is defined as the number of edit operations in the optimal alignments that transform one graph into the other, where an edit operation can be an insertion or a deletion of a node/edge, or relabelling of a node. It is NP-hard [3] and costly to compute in practice [4].

The key idea is to turn the pairwise graph distance computation problem into a learning problem. This new approach not only offers parallelizability and efficiency due to the nature of neural computation, but also achieves significant improvement over state-of-the-art GED approximation algorithms.

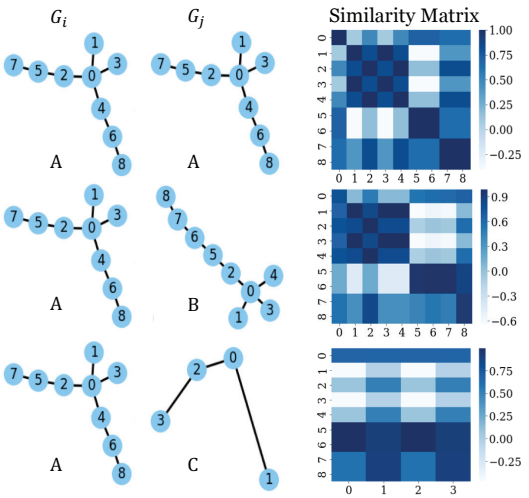


Figure 1: Illustration of three graph-graph similarity matrices generated by our end-to-end GSimCNN trained on the LINUX dataset [1]. Nodes are ordered and labelled with their ids, and darker colors indicate greater similarities between nodes. Convolutional Neural Networks are applied to these matrices to generate the graph-graph similarity score.

Definitions We are given an undirected, unweighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $N = |\mathcal{V}|$ nodes. Node features are summarized in an $N \times D$ matrix $\boldsymbol{H}$. We transform GED into a similarity metric ranging between 0 and 1. Our goal is to learn a neural network based function that takes two graphs as input and outputs the similarity score that can be transformed back to GED through a one-to-one mapping.

GSimCNN GSimCNN consists of the following sequential stages: 1) *Multi-Scale Graph Convolutional Network [5] layers* generate vector representations for each node in the two graphs at different scales; 2) *Graph Interaction layers* compute the inner products between the embeddings of every pair of nodes in the two graphs, resulting in multiple similarity matrices capturing the node-node interaction scores at different scales; 3) *Convolutional Neural Network layers* convert the similarity computation problem into a pattern recognition problem, which provides multi-scale features to a *fully connected network* to obtain a final predicted graph-graph similarity score. An overview of our model is illustrated in Fig. 2.

[*]The two first authors made equal contributions.

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

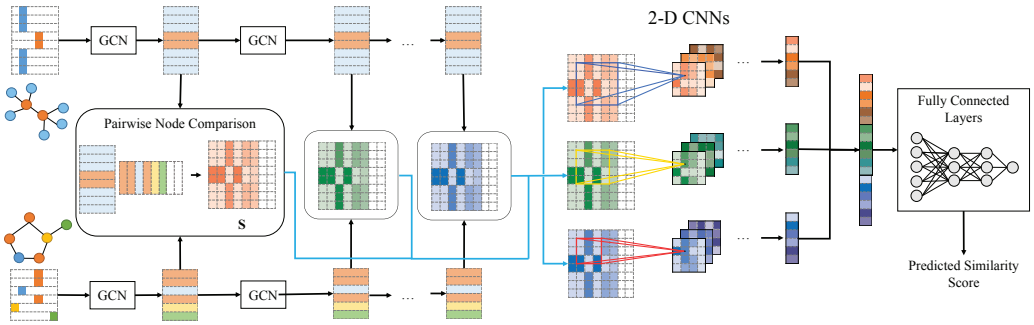


Figure 2: An overview illustration. Three similarity matrices are generated with three 2-D CNNs.

1.1 Stage I: Multi-Scale GCN Layers

In Stage I, we generate node embeddings by multi-layer GCNs, where each layer is defined as [5]:

$$\text{conv}(\boldsymbol{x_i}) = \text{ReLU}(\sum_{j \in \mathcal{N}(i)} \frac{1}{\sqrt{d_i d_j}} \boldsymbol{x}_j \boldsymbol{W}^{(l)} + \boldsymbol{b}^{(l)}) \tag{1}$$

Here, $\mathcal{N}(i)$ is the set of all first-order neighbors of node i plus node i itself; d_i is the degree of node i plus 1; $\boldsymbol{W}^{(l)} \in \mathbb{R}^{D^l \times D^{l+1}}$ is the weight matrix associated with the n-th GCN layer; $\boldsymbol{b}^{(l)} \in \mathbb{R}^{D^{l+1}}$ is the bias; and $\text{ReLU}(x) = \max(0, x)$ is the activation function.

In Fig. 2, different node types are represented by different colors and one-hot encoded as the initial node representation. For graphs with unlabeled nodes, we use the same constant vector as the initial representation. As shown in [6] and [7], the graph convolution operation aggregates the features from the first-order neighbors of the node. Stacking N GCN layers would enable the final representation of a node to include its N-th order neighbors.

Multi-Scale GCN The potential issue of using a deep GCN structure is that the embeddings may be too coarse after aggregating neighbors from multiple scales. The problem is especially severe when the two graphs are very similar, as the differences mainly lie in small substructures. Due to the fact that structural difference may occur at different scales, we extract the output of each GCN layer and construct multi-scale interaction matrices, which will be described in the next stage.

1.2 Stage II: Graph Interaction Layers

We calculate the inner products between all possible pairs of node embeddings between two graphs from different GCN layers, resulting in multiple similarity matrices $\boldsymbol{S}$. Since we later use CNNs to process these matrices, we utilize the breadth-first-search (BFS) node-ordering scheme proposed in [8] to reorder the node embeddings, running in quadratic time in the worst case.

Max Padding Suppose $\mathcal{G}_1$ and $\mathcal{G}_2$ contain N_1 and N_2 nodes, respectively. To reflect the difference in graph sizes in the similarity matrix, we pad $|N_1 - N_2|$ rows of zeros to the node embedding matrix of the smaller of the two graphs, so that both graphs contain $\max(N_1, N_2)$ nodes.

Matrix Resizing To apply CNNs to the similarity matrices, we apply bilinear interpolation, an image resampling technique [9] to resize every similarity matrix. The resulting similarity matrix $\boldsymbol{S}$ has fixed shape $M \times M$, where M is a hyperparameter controlling the degree of loss of information due to the resampling.

