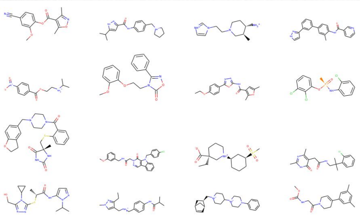
Graph Matching Networks for Learning the Similarity of Graph Structured Objects

Yujia Li, Chenjie Gu, Thomas Dullien*, Oriol Vinyals, Pushmeet Kohli

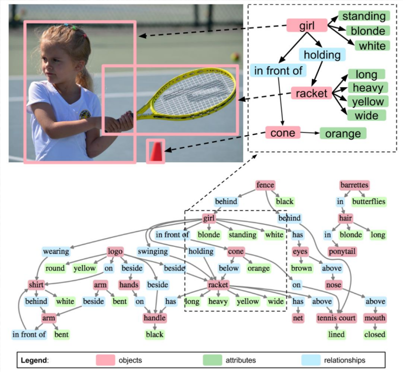
DeepMind

*Google

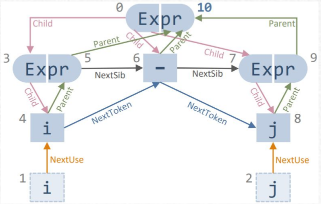
Graph structured data appear in many applications



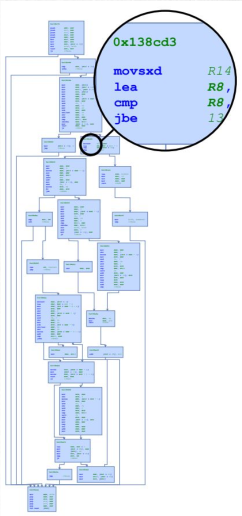
Molecules



Scene Graphs*



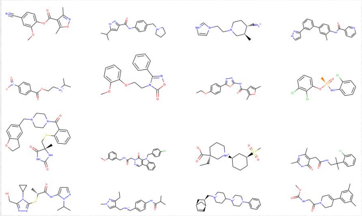
Programs**



Binaries

Image credit: *Johnson et al. Image Retrieval using Scene Graphs. **Brockschmidt et al. Generative Code Modeling with Graphs

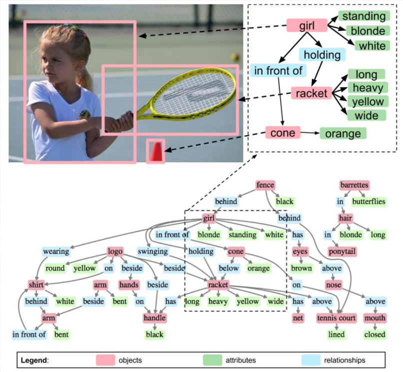
Graph structured data appear in many applications



Molecules

↓

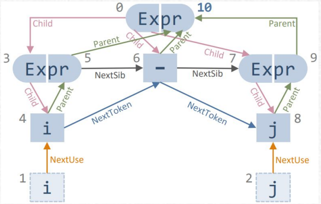
Drug Discovery



Scene Graphs*

↓

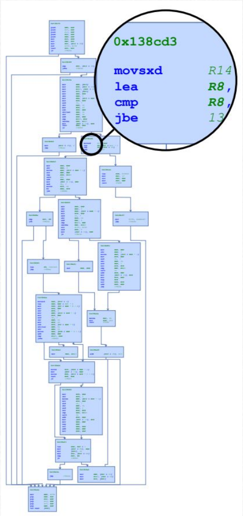
Semantic Image Retrieval



Programs**

↓

Code Search



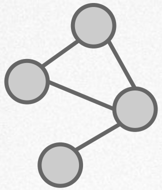
Binaries

↓

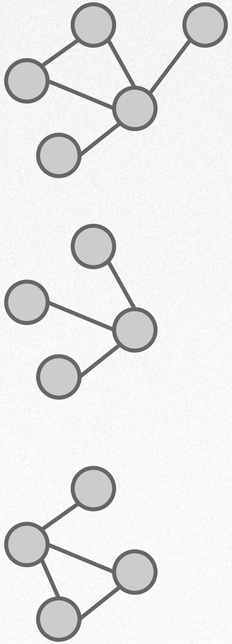
Software Vulnerabilities

Image credit: *Johnson et al. Image Retrieval using Scene Graphs. **Brockschmidt et al. Generative Code Modeling with Graphs

