Graph Edit Distance Computation via Graph Neural Networks

Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, Wei Wang

Presenter: Jack Lanchantin
https://qdata.github.io/deep2Read
Outline

1. Introduction
2. Background
3. SimGNN
4. Experiments and Results
5. Conclusion
Outline

1 Introduction

2 Background

3 SimGNN

4 Experiments and Results

5 Conclusion
Graph similarity: finding a graph most similar to query graph (e.g. finding the chemical compounds similar to query compound).
SimGNN

- Graph similarity/distance computation, such as Graph Edit Distance and Maximum Common Subgraph, are the core operation of graph similarity search, but very costly to compute.
- The proposed approach, called SimGNN, uses two steps:
  - First, a learnable embedding function is used to map every graph into an embedding vector for each node (or a global embedding vector).
  - Second, a graph similarity function is used to compare the two graphs.
Task

Graph Edit Distance Computation via Graph Neural Networks

Presenter: Jack Lanchantin

https://qdata.github.io/deep2Read
The GED between the graph to the left and the graph to the right is 3, as the transformation needs 3 edit operations: (1) an edge deletion, (2) an edge insertion, and (3) a node relabeling.
Outline

1. Introduction
2. Background
3. SimGNN
4. Experiments and Results
5. Conclusion
SimGNN

Node-Level Embeddings

Graph-Level Embeddings

Graph-Graph Interactions

Pairwise Node Comparison

Neural Tensor Network

Fully Connected Layers

Predicted Similarity Score

GCNs

Att

GCNs

Att

Neural Tensor Network

Pairwise Node Comparison

Node-Level Embeddings

Graph-Level Embeddings

Graph-Graph Interactions

Yunsheng Bai, Hao Ding, Song Bian, Ting Ci

Graph Edit Distance Computation via Graph Neural Networks

Presenter: Jack Lanchantin

https://qdata.github.io/deep2Read
Stage I: Node Embedding

- 3 Layer GCN to produce node embeddings
- Nodes, denoted by $u_n \in \mathbb{R}^D$, are updated as follows:

$$
\text{conv}(u_n) = \sigma\left( \sum_{m \in \mathcal{N}(n)} \frac{1}{\sqrt{d_n d_m}} u_m W_{1}^{(l)} \right)
$$

(1)

$\mathcal{N}(n)$ is the first-order neighbors of node $n$, $d_n$ is the degree of node $n$, $W_1^{(l)} \in \mathbb{R}^{D_l \times D_l+1}$ is the weight matrix for the $l$-th GCN layer.
Stage II: Graph Embedding: Global Context-Aware Attention

- Given node embeddings $u_n \in \mathbb{R}^D$, global graph context $c \in \mathbb{R}^D$ is computed, which is an average of node embeddings followed by tanh:

$$c = \tanh\left(\left(\frac{1}{N} \sum_{n=1}^{N} u_n\right) W_2\right), \quad (2)$$

where $W_2 \in \mathbb{R}^{D \times D}$ is a learnable weight matrix.
Stage II: Graph Embedding: Global Context-Aware Attention

- Based on $c$, we can compute one attention weight for each node embedding $a_n$ by computing an inner product with $c$.
- Graph embedding $h \in \mathbb{R}^D$ is the weighted sum of node embeddings:

$$h = \sum_{n=1}^{N} f_2(u_n^T c)u_n$$

(3)

where $f_2(\cdot)$ is the **sigmoid function** $\sigma(\cdot)$, i.e. not normalized.
Instead of using inner product between graph embeddings, $h_i \in \mathbb{R}^D$, $h_j \in \mathbb{R}^D$ we use Neural Tensor Networks to model the relations:

$$g(h_i, h_j) = \sigma(h_i^T W_3^{[1:K]} h_j + V \left[ \begin{array}{c} h_i \\ h_j \end{array} \right])$$

(4)

where $W_3^{[1:K]} \in \mathbb{R}^{D \times D \times K}$ is a weight tensor, $\left[ \right]$ denotes the concatenation operation, $V \in \mathbb{R}^{K \times 2D}$ is a weight vector, $K$ is a hyperparameter controlling the number of interaction (similarity) scores for each graph embedding pair.
Stage IV: Graph Similarity Score Computation