The following equation summarizes a single Graph Interaction Layer:

$$\boldsymbol{S} = \text{RES}_M(\widetilde{\boldsymbol{H}}_1 \widetilde{\boldsymbol{H}}_2^T) \tag{2}$$

where $\widetilde{\boldsymbol{H}}_i \in \mathbb{R}^{\max(N_1,N_2) \times D}, i \in \{1, 2\}$ is the padded node embedding matrix $\boldsymbol{H}_i \in \mathbb{R}^{N_i \times D}, i \in \{1, 2\}$ with zero or $|N_1 - N_2|$ nodes padded, and $\text{RES}(\cdot) : \mathbb{R}^{\max(N_1,N_2) \times \max(N_1,N_2)} \mapsto \mathbb{R}^{M \times M}$ is the resizing function.

1.3 Stage III: CNN and Dense Layers

The similarity matrices at different scales are processed by multiple independent CNNs, turning the task of graph similarity measurement into an image processing problem. The filter of CNN detect the

optimal node matching pattern in the image, and max pooling in CNN select the best matching. The CNN results are concatenated and fed into multiple fully connected layers, so that a final similarity score $\hat{s}_{ij}$ is generated for the graph pair $\mathcal{G}_i$ and $\mathcal{G}_j$. The mean squared error loss function is used to train the model.

2 Set Matching Based Graph Similarity Computation

Through GCN transformation, GSimCNN encodes the link structure around each node into its vector representation, and thus regards a graph as a set of node embeddings. It essentially reduces the link structure, and simplifies the graph similarity/distance computation into matching two sets. In this section, we formally define the general approach of using set matching to compute graph similarity/distance, and provide detailed theoretical analysis in Appendix A.

Definition 1 *Graph transforming function: A graph transforming function $f(\cdot)$ transforms a graph $\mathcal{G}$ into a set of objects, M.*

Definition 2 *Set matching function: A set matching function $g(\cdot, \cdot)$ takes two sets as input, and returns a score denoting the degree of matching between the two input sets.*

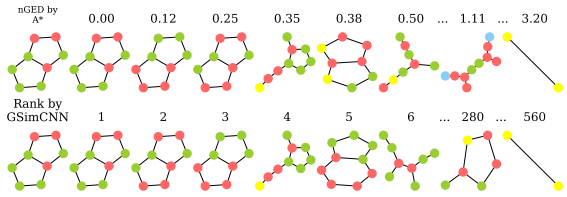
In fact, the forward pass of GSimCNN can be interpreted as a two-step procedure: 1. Applying a GCN-based graph transforming function; 2. Applying a CNN-based set matching function. The Appendix A furnishes the comparisons with two types of graph distance algorithms, which would shed light on why GSimCNN works effectively.

3 Experiments on Graph Similarity Search

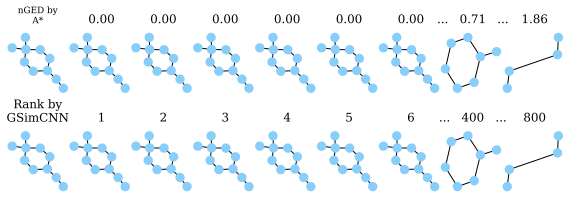
Graph similarity search is among the most important graph-based applications, e.g. finding the chemical compounds that are most similar to a query compound. The goal of these experiments is to demonstrate that GSimCNN can alleviate the computational burden while preserving a good performance of GED approximation. We train GSimCNN on three real graph datasets [3, 1, 10], whose details [2] can be found in Appendix B.

We compare methods based on their ability to correctly compute the pairwise graph similarity and rank the database graphs for user query graphs. The training and validation sets contain 60% and 20% of graphs, respectively, and serve as the database graphs. The validation set is used for optimization of hyperparameters. The test set contains 20% of graphs, treated as the query graphs.

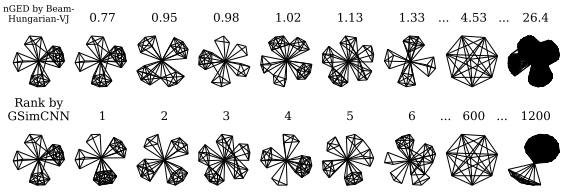
We compare against two sets of baselines: (1) Combinatorial optimization-based algorithms for approximate GED computation: Beam [11], VJ [12], Hungarian [13], HED [14]; (2) Neural Network based models: Siamese MPNN [15], EebAvg, GCN-Mean, GCNMax [16] (see the Appendix C for details).



(a) On AIDS. Different colors represent different node labels.



(b) On LINUX.



(c) On IMDB.

Figure 3: Query case studies.

[2] We make the datasets used in this paper publicly available at `https://drive.google.com/drive/folders/1BFj66jqzR_VlWgASEfNMHwAQZ967HVOW?usp=sharing`.

Table 1: Results on AIDS, LINUX and IMDB. mse is in 10^{-3}. The results are based on the split ratio of 6:2:2. We repeated 10 times on AIDS, and the standard deviation of mse is $4.56 * 10^{-5}$.

Method	AIDS			LINUX			IMDB		
	mse	τ	p@10	mse	τ	p@10	mse	τ	p@10
A*	0.000*	1.000*	1.000*	0.000*	1.000*	1.000*	-	-	-
Beam	12.090	0.463	0.481	9.268	0.714	0.973	2.413*	0.837*	0.803*
Hungarian	25.296	0.378	0.360	29.805	0.517	0.913	1.845*	0.872*	0.825*
VJ	29.157	0.383	0.310	63.863	0.450	0.287	1.831*	0.874*	0.815*
HED	28.925	0.469	0.386	19.553	0.801	0.982	19.400	0.627	0.801
Siamese MPNN	5.184	0.210	0.032	11.737	0.024	0.009	32.596	0.093	0.023
EmbAvg	3.642	0.455	0.176	18.274	0.012	0.071	71.789	0.179	0.233
GCNMean	3.352	0.501	0.186	8.458	0.424	0.141	68.823	0.307	0.200
GCNMax	3.602	0.480	0.195	6.403	0.495	0.437	50.878	0.342	0.425
GSimCNN	**0.787**	**0.724**	**0.534**	**0.058**	**0.962**	**0.992**	**0.743**	**0.847**	**0.828**

* On AIDS and LINUX, A* is used as the ground truth. On the largest dataset, IMDB, A* runs too slow; Since Beam, Hungarian, and VJ are guaranteed to return upper bounds to the exact GEDs, we take the minimum of the three as the ground truth. This approach has been adopted by the ICPR 2016 Graph Distance Contest: `https://gdc2016.greyc.fr/`.

