GRAPH MATCHING NETWORKS FOR LEARNING THE SIMILARITY OF GRAPH STRUCTURED OBJECTS

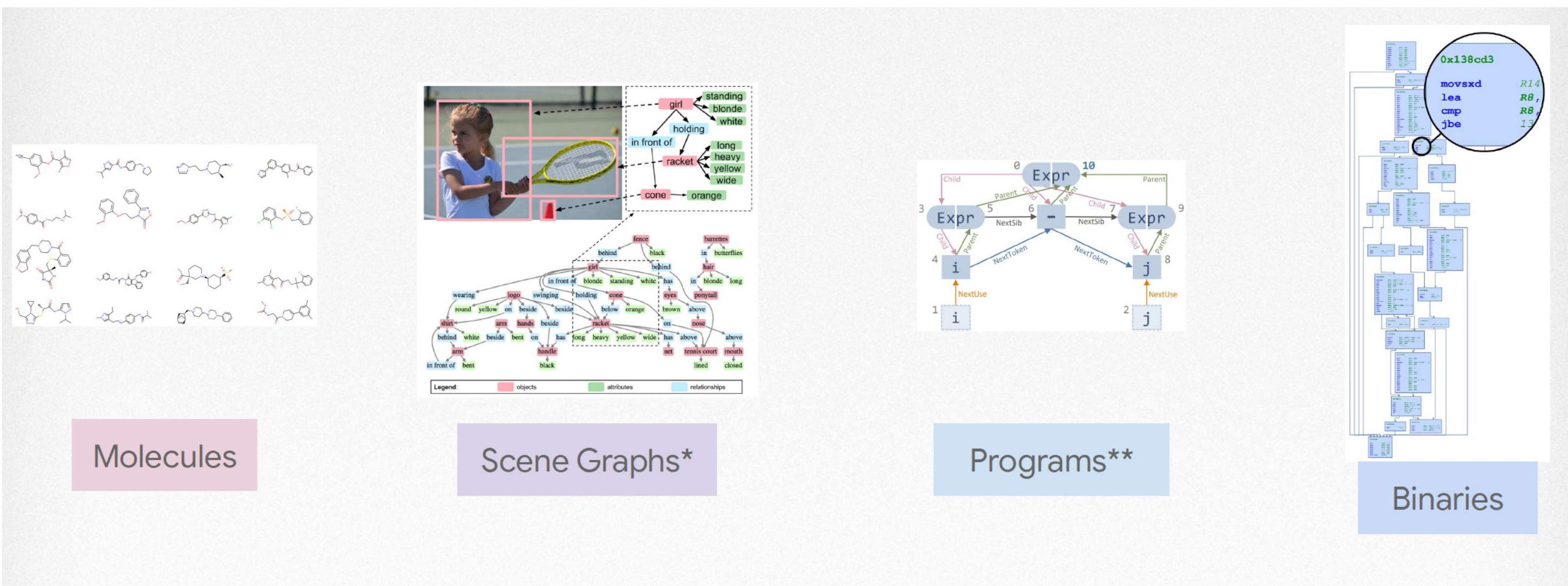
YUJIA LI, CHENJIE GU, THOMAS DULLIEN*, ORIOL VINYALS, PUSHMEET KOHLI

Presented by Xuehan Chen

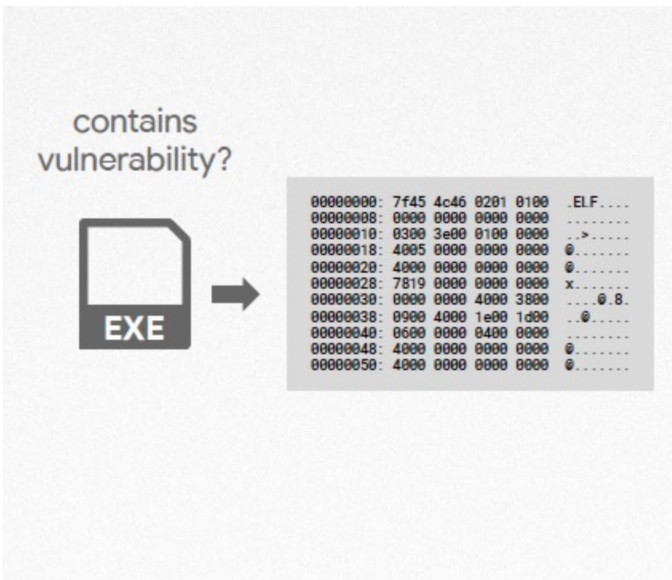
04/26/2021

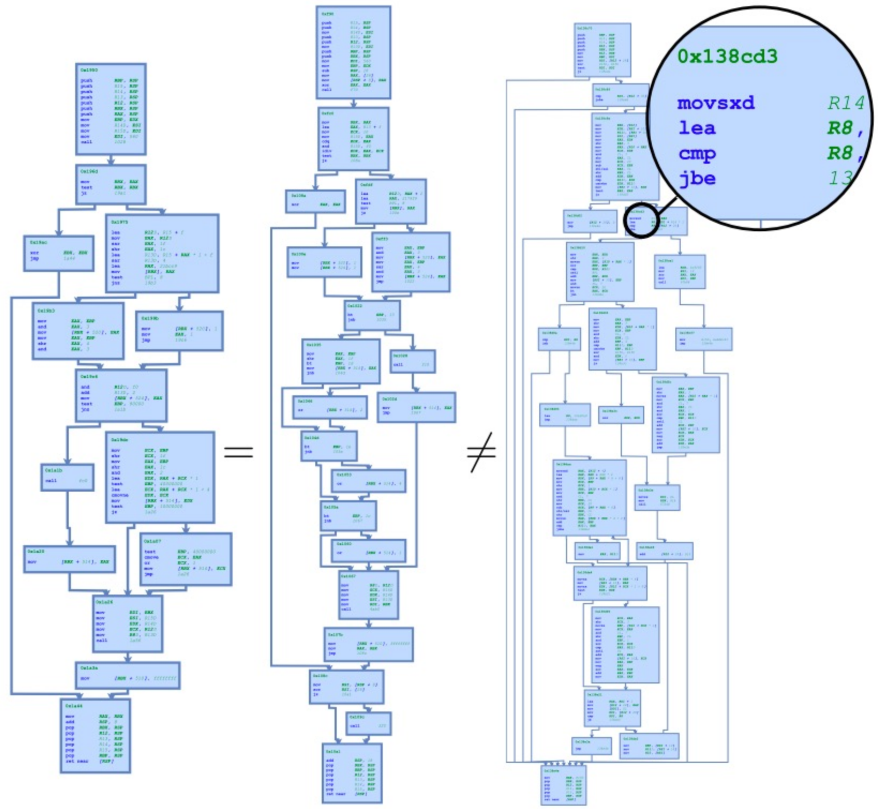
BACKGROUND

- Graph structure data application-> Graph similarity learning
  - eg, Binaries -> Software Vulnerabilities



BINARY FUNCTION SIMILARITY SEARCH PROBLEM





APROACHES

- Exsiting approaches:
  - Graph hashes :
    - human-designed hash functions
    - good at exact match rather than estimating similarity
  - Graph Kernels
    - human-designed kernels to measure similarity between graphs

- Proposed approaches:
  - Graph Neural Network (GNN)
  - Graph Matching Network (GMN)

ANNOTATION

- Graph: $G(V,E)$; $X_i, i \in V$; $X_{ij}, (i,j) \in E$
- Two graphs $G1 = (V1,E1)$ and $G2 = (V2,E2)$
- Similarity score: $s(G1,G2)$