Finding similar graphs



Query Graph



Candidate Graphs

Graph structures vary a lot

Nodes and edges can have attributes

Reasoning about both the graph **structure** and the **semantics**

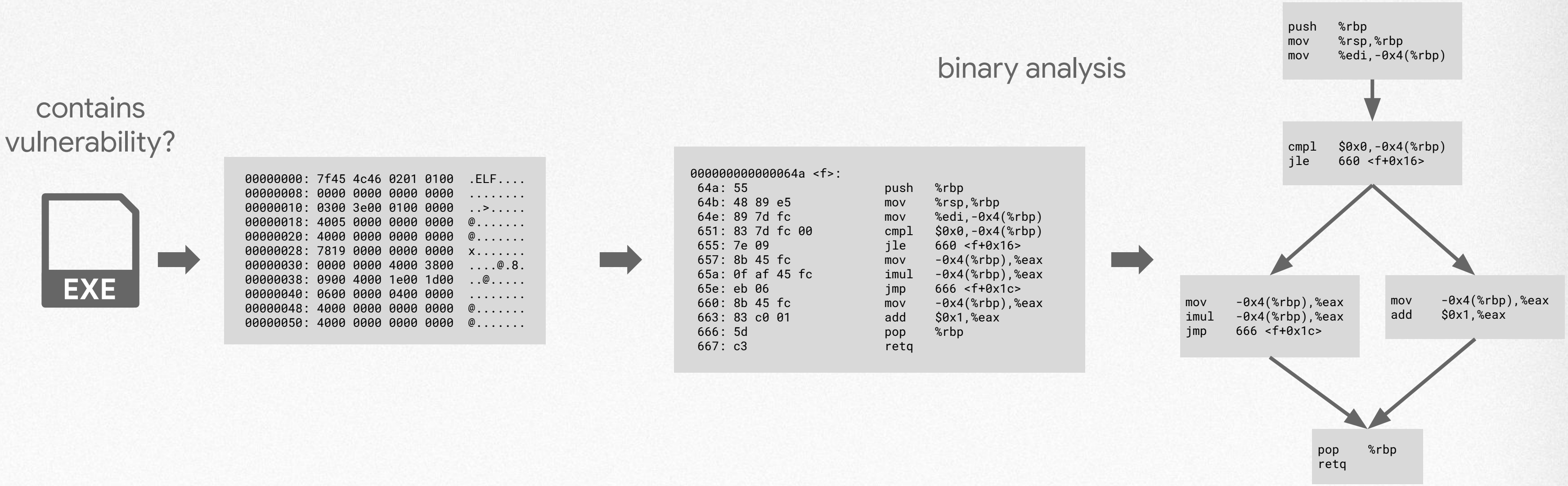
The notion of “similarity” varies across problems

The binary function similarity search problem

contains vulnerability?


```
00000000: 7f45 4c46 0201 0100  .ELF....
00000008: 0000 0000 0000 0000  ........
00000010: 0300 3e00 0100 0000  ..>.....
00000018: 4005 0000 0000 0000  @.......
00000020: 4000 0000 0000 0000  @.......
00000028: 7819 0000 0000 0000  x.......
00000030: 0000 0000 4000 3800  ....@.8.
00000038: 0900 4000 1e00 1d00  ..@.....
00000040: 0600 0000 0400 0000  ........
00000048: 4000 0000 0000 0000  @.......
00000050: 4000 0000 0000 0000  @.......
```

The binary function similarity search problem



graph sizes in our dataset: from 10 to 10^3

The binary function similarity search problem

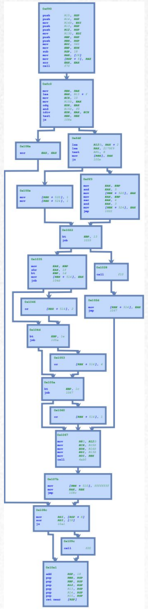
search in a library of binaries
with known vulnerabilities

contains
vulnerability?

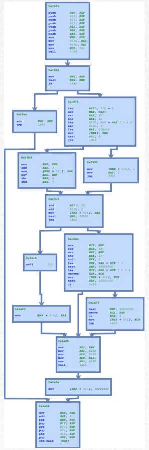
EXE

```
00000000: 7f45 4c46 0201 0100  .ELF....
00000008: 0000 0000 0000 0000  ........
00000010: 0300 3e00 0100 0000  ..>.....
00000018: 4005 0000 0000 0000  @.......
00000020: 4000 0000 0000 0000  @.......
00000028: 7819 0000 0000 0000  x.......
00000030: 0000 0000 4000 3800  ....@.8.
00000038: 0900 4000 1e00 1d00  ..@.....
00000040: 0600 0000 0400 0000  ........
00000048: 4000 0000 0000 0000  @.......
00000050: 4000 0000 0000 0000  @.......
```