To transform ground-truth GEDs into ground-truth similarity scores to train our model, we first normalize the GEDs: $\mathrm{nGED}(\mathcal{G}_1, \mathcal{G}_2) = \frac{\mathrm{GED}(\mathcal{G}_1, \mathcal{G}_2)}{(|\mathcal{G}_1| + |\mathcal{G}_2|)/2}$, where $|\mathcal{G}_i|$ denotes the number of nodes of $\mathcal{G}_i$ [17], and then adopt the exponential function $\lambda(x) = e^{-x}$, an one-to-one function, to transform the normalized GED into a similarity score in the range of $(0, 1]$.

Effectiveness The results on the three datasets can be found in Table 1. We report *Mean Squared Error (mse)*, *Kendall's Rank Correlation Coefficient (τ)* [18] and *Precision at k (p@k)* for each model on the test set. As shown in Fig. 3. In each demo, the top row depicts the query along with the ground-truth ranking results, labeled with their normalized GEDs to the query. The bottom row shows the graphs returned by our model, each with its rank shown at the top. GSimCNN is able to retrieve graphs similar to the query.

Efficiency In Fig. 4, the results are averaged across queries and in wall time. EmbAvg is the fastest method among all, but its performance is poor, since it simply takes the dot product between two graph-level embeddings (average of node embeddings) as the predicted similarity score. Beam and Hungarian run fast on LINUX, but due to their higher time complexity as shown in Table 2, they scale poorly on the largest dataset, IMDB. In general, neural network based models benefit from the parallelizability and acceleration provided by GPU, and in particular, our model GSimCNN achieves the best trade-off between running time and performance.

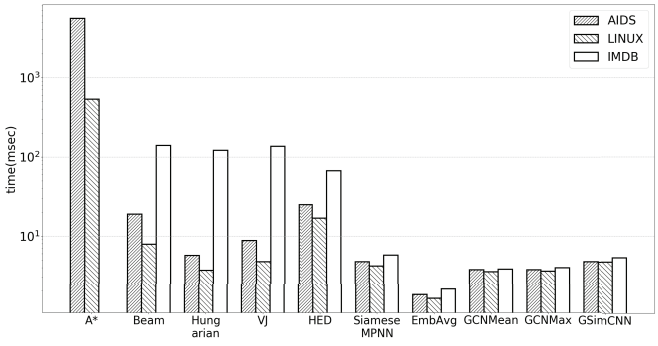


Figure 4: Running time comparison.

Method	Time Complexity
A* [19]	$O({N_1}^{N_2})$
Beam [11]	subexponential
Hungarian [13]	$O((N_1 + N_2)^3)$
VJ [12]	$O((N_1 + N_2)^3)$
HED [14]	$O((N_1 + N_2)^2)$
Siamese MPNN [15]	$O(\max(E_1, E_2, N_1 N_2))$
EmbAvg	$O(\max(E_1, E_2))$
GCNMean [5]	$O(\max(E_1, E_2))$
GCNMax [5]	$O(\max(E_1, E_2))$
GSimCNN	$O(\max(N_1, N_2)^2)$

Table 2: Time complexity comparison. N and E denote the number of nodes and edges respectively.

Future work will investigate the generation of edit sequences for better interpretability of the predicted similarity, the effects of the usage of other node embedding methods, e.g. GraphSAGE [7], the adoption of other graph similarity metrics, and the generalization performance of the model across datasets of different domains.

Acknowledgments

The work was partially supported by NSF DBI 1565137, NSF DGE1829071, NSF III-1705169, NSF CAREER Award 1741634, NIH U01HG008488, NIH R01GM115833, Snapchat gift funds, and PPDai gift fund.

References

[1] Xiaoli Wang, Xiaofeng Ding, Anthony KH Tung, Shanshan Ying, and Hai Jin. An efficient graph indexing method. In *ICDE*, pages 210–221. IEEE, 2012.

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

[3] Zhiping Zeng, Anthony KH Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing stars: On approximating graph edit distance. *PVLDB*, 2(1):25–36, 2009.

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

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

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

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

[8] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *ICML*, pages 5694–5703, 2018.

[9] Philippe Thévenaz, Thierry Blu, and Michael Unser. Image interpolation and resampling. *Handbook of medical imaging, processing and analysis*, 1(1):393–420, 2000.

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

[11] Michel Neuhaus, Kaspar Riesen, and Horst Bunke. Fast suboptimal algorithms for the computation of graph edit distance. In *S+SSPR*, pages 163–172. Springer, 2006.

[12] Stefan Fankhauser, Kaspar Riesen, and Horst Bunke. Speeding up graph edit distance computation through fast bipartite matching. In *GbRPR*, pages 102–111. Springer, 2011.

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

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

[15] Pau Riba, Andreas Fischer, Josep Lladós, and Alicia Fornés. Learning graph distances with message passing neural networks. In *ICPR*, pages 2239–2244, 2018.

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

[17] Rashid Jalal Qureshi, Jean-Yves Ramel, and Hubert Cardot. Graph based shapes representation and recognition. In *GbRPR*, pages 49–60. Springer, 2007.

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

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

[20] Bruce M Maggs and Ramesh K Sitaraman. Algorithmic nuggets in content delivery. *ACM SIGCOMM Computer Communication Review*, 45(3):52–66, 2015.

[21] Andrei Zanfir and Cristian Sminchisescu. Deep learning of graph matching. In *CVPR*, pages 2684–2693, 2018.

[22] Mikhail Zaslavskiy, Francis Bach, and Jean-Philippe Vert. Global alignment of protein–protein interaction networks by graph matching methods. *Bioinformatics*, 25(12):i259–1267, 2009.

[23] Benjamin Edelman, Michael Ostrovsky, and Michael Schwarz. Internet advertising and the generalized second-price auction: Selling billions of dollars worth of keywords. *American Economic Review*, 97(1):242–259, March 2007.

[24] Alvin E Roth. The evolution of the labor market for medical interns and residents: A case study in game theory. *Journal of Political Economy*, 92:991–1016, 1984.