GRAPH EMBEDDING MODELS WITH GNN

- 1. Encoder

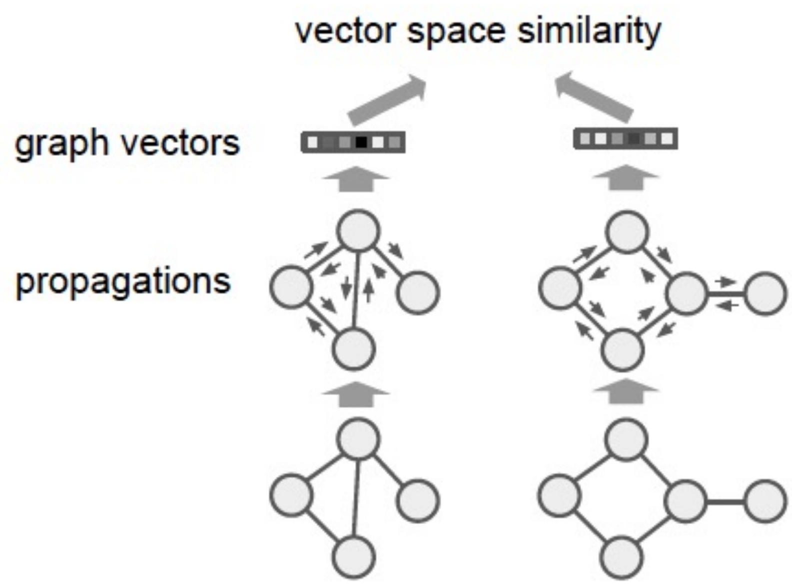
$$
\begin{aligned}
\mathbf{h}_i^{(0)} &= \mathrm{MLP}_{\mathrm{node}}(\mathbf{x}_i), \quad \forall i \in V \\
\mathbf{e}_{ij} &= \mathrm{MLP}_{\mathrm{edge}}(\mathbf{x}_{ij}), \quad \forall (i,j) \in E.
\end{aligned} \tag{1}
$$

- 2. Propagation layers

$$
\begin{aligned}
\mathbf{m}_{j\to i} &= f_{\mathrm{message}}(\mathbf{h}_i^{(t)}, \mathbf{h}_j^{(t)}, \mathbf{e}_{ij}) \\
\mathbf{h}_i^{(t+1)} &= f_{\mathrm{node}}\left(\mathbf{h}_i^{(t)}, \sum_{j:(j,i)\in E} \mathbf{m}_{j\to i}\right)
\end{aligned} \tag{2}
$$

- 3. Aggregator

$$
\mathbf{h}_G = \mathrm{MLP}_G\left(\sum_{i\in V} \sigma(\mathrm{MLP}_{\mathrm{gate}}(\mathbf{h}_i^{(T)})) \odot \mathrm{MLP}(\mathbf{h}_i^{(T)})\right), \tag{3}
$$



GRAPH MATCHING NETWORKS

- Matching models compute the similarity score jointly on the pair, rather than first independently mapping each graph to a vector.

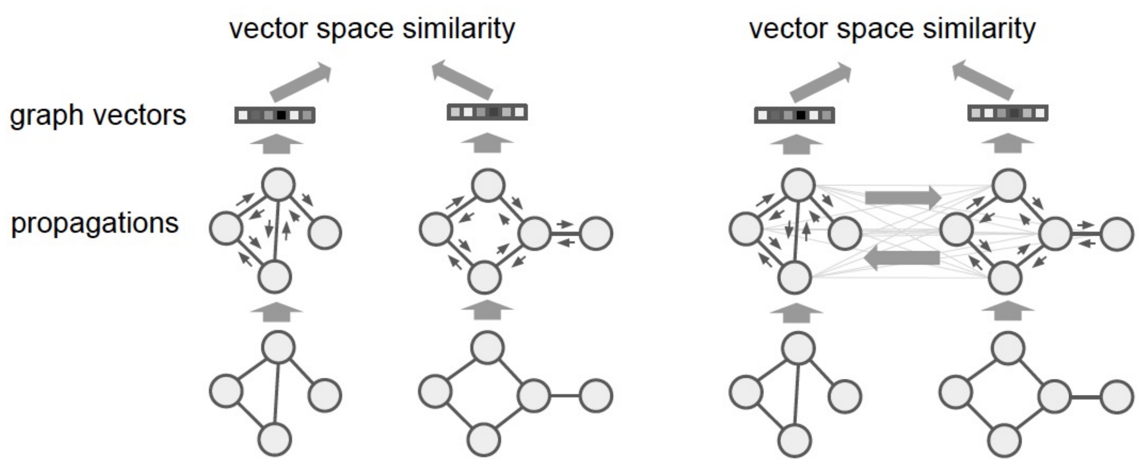
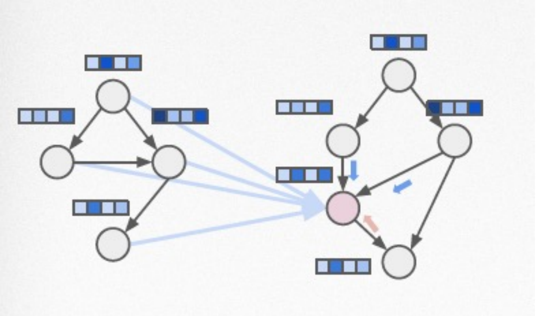


Figure 2. Illustration of the graph embedding (left) and matching models (right).