- MLP used to output final similarity score from neural tensor vector.
- \( \hat{s}_{ij} \in \mathbb{R} \) is predicted and compared against the ground-truth similarity score \( s(G_i, G_j) \):

\[
\mathcal{L} = \frac{1}{|\mathcal{D}|} \sum_{(i,j) \in \mathcal{D}} (\hat{s}_{ij} - s(G_i, G_j))^2
\]  

where \( \mathcal{D} \) is the set of training graph pairs, and \( s(G_i, G_j) \) is the ground-truth similarity between \( G_i \) and \( G_j \).
Limitations of Strategy One

- The node-level information such as the node feature distribution and graph size may be lost by the graph-level embedding.
- In many cases, differences between two graphs lie in small substructures and can’t be reflected by a graph level embedding.
- To overcome these limitations, consider using the node-level embeddings directly.
If $G_i$ has $N_i$ nodes and $G_j$ has $N_j$ nodes, there would be $N_i N_j$ pairwise interaction scores, obtained by $S = \sigma(U_i U_j^T)$, where $U_i \in \mathbb{R}^{N_i \times D}$ and $U_j \in \mathbb{R}^{N_j \times D}$ are the node embeddings of $G_i$ and $G_j$, respectively.
Stage III: Graph-Graph Interaction: Pairwise Node Comparison (Strategy II)

- If $G_i$ has $N_i$ nodes and $G_j$ has $N_j$ nodes, there would be $N_iN_j$ pairwise interaction scores, obtained by $S = \sigma(U_i U_j^T)$, where $U_i \in \mathbb{R}^{N_i \times D}$ and $U_j \in \mathbb{R}^{N_j \times D}$ are the node embeddings of $G_i$ and $G_j$, respectively.

- One simple way to utilize $S$ is to vectorize it: $\text{vec}(S) \in \mathbb{R}^{N_iN_j}$, and feed to the fully connected layers. However, there is usually no natural ordering between graph nodes.
To ensure the model is invariant to the graph representations, we extract its histogram features: \( \text{hist}(\mathbf{S}) \in \mathbb{R}^B \), where \( B \) is a hyperparameter that controls the number of bins in the histogram.

The histogram feature vector is normalized and concatenated with the graph-level interaction scores \( g(h_i, h_j) \), and fed to the MLP.
Outline

1. Introduction
2. Background
3. SimGNN
4. Experiments and Results
5. Conclusion
## Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Graph Meaning</th>
<th>#Graphs</th>
<th>#Pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIDS</td>
<td>Chemical Compounds</td>
<td>700</td>
<td>490K</td>
</tr>
<tr>
<td>LINUX</td>
<td>Program Dependency Graphs</td>
<td>1000</td>
<td>1M</td>
</tr>
<tr>
<td>IMDB</td>
<td>Actor/Actress Ego-Networks</td>
<td>1500</td>
<td>2.25M</td>
</tr>
</tbody>
</table>
## AIDS Chemical Compounds

<table>
<thead>
<tr>
<th>Method</th>
<th>mse((10^{-3}))</th>
<th>(\rho)</th>
<th>(\tau)</th>
<th>p@10</th>
<th>p@20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beam Hungarian VJ</td>
<td>12.090</td>
<td>0.609</td>
<td>0.463</td>
<td>0.481</td>
<td>0.493</td>
</tr>
<tr>
<td>SimpleMean</td>
<td>3.115</td>
<td>0.633</td>
<td>0.480</td>
<td>0.269</td>
<td>0.279</td>
</tr>
<tr>
<td>HierarchicalMean</td>
<td>3.046</td>
<td>0.681</td>
<td>0.629</td>
<td>0.246</td>
<td>0.340</td>
</tr>
<tr>
<td>HierarchicalMax</td>
<td>3.396</td>
<td>0.655</td>
<td>0.505</td>
<td>0.222</td>
<td>0.295</td>
</tr>
<tr>
<td>AttDegree</td>
<td>3.338</td>
<td>0.628</td>
<td>0.478</td>
<td>0.209</td>
<td>0.279</td>
</tr>
<tr>
<td>AttGlobalContext</td>
<td>1.472</td>
<td>0.813</td>
<td>0.653</td>
<td>0.376</td>
<td>0.473</td>
</tr>
<tr>
<td>AttLearnableGC</td>
<td>1.340</td>
<td>0.825</td>
<td>0.667</td>
<td>0.400</td>
<td>0.488</td>
</tr>
<tr>
<td>SimGNN</td>
<td><strong>1.189</strong></td>
<td><strong>0.843</strong></td>
<td><strong>0.690</strong></td>
<td><strong>0.421</strong></td>
<td><strong>0.514</strong></td>
</tr>
</tbody>
</table>
### Program Dependence Graphs generated from the Linux kernel