[25] Holger Fröhlich, Jörg K Wegner, Florian Sieker, and Andreas Zell. Optimal assignment kernels for attributed molecular graphs. In *ICML*, pages 225–232. ACM, 2005.

[26] Fredrik D Johansson and Devdatt Dubhashi. Learning with similarity functions on graphs using matchings of geometric embeddings. In *KDD*, pages 467–476. ACM, 2015.

[27] Nils M Kriege, Pierre-Louis Giscard, and Richard Wilson. On valid optimal assignment kernels and applications to graph classification. In *NIPS*, pages 1623–1631, 2016.

[28] Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. Matching node embeddings for graph similarity. In *AAAI*, pages 2429–2435, 2017.

[29] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40(2):99–121, 2000.

[30] Michel L Balinski. Fixed-cost transportation problems. *Naval Research Logistics Quarterly*, 8(1):41–54, 1961.

[31] Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3d object reconstruction from a single image. In *CVPR*, volume 2, page 6, 2017.

[32] Ofir Pele and Michael Werman. Fast and robust earth mover's distances. In *ICCV*, volume 9, pages 460–467, 2009.

[33] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[34] Roy Jonker and Anton Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38(4):325–340, 1987.

[35] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

Appendices

A Connections with Set Matching

In this section, we present GSimCNN from the perspective of set matching, by making theoretical connections with two types of graph matching methods: optimal assignment kernels for graph classification and bipartite graph matching for GED computation. In fact, beyond graphs, set matching has broader applications in Computer Networking (e.g. Internet content delivery) [20], Computer Vision (e.g. semantic visual matching) [21], Bioinformatics (e.g. protein alignment) [22], Internet Advertising (e.g. advertisement auctions) [23], Labor Markets (e.g. intern host matching) [24], etc. This opens massive possibilities for future work and suggests the potential impact of GSimCNN beyond the graph learning community.

A.1 Connection with Optimal Assignment Kernels

Graph kernels measure the similarity between two graphs, and have been extensively applied to the task of graph classification. Formally speaking, a valid kernel on a set $\mathcal{X}$ is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that there is a real Hilbert space (feature space) $\mathcal{H}$ and a feature map function $\phi : \mathcal{X} \to \mathbb{R}$ such that $k(x, y) = \langle \phi(x), \phi(y) \rangle$ for every x and y in $\mathcal{X}$, where $\langle \cdot, \cdot \rangle$ denotes the inner product of $\mathcal{H}$.

Among different families of graph kernels, optimal assignment kernels establish the correspondence between parts of the two graphs, and have many variants [25, 26, 27, 28]. Let $\mathcal{B}(X, Y)$ denote the set of all bijections between two sets of nodes, X and Y Let $k(x, y)$ denote a base kernel that measures the similarity between two nodes x and y. An optimal assignment graph kernel $K^k_{\mathcal{B}}$ is defined as

$$\begin{aligned} K^k_{\mathcal{B}}(X, Y) &= \max_{B \in \mathcal{B}(X,Y)} W(B), \\ \text{where} \quad W(B) &= \sum_{(x,y) \in B} k(x, y) \end{aligned} \tag{3}$$

Intuitively, the optimal assignment graph kernels maximize the total similarity between the assigned parts. If the two sets are of different cardinalities, one can add new objects z with $k(z, x) = 0, \forall x \in \mathcal{X}$ to the smaller set [27].

Let us take the Earth Mover's Distance (EMD) kernel [28] as an example, since it is among the most similar method to our proposed approach. It treats a graph as a bag of node embedding vectors, but instead of utilizing the pairwise inner products between node embeddings to approximate GED, it computes the optimal "travel cost" between two graphs, where the cost is defined as the L-2 distance between node embeddings. Given two graphs with node embeddings $\boldsymbol{X} \in \mathbb{R}^{N_1 \times D}$ and $\boldsymbol{Y} \in \mathbb{R}^{N_2 \times D}$, it solves the following transportation problem [29]:

$$\begin{aligned} &\min \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} \boldsymbol{T}_{ij} ||\boldsymbol{x}_i - \boldsymbol{y}_j||_2 \\ &\text{subject to} \\ &\sum_{i=1}^{N_1} \boldsymbol{T}_{ij} = \frac{1}{N_2} \quad \forall j \in \{1, ..., N_2\} \\ &\sum_{j=1}^{N_2} \boldsymbol{T}_{ij} = \frac{1}{N_1} \quad \forall i \in \{1, ..., N_1\} \\ &\boldsymbol{T}_{ij} \geq 0 \quad \forall i \in \{1, ..., N_1\}, \forall j \in \{1, .., N_2\} \end{aligned} \tag{4}$$

where $\boldsymbol{T} \in \mathbb{R}^{N_1 \times N_2}$ denotes the flow matrix, with $\boldsymbol{T}_{ij}$ being how much of node i in $\mathcal{G}_1$ travels (or "flows") to node j in $\mathcal{G}_2$. In other words, the EMD between two graphs is the minimum amount of "work" that needs to be done to transform one graph to another, where the optimal transportation plan is encoded by $\boldsymbol{T}^*$.

It has been shown that if $N_1 = N_2 = N$, the optimal solution satisfies $\boldsymbol{T}^*_{ij} \in \{0, \frac{1}{N}\}$ [30], satisfying the optimal bijection requirement of the assignment kernel. Even if $N_1 \neq N_2$, this can still be regarded as approximating an assignment problem [31].

To show the relation between the EMD kernel and our approach, we consider GSimCNN as a mapping function that, given two graphs with node embeddings $\boldsymbol{X} \in \mathbb{R}^{N_1 \times D}$ and $\boldsymbol{Y} \in \mathbb{R}^{N_2 \times D}$, produces one score as the predicted similarity score, which is compared against the ground-truth similarity score:

$$\min(h_\Theta(\boldsymbol{X}, \boldsymbol{Y}) - \lambda(\mathrm{GED}(\mathcal{G}_1, \mathcal{G}_2)))^2 \tag{5}$$

where $h_\Theta(\boldsymbol{X}, \boldsymbol{Y})$ represents the Graph Interaction and CNN layers but can potentially be replaced by any neural network transformation.