GRAPH MATCHING NETWORKS



- Change the node update module in each propagation layer: (embed+match)
- Aggregated messages on the edges for each graph + Cross-graph matching information

$$\mathbf{h}_i^{(t+1)} = f_{\text{node}}\left(\mathbf{h}_i^{(t)}, \sum_j \mathbf{m}_{j\to i}, \sum_{j'} \boldsymbol{\mu}_{j'\to i}\right) \quad (6)$$

$$\mathbf{m}_{j\to i} = f_{\text{message}}(\mathbf{h}_i^{(t)}, \mathbf{h}_j^{(t)}, \mathbf{e}_{ij}), \forall (i,j) \in E_1 \cup E_2 \quad (4)$$

$$\boldsymbol{\mu}_{j\to i} = f_{\text{match}}(\mathbf{h}_i^{(t)}, \mathbf{h}_j^{(t)}), \quad \forall i \in V_1, j \in V_2, \text{or } i \in V_2, j \in V_1 \quad (5)$$

- f_{match}: Cross-graph matching measures how will a node in one graph can be matched to one or more nodes in the other.
- $S_h$is a vector space similarity metric, like Euclidean or cosine similarity
- $a_{j\to i}$: attention weights: softmax(e_{ij}), $e_{ij} = s_h(h_i$(t), h_j(t))
- $\sum_j \mu_{j\to i}$: (Total cross-graph message: sum of weighted difference) measures the difference between h_i(t) and its closest neighbor in the other graph

$$a_{j\to i} = \frac{\exp(s_h(\mathbf{h}_i^{(t)}, \mathbf{h}_j^{(t)}))}{\sum_{j'} \exp(s_h(\mathbf{h}_i^{(t)}, \mathbf{h}_{j'}^{(t)}))}, \quad \boldsymbol{\mu}_{j\to i} = a_{j\to i}(\mathbf{h}_i^{(t)} - \mathbf{h}_j^{(t)}) \quad (10)$$

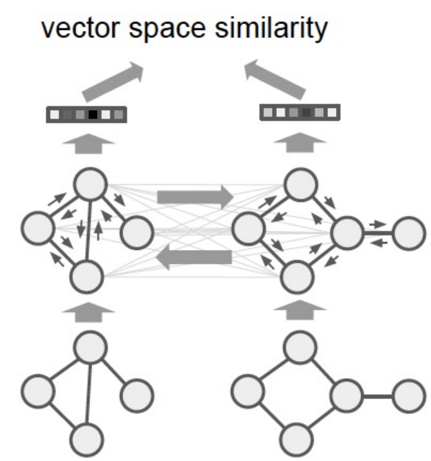
$$\sum_j \boldsymbol{\mu}_{j\to i} = \sum_j a_{j\to i}(\mathbf{h}_i^{(t)} - \mathbf{h}_j^{(t)}) = \mathbf{h}_i^{(t)} - \sum_j a_{j\to i}\mathbf{h}_j^{(t)}. \quad (11)$$

GRAPH MATCHING NETWORK

- T: the number of rounds of propagation
- $\{h_i(\mathrm{T})\}$: the set of node representations ->input
- h_{G1}=f_G $\{h_i(\mathrm{T})\}$, i∈ V_1
- h_{G2}=f_G $\{h_i(\mathrm{T})\}$, i∈ V_2
- f_G: aggregation module
- s = $f_s(h_{G1}, h_{G2})$
- f_s: a standard vector space similarity between h_{G1} and h_{G2} .
- eg, the Euclidean, cosine or Hamming similarities.

$$\mathbf{h}_i^{(t+1)} = f_{\text{node}}\left(\mathbf{h}_i^{(t)}, \sum_j \mathbf{m}_{j\to i}, \sum_{j'} \boldsymbol{\mu}_{j'\to i}\right)$$

$$\begin{aligned}
\mathbf{h}_{G_1} &= f_G(\{\mathbf{h}_i^{(T)}\}_{i\in V_1}) \\
\mathbf{h}_{G_2} &= f_G(\{\mathbf{h}_i^{(T)}\}_{i\in V_2}) \\
s &= f_s(\mathbf{h}_{G_1}, \mathbf{h}_{G_2}).
\end{aligned}$$