**Graph Edit Distance Computation via Graph Neural Networks**

**Presenter:** Jack Lanchantin

[https://qdata.github.io/deep2Read](https://qdata.github.io/deep2Read)

<table>
<thead>
<tr>
<th>Method</th>
<th>mse($10^{-3}$)</th>
<th>$\rho$</th>
<th>$\tau$</th>
<th>p@10</th>
<th>p@20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beam</td>
<td>12.090</td>
<td>0.609</td>
<td>0.463</td>
<td>0.481</td>
<td>0.493</td>
</tr>
<tr>
<td>Hungarian</td>
<td>25.296</td>
<td>0.510</td>
<td>0.378</td>
<td>0.360</td>
<td>0.392</td>
</tr>
<tr>
<td>VJ</td>
<td>29.157</td>
<td>0.517</td>
<td>0.383</td>
<td>0.310</td>
<td>0.345</td>
</tr>
<tr>
<td>SimpleMean</td>
<td>3.115</td>
<td>0.633</td>
<td>0.480</td>
<td>0.269</td>
<td>0.279</td>
</tr>
<tr>
<td>HierarchicalMean</td>
<td>3.046</td>
<td>0.681</td>
<td>0.629</td>
<td>0.246</td>
<td>0.340</td>
</tr>
<tr>
<td>HierarchicalMax</td>
<td>3.396</td>
<td>0.655</td>
<td>0.505</td>
<td>0.222</td>
<td>0.295</td>
</tr>
<tr>
<td>AttDegree</td>
<td>3.338</td>
<td>0.628</td>
<td>0.478</td>
<td>0.209</td>
<td>0.279</td>
</tr>
<tr>
<td>AttGlobalContext</td>
<td>1.472</td>
<td>0.813</td>
<td>0.653</td>
<td>0.376</td>
<td>0.473</td>
</tr>
<tr>
<td>AttLearnableGC</td>
<td>1.340</td>
<td>0.825</td>
<td>0.667</td>
<td>0.400</td>
<td>0.488</td>
</tr>
<tr>
<td>SimGNN</td>
<td><strong>1.189</strong></td>
<td><strong>0.843</strong></td>
<td><strong>0.690</strong></td>
<td><strong>0.421</strong></td>
<td><strong>0.514</strong></td>
</tr>
</tbody>
</table>
### Table 4: Results on IMDB. Beam, Hungarian, and VJ together are used to determine the ground-truth results.

<table>
<thead>
<tr>
<th>Method</th>
<th>$\text{mse}(10^{-3})$</th>
<th>$\rho$</th>
<th>$\tau$</th>
<th>p@10</th>
<th>p@20</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleMean</td>
<td>3.749</td>
<td>0.774</td>
<td>0.644</td>
<td>0.547</td>
<td>0.588</td>
</tr>
<tr>
<td>HierarchicalMean</td>
<td>5.019</td>
<td>0.456</td>
<td>0.378</td>
<td>0.567</td>
<td>0.553</td>
</tr>
<tr>
<td>HierarchicalMax</td>
<td>6.993</td>
<td>0.455</td>
<td>0.354</td>
<td>0.572</td>
<td>0.570</td>
</tr>
<tr>
<td>AttDegree</td>
<td>2.144</td>
<td>0.828</td>
<td>0.695</td>
<td>0.700</td>
<td>0.695</td>
</tr>
<tr>
<td>AttGlobalContext</td>
<td>3.555</td>
<td>0.684</td>
<td>0.553</td>
<td>0.657</td>
<td>0.656</td>
</tr>
<tr>
<td>AttLearnableGC</td>
<td>1.455</td>
<td>0.835</td>
<td>0.700</td>
<td>0.732</td>
<td>0.742</td>
</tr>
<tr>
<td>SimGNN</td>
<td>1.264</td>
<td>0.878</td>
<td>0.770</td>
<td>0.759</td>
<td>0.777</td>
</tr>
</tbody>
</table>
Conclusion

- Used GNN for embedding entire graphs to do graph similarity matching (e.g. querying similar graphs)
- Proposed method is empirically good, but I think the graph embedding and edge similarity parts could be improved
- No edge features used