To further see the connection, we consider one CNN layer with one filter of size N by N, where $N = \max(N_1, N_2)$. Then Eq. 5 becomes:

$$\min(\sigma(\sum_{i=1}^{N}\sum_{j=1}^{N} \boldsymbol{\Theta}_{ij}(\boldsymbol{x}_i^T \boldsymbol{y}_j)) - \lambda(\mathrm{GED}(\mathcal{G}_1, \mathcal{G}_2)))^2 \tag{6}$$

where $\boldsymbol{\Theta} \in \mathbb{R}^{N \times N}$ is the convolutional filter.

Compared with the EMD kernel, our method has two benefits. (1) The mapping function and the node embeddings $\boldsymbol{X}$ and $\boldsymbol{Y}$ are simultaneously learned through backpropagation, while the kernel method solves the assignment problem to obtain $\boldsymbol{T}$ and uses fixed node embeddings $\boldsymbol{X}$ and $\boldsymbol{Y}$, e.g. generated by the decomposition of the graph Laplacian matrix. Thus, GSimCNN is suitable for *learning* an approximation of the GED graph distance metric, while the kernel method cannot. The typical usage of a graph kernel is to feed the graph-graph similarities into a SVM classifier for graph classification. (2) The best average time complexity of solving Eq. 4 scales $O(N^3 logN)$ [32], where N denotes the number of total nodes in two graphs, while the convolution operation is in O($(\max(N_1, N_2))^2$) time.

A.2 Connection with Bipartite Graph Matching

Among the existing approximate GED computation algorithms, Hungarian [13] and VJ [12] are two classic ones based on bipartite graph matching. Similar to the optimal assignment kernels, Hungarian and VJ also find an optimal match between the nodes of two graphs. However, different from the EMD kernel, the assignment problem has stricter constraints: One node in $\mathcal{G}_1$ can be only mapped to one other node in $\mathcal{G}_2$. Thus, the entries in the assignment matrix $\boldsymbol{T} \in \mathbb{R}^{N^{'} \times N^{'}}$ are either 0 or 1, denoting the operations transforming $\mathcal{G}_1$ into $\mathcal{G}_2$, where $N^{'} = N_1 + N_2$. The assignment problem takes the following form:

$$\begin{aligned}
&\min \sum_{i=1}^{N^{'}}\sum_{j=1}^{N^{'}} \boldsymbol{T}_{ij}\boldsymbol{C}_{ij} \\
&\text{subject to} \\
&\sum_{i=1}^{N^{'}} \boldsymbol{T}_{ij} = 1 \quad \forall j \in \{1, ..., N^{'}\} \\
&\sum_{j=1}^{N^{'}} \boldsymbol{T}_{ij} = 1 \quad \forall i \in \{1, ..., N^{'}\} \\
&\boldsymbol{T}_{ij} \in \{0, 1\} \quad \forall i \in \{1, ..., N^{'}\}, \forall j \in \{1, .., N^{'}\}
\end{aligned} \tag{7}$$

The cost matrix $\boldsymbol{C} \in \mathbb{R}^{N^{'} \times N^{'}}$ reflects the GED model, and is defined as follows:

$$\boldsymbol{C} = \left[\begin{array}{ccc|ccc}
\boldsymbol{C}_{1,1} & \dots & \boldsymbol{C}_{1,N_2} & \boldsymbol{C}_{1,\epsilon} & \dots & \infty \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
\boldsymbol{C}_{N_1,1} & \dots & \boldsymbol{C}_{N_1,N_2} & \infty & \dots & \boldsymbol{C}_{N_1,\epsilon} \\
\hline
\boldsymbol{C}_{\epsilon,1} & \dots & \infty & 0 & \dots & 0 \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
\infty & \dots & \boldsymbol{C}_{\epsilon,N_2} & 0 & \dots & 0
\end{array}\right]$$

where $\boldsymbol{C}_{i,j}$ denotes the cost of a substitution, $\boldsymbol{C}_{i,\epsilon}$ denotes the cost of a node deletion, and $\boldsymbol{C}_{\epsilon,j}$ denotes the cost of a node insertion. According to our GED definition, $\boldsymbol{C}_{ij} = 0$ if the labels of node i and node j are the same, and 1 otherwise; $\boldsymbol{C}_{i,\epsilon} = \boldsymbol{C}_{\epsilon,j} = 1$.

Exactly solving this constrained optimization program would yield the exact GED solution [12], but it is NP-complete since it is equivalent to finding an optimal matching in a complete bipartite graph [13].

To efficiently solve the assignment problem, the Hungarian algorithm [33] and the Volgenant Jonker (VJ) [34] algorithm are commonly used, which both run in cubic time. In contrast, GSimCNN takes advantage of the exact solutions of this problem during the training stage, and computes the approximate GED during testing in quadratic time, without the need for solving any optimization problem for a new graph pair.

A.3 Summary of Connections with Set Matching

To sum up, our model, GSimCNN, represents a new approach to modeling the similarities between graphs, by first transforming each graph into a set of node embeddings, where embeddings encode the link structure around each node, and then matching two sets of node embeddings. The entire model can be trained in an end-to-end fashion. In contrast, the other two approaches in Table 3 also model the graph-graph similarity by viewing a graph as a set, but suffer from limited learnability and cannot be trained end-to-end. Due to its neural network nature, the convolutional set matching approach enjoys flexibility and thus has the potential to be extended to solve other set matching problems.

Table 3: Summary of three set matching based approaches to graph similarity. $f(\cdot)$ denotes the graph transforming function, and $g(\cdot, \cdot)$ denotes the set matching function. They are defined in Section 2 in the main paper.

Approach	Example(s)	$f(\mathcal{G})$	$g(\mathcal{G}_1, \mathcal{G}_2)$
Optimal Alignment Kernels	EMD kernel [28]	Node Embedding	Solver of Eq. 4
Bipartite Graph Matching	Hungarian [13], VJ [12]	Nodes of $\mathcal{G}$	Solver of Eq. 7
Convolutional Set Matching	GSimCNN	Node Embedding	Graph Interaction + CNNs

B Dataset Description

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

AIDS AIDS is a collection of antivirus screen chemical compounds from the Developmental Therapeutics Program at NCI/NIH 7 [3], and has been used in several existing work on graph similarity search [3, 1]. It contains 42,687 chemical compound structures with Hydrogen atoms omitted. We randomly select 700 graphs, each of which has 10 or less than 10 nodes.