OTHER MODELS FOR GRAPH SIMILIARITY LEARNING

- Graph Convolutional Networks (GCNs), which is a simpler variant without modeling edge features
- A: adjacency matrix
- D: degree matrix
- I: identity matrix
- A^=A+I $\tilde{A} = A + I_N$
- D^ : diagonal node degree matrix of A^. $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$

$$f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

$$Z = f(X, A) = \mathrm{softmax}\left(\hat{A}\ \mathrm{ReLU}\left(\hat{A}XW^{(0)}\right)W^{(1)}\right)$$

- Siamese networks:
  - instead of using Euclidean or Hamming distance, learn a distance score through a neural net
  - d(G1, G2) = MLP(concat(embed(G1), embed(G2)))
  - learn the embedding model and the scoring MLP jointly

GRAPH SIMILARITY LEARNING

Margin-based pairwise loss: Euclidean similarity $d(G_1, G_2) = \|\mathbf{h}_{G_1} - \mathbf{h}_{G_2}\|^2$

$$L_{\text{pair}} = \mathbb{E}_{(G_1,G_2,t)}[\max\{0, \gamma - t(1 - d(G_1, G_2))\}], \quad (12)$$

$t \in \{-1, 1\}$

$\gamma > 0$

t = +1 ⇒ G1, G2 similar ⇒ d(G1, G2) ↙
t = -1 ⇒ G1, G2 not similar ⇒ d(G1, G2) ↗

This loss encourages d(G1, G2) < 1−γ when the pair is similar (t = 1), d(G1, G2) > 1+γ when t = −1.

$$L_{\text{triplet}} = \mathbb{E}_{(G_1,G_2,G_3)}[\max\{0, d(G_1, G_2) - d(G_1, G_3) + \gamma\}]. \quad (13)$$

G1, G2 similar, G1, G3 not similar
⇒ d(G1, G2) ↙ d(G1, G3) ↗

GRAPH SIMILARITY LEARNING

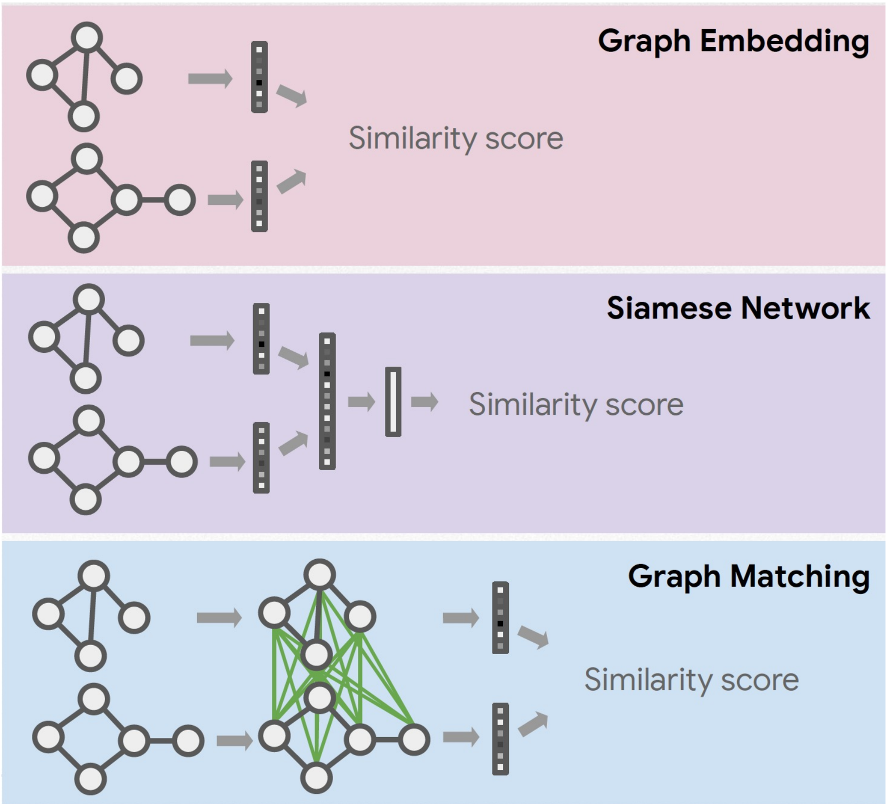
Binary graph representation-> Hamming similarity