LINUX The LINUX dataset was originally introduced in [1]. It consists of 48,747 Program Dependence Graphs (PDG) generated from the Linux kernel. Each graph represents a function, where a node represents one statement and an edge represents the dependency between the two statements. We randomly select 1000 graphs of equal or less than 10 nodes each.

IMDB The IMDB dataset [10] (named "IMDB-MULTI") consists of 1500 ego-networks of movie actors/actresses, where there is an edge if the two people appear in the same movie. To test the scalability and efficiency of our proposed approach, we use the full dataset without any selection.

Since the GED computation is pairwise, it is necessary to take the number of pairs into consideration. There are 294K, 0.6M and 1.35M total number of training graph pairs in the AIDS, LINUX and IMDB dataset, respectively.

[3] `https://wiki.nci.nih.gov/display/NCIDTPdata/AIDS+Antiviral+Screen+Data`.

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	Min	Max	Mean	Std
AIDS	Chemical Compounds	29	700	2	10	8.9	1.4
LINUX	Program Dependence Graphs	1	1000	4	10	7.7	1.5
IMDB	Ego-Networks	1	1500	7	89	13.0	8.5

C Baseline Details

Our baselines include two types of approaches, fast approximate GED computation algorithms and neural network based models.

The first category of baselines includes four classic algorithms for GED computation. (1) *A*-Beamsearch (Beam)*, (2) *Hungarian*, and (3) *VJ* return upper bounds of the true GEDs. (2) and (3) are described in Section A.2. (4) *HED* [14] is based on Hausdorff matching, and yields GEDs smaller than or equal to the actual GEDs. Therefore, Beam, Hungarian, and VJ are used to determine the ground truth for IMDB without considering HED.

The second category of baselines includes the following neural network architectures. (1) *Siamese MPNN* [15], (2) *EmbAvg* , (3) *GCNMean* and (4) *GCNMax* [16] are four neural network architectures. (1) generates all the pairwise node embedding similarity scores, and for each node, it finds one node in the other graph with the highest similarity score. It simply sums up all these similarity scores as the final result. (2), (3), and (4) take the dot product of the graph-level embeddings of the two graphs to produce the similarity score. (2) takes the unweighted average of node embeddings as the graph embedding. (3) and (4) adopt the original GCN architectures based on graph coarsening with mean and max pooling, respectively, to gain the graph embedding. *GSimCNN* is our complete model with three levels of comparison granularities.

D Parameter Setting and Implementation Details

For the proposed model, we use the same network architecture on all the datasets. We set the number of GCN layers to 3, and use ReLU as the activation function. For the resizing scheme, all the similarity matrices are resized to 10 by 10. For the CNNs, we use the following architecture: conv(6,1,1,16), maxpool(2), conv(6,1,16,32), maxpool(2), conv(5,1,32,64), maxpool(2), conv(5,1,64,128), maxpool(3), conv(5,1,128,128), maxpool(3) ("conv(window size, kernel stride, input channels, output channels)"; "maxpool(pooling size)").

GSimCNN is implemented using TensorFlow, and tested on a single machine with an Intel i7-6800K CPU and one Nvidia Titan GPU. As for training, we set the batch size to 128, use the Adam algorithm for optimization [35], and fix the initial learning rate to 0.001. We set the number of iterations to 15000, and select the best model based on the lowest validation loss.

E Discussion and Result Analysis

E.1 Comparison between GSimCNN and Baselines

The classic algorithms for GED computation (e.g. A*, Beam, Hungarian, VJ, HED, etc.) usually require rather complicated design and implementation based on discrete optimization or combinatorial search. In contrast, GSimCNN is learnable and can take advantage of the exact solutions of GED computation during the training stage. Regarding time complexity, GSimCNN computes the approximate GED in quadratic time, without the need for solving any optimization problem for a new graph pair. In fact, GSimCNN computes the similarity score. However, it can be mapped back to the corresponding GED.

For the simple neural network based approaches:

1. For EmbAvg, GCNMean and GCNMax, they are all calculating the similarity score based on the inner product of graph embeddings. For EmbAvg, it first takes the unweighted average of node embeddings to gain the graph embedding, while GCNMean and GCNMax adopt graph coarsening

with mean and max pooling, respectively, to obtain the graph embedding. The potential issue for these models are: (1) They fail to leverage the information from fine-grained node-level comparisons; (2) They calculate the final result based on the inner product between two graph embeddings, without any module to learn the graph level interactions. In contrast, GSimCNN constructs multi-scale similarity matrices to encode the node-level interaction at different scales, and adopts CNNs to detect the matching patterns afterwards.

2. For Siamese MPNN, it also computes all the pairwise node embedding similarity scores, like GSimCNN. However, it goes through all the nodes in both graphs, and for every node, it finds one node in the other graph with the highest similarity score, and simply sums up all these similarity scores as the final result with no trainable components afterwards. Our model, instead, is equipped with the learnable CNN kernels and dense layers to extract matching patterns with similarity scores from multiple scales.

For efficiency, notice that A* can no longer be used to provide the ground truth for IMDB, the largest dataset, as "no currently available algorithm manages to reliably compute GED within reasonable time between graphs with more than 16 nodes" [4]. This not only shows the significance of time reduction for computing pairwise graph similarities/distances, but also highlights the challenges of creating a fast and accurate graph similarity search system.

As seen in Table 1 and Fig. 4 in the main paper, GSimCNN strikes a good balance between effectiveness and efficiency. Specifically, GSimCNN achieves the smallest error, the best ranking performance, and great time reduction on the task of graph similarity search.

E.2 Comparison between GSimCNN and Simple Variants of GSimCNN

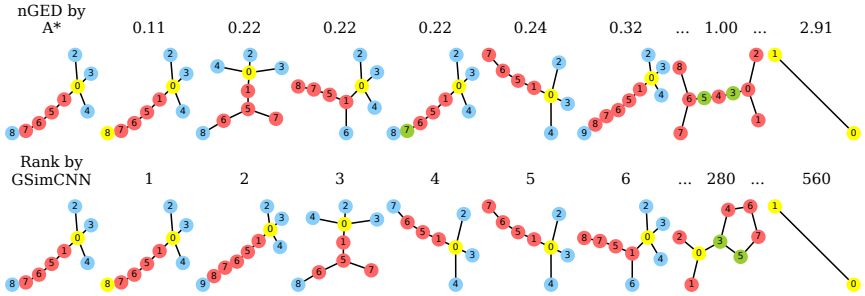
Table 5: GSimCNN-L1-Pad and GSimCNN-L1-Resize use the node embedding by the last of the three GCN layers to generate the similarity matrix, with the padding and resizing schemes, respectively.

	AIDS			**LINUX**			**IMDB**		
Method	mse	τ	p@10	mse	τ	p@10	mse	τ	p@10
GSimCNN-L1-Pad,	0.807	0.714	0.514	0.141	0.951	0.983	6.455	0.562	0.552
GSimCNN-L1-Resize	0.811	0.715	0.499	0.103	0.959	0.986	0.896	0.828	0.810
GSimCNN	**0.787**	**0.724**	**0.534**	**0.058**	**0.962**	**0.992**	**0.743**	**0.847**	**0.828**

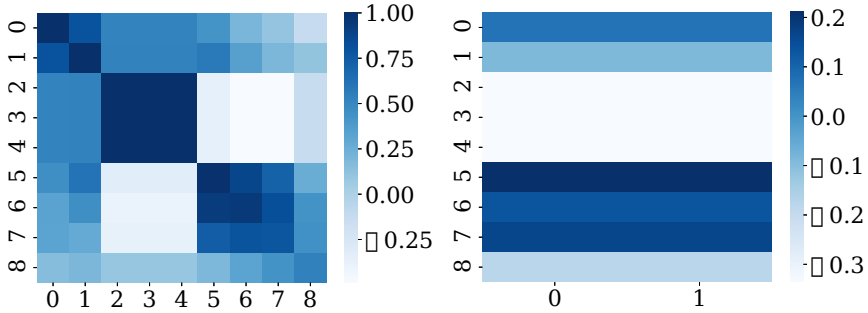
The effectiveness of multiple levels of comparison over a single level can be seen from the performance boost from the last two rows of Table 5. The improvement is especially significant on IMDB, which can be attributed to the large average graph size, as seen from Table 4. The large variance of graph sizes associated with IMDB also favors the proposed resizing scheme over the padding scheme, which is also reflected in the results. On AIDS and LINUX, the use of resizing does not improve the performance much, but on IMDB, the improvement is much more significant.

F Extra Visualization

A few more visualizations are included in Fig. 5, 6, 7, 8, 9, and 10. In each figure, two similarity matrices are visualized, the left showing the similarity matrix between the query graph and the most similar graph, the right showing the similarity matrix between the query and the least similar graph.



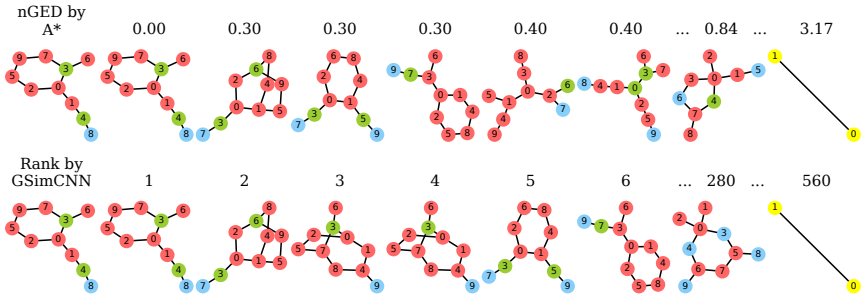
(a) Query ranking result.



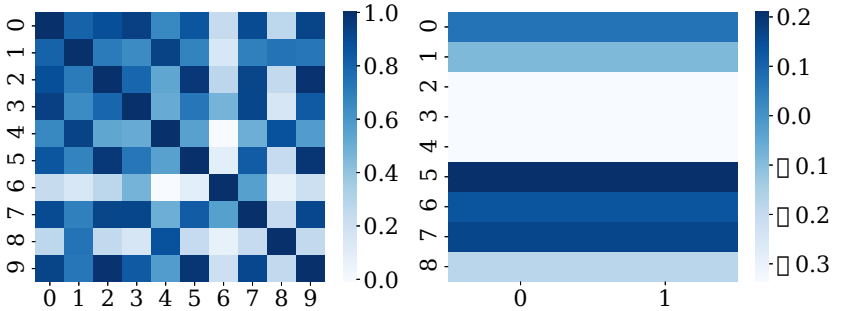
(b) Similarity matrix between the query and the most similar graph ranked by both methods.

(c) Similarity matrix between the query and the least similar graph ranked by both methods.

Figure 5: A query case study on AIDS.



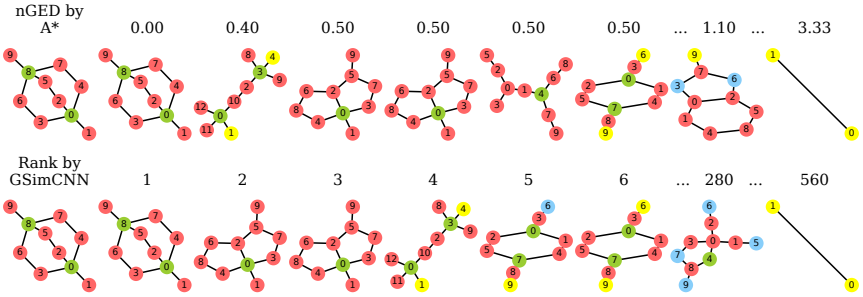
(a) Query ranking result.



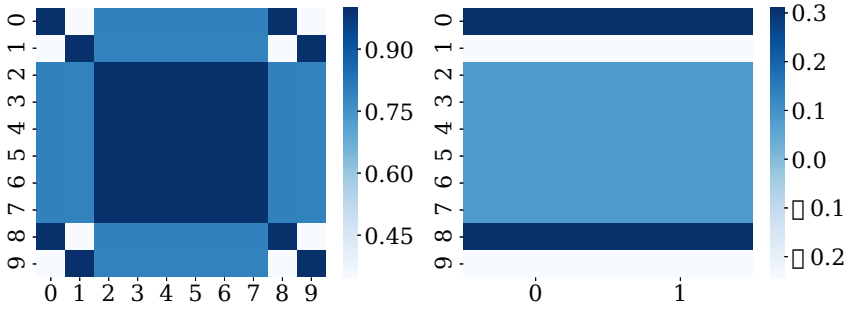
(b) Similarity matrix between the query and the most similar graph ranked by both methods.

(c) Similarity matrix between the query and the least similar graph ranked by both methods.