$$s(G_1, G_2) = \frac{1}{H} \sum_{i=1}^{H} \tanh(h_{G_1 i}) \cdot \tanh(h_{G_2 i}) \qquad \mathbf{h}_G \in \{-1, 1\}^H$$

$$L_{\text{pair}} = \mathbb{E}_{(G_1, G_2, t)}[(t - s(G_1, G_2))^2]/4, \quad \text{and} \quad (14)$$

$$L_{\text{triplet}} = \mathbb{E}_{(G_1, G_2, G_3)}[(s(G_1, G_2) - 1)^2 + (s(G_1, G_3) + 1)^2]/8, \quad (15)$$

COMPARISON



d(G1, G2) = Euclidean/Hamming distance(embed(G1), embed(G2))

d(G1, G2) = MLP(concat(embed(G1), embed(G2)))

h1, h2 = embed-and-match(G1, G2)

d(G1, G2) = Euclidean/Hamming distance(h1, h2)

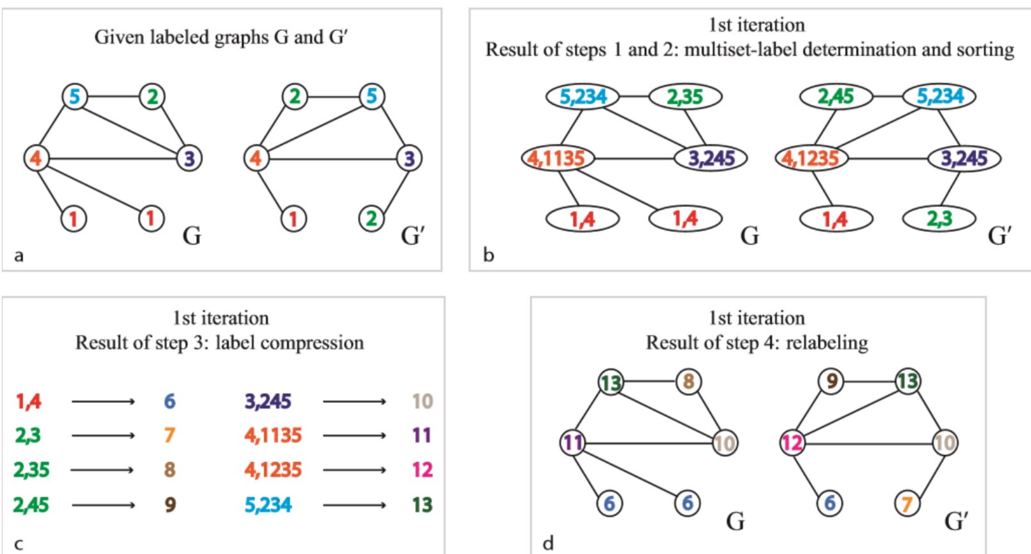
EXPERIMENTS

- Graph edit distance learning
  - Data: synthetic graphs
  - Similarity: small edit distance → similar

- Control-flow graph based binary function similarity search
  - Data: compile ffmpeg with different compilers and optimization levels.
  - Similarity: binary functions associated with the same original function → similar

- Mesh graph retrieval
  - Data: mesh graphs for 100 object classes (COIL-DEL dataset)
  - Similarity: mesh for the same object class → similar

WL KERNEL



- Baseline: WL Kernel
- Weisfeiler Lehman algorithm behind this kernel is a strong method for checking graph isomorphism (edit distance of 0)

Algorithm 1 One iteration of the 1-dim. Weisfeiler-Lehman test of graph isomorphism

1: Multiset-label determination
- For $i = 0$, set $M_i(v) := l_0(v) = \ell(v)$. [2]
- For $i > 0$, assign a multiset-label $M_i(v)$ to each node v in G and G' which consists of the multiset $\{l_{i-1}(u) | u \in \mathcal{N}(v)\}$.

2: Sorting each multiset
- Sort elements in $M_i(v)$ in ascending order and concatenate them into a string $s_i(v)$.
- Add $l_{i-1}(v)$ as a prefix to $s_i(v)$ and call the resulting string $s_i(v)$.

3: Label compression
- Sort all of the strings $s_i(v)$ for all v from G and G' in ascending order.
- Map each string $s_i(v)$ to a new compressed label, using a function $f : \Sigma^* \to \Sigma$ such that $f(s_i(v)) = f(s_i(w))$ if and only if $s_i(v) = s_i(w)$.

4: Relabeling
- Set $l_i(v) := f(s_i(v))$ for all nodes in G and G'.

SYNTHETIC TASK: GRAPH EDIT DISTANCE LEARNING

- Training and evaluating on graphs of size n, and edge density (probability) p
- Measuring pair classification AUC / triplet prediction accuracy.

Graph Distribution	WL kernel	GNN	GMN
$n = 20, p = 0.2$	80.8 / 83.2	88.8 / 94.0	**95.0 / 95.6**
$n = 20, p = 0.5$	74.5 / 78.0	92.1 / 93.4	**96.6 / 98.0**
$n = 50, p = 0.2$	93.9 / **97.8**	95.9 / 97.2	**97.4** / 97.6
$n = 50, p = 0.5$	82.3 / 89.0	88.5 / 91.0	**93.8 / 92.6**

Comparing the graph embedding (GNN) and matching(GMN) models trained on graphs from different distributions with the baseline, measuring pair AUC / triplet accuracy (100).

- Learned models do better than WL kernel.
- Matching model better than embedding model.

BINARY FUNCTION SIMILARITY SEARCH

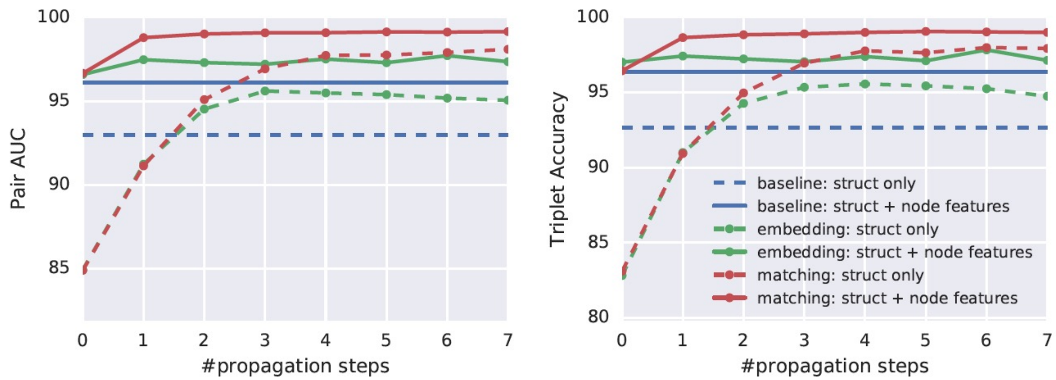


Figure 4. Performance ($\times 100$) of different models on the binary function similarity search task.

Model	Pair AUC	Triplet Acc
Baseline	96.09	96.35
GCN	96.67	96.57
Siamese-GCN	97.54	97.51
GNN	97.71	97.83
Siamese-GNN	97.76	97.58
GMN	**99.28**	**99.18**

Function Similarity Search

Model	Pair AUC	Triplet Acc
GCN	94.80	94.95
Siamese-GCN	95.90	96.10
GNN	98.58	98.70
Siamese-GNN	98.76	98.55
GMN	**98.97**	**98.80**

COIL-DEL

Table 2. More results on the function similarity search task and the extra COIL-DEL dataset.

REFERANCES

- Y. Li al. Graph Matching Networks for Learning the Similarity of Graph Structured Objects. ICML, 2019
- Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. Gated graph sequence neural networks. arXiv preprint arXiv:1511.05493, 2015.
- Vishwanathan, S. V. N., Schraudolph, N. N., Kondor, R., and Borgwardt, K. M. Graph kernels. Journal of Machine Learning Research, 11(Apr):1201–1242, 2010.
- Xu, K., Hu, W., Leskovec, J., and Jegelka, S. Howpowerful are graph neural networks? arXiv preprint arXiv:1810.00826, 2018.