Figure 6: A query case study on AIDS.



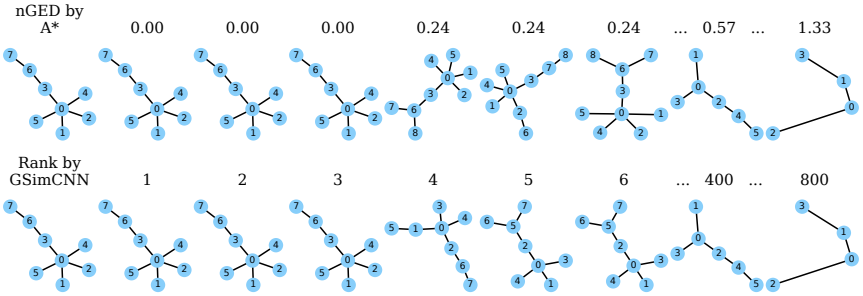
(a) Query ranking result.



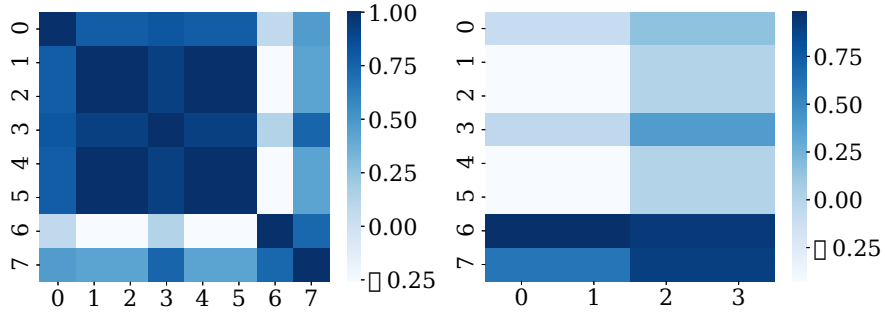
(b) Similarity matrix between the query and the most similar graph ranked by both methods.

(c) Similarity matrix between the query and the least similar graph ranked by both methods.

Figure 7: A query case study on AIDS.



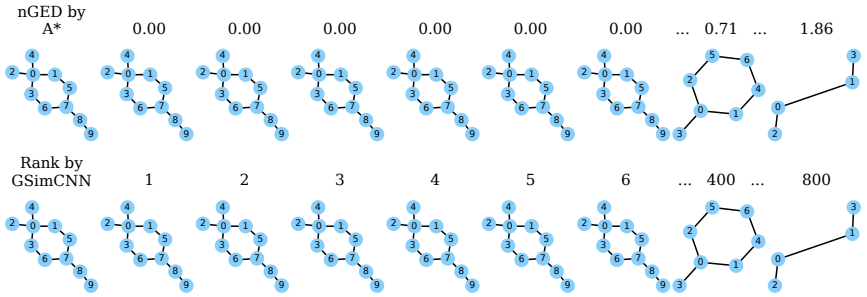
(a) Query ranking result.



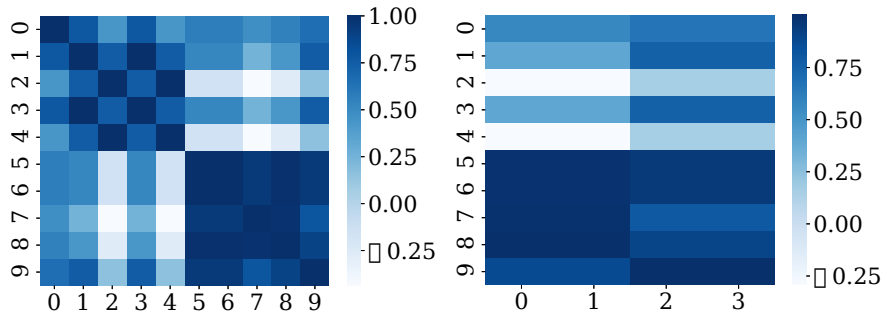
(b) Similarity matrix between the query and the most similar graph ranked by both methods.

(c) Similarity matrix between the query and the least similar graph ranked by both methods.

Figure 8: A query case study on LINUX.



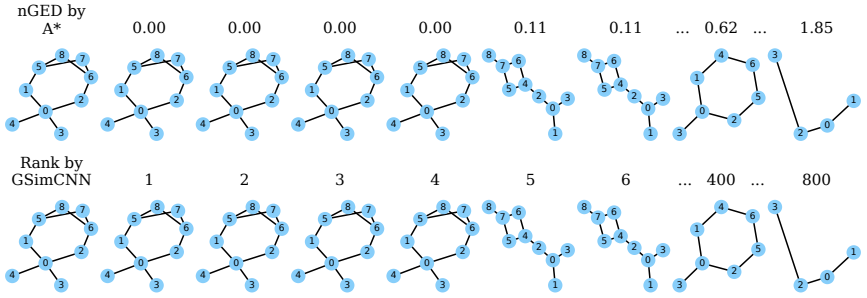
(a) Query ranking result.



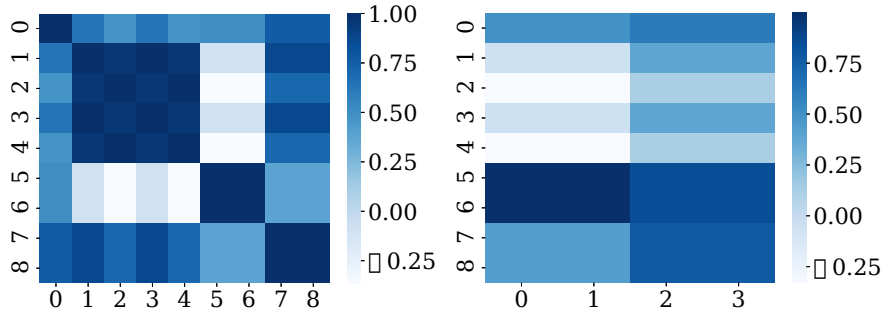
(b) Similarity matrix between the query and the most similar graph ranked by both methods.

(c) Similarity matrix between the query and the least similar graph ranked by both methods.

Figure 9: A query case study on LINUX.



(a) Query ranking result.



(b) Similarity matrix between the query and the most similar graph ranked by both methods.

(c) Similarity matrix between the query and the least similar graph ranked by both methods.

Figure 10: A query case study on LINUX.