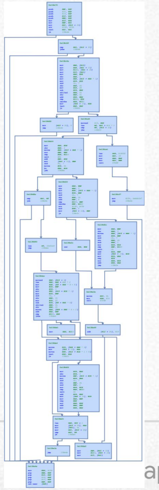
binary
analysis



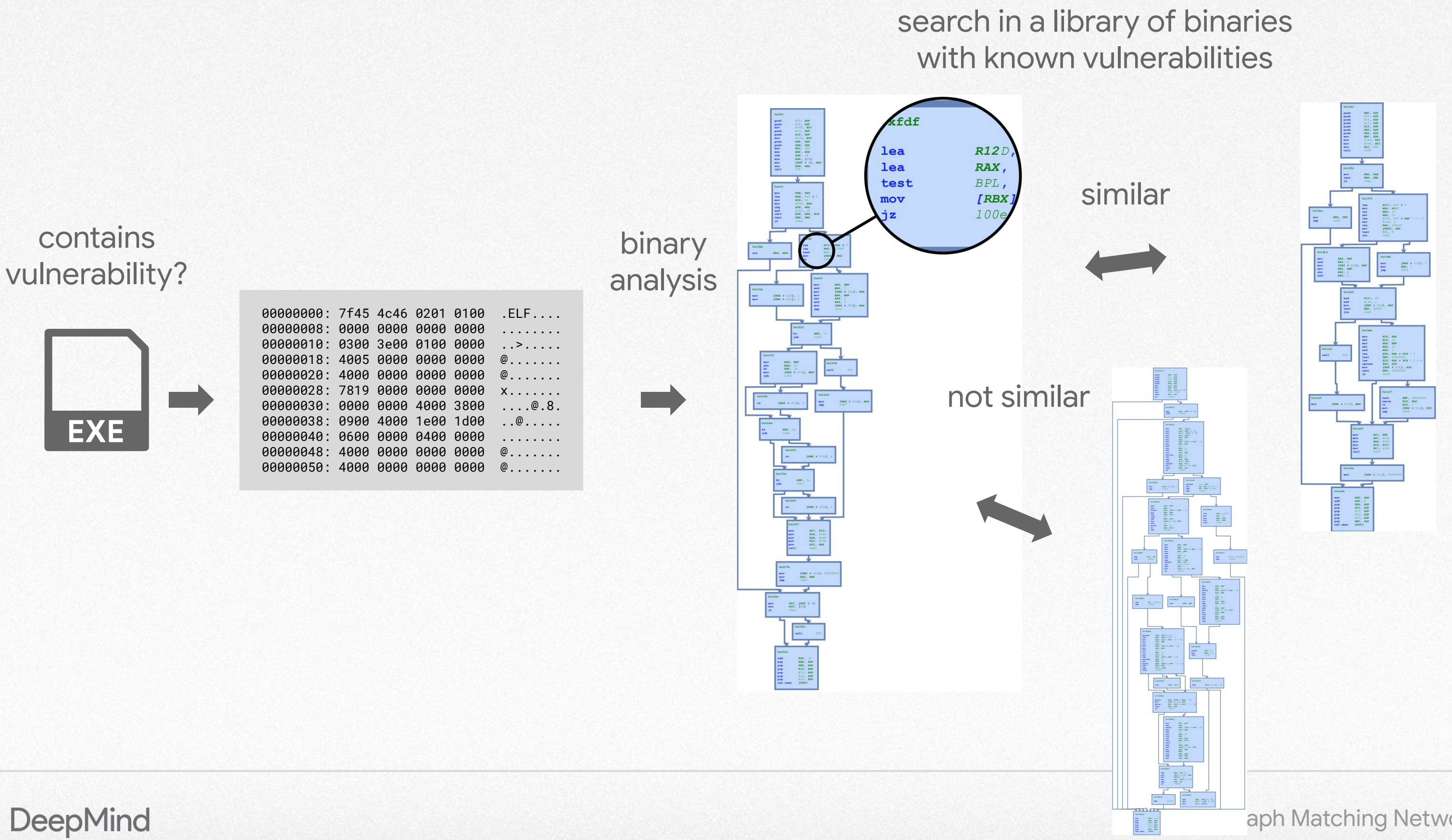
similar



not similar



The binary function similarity search problem



Most existing approaches

Mostly hand-engineered algorithms / heuristics with limited learning:

Graph hashes (graph ⟶ descriptor): widely used in security applications

- human-designed hash functions that encode graph structure
- good at exact matches, not so good at estimating similarity

Graph kernels (pair of graphs ⟶ similarity): popular in various graph-level prediction tasks

- human-designed kernels as a measure of similarity between graphs
- the design of kernels is important for performance

Different graph similarity estimation paradigms

Graph embedding

Graph ⟶ descriptor

Measure distance on descriptors

Fast hashing based retrieval

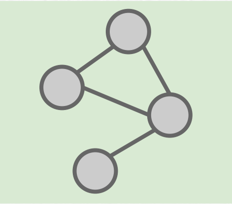
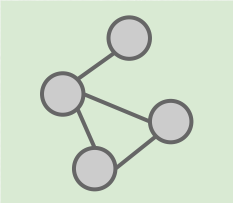
Graph matching

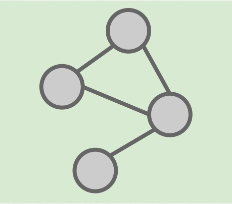
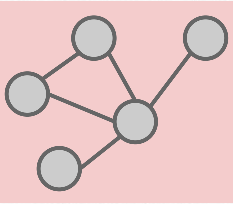
Compute distance jointly on the pair of graphs

More computation for **better accuracy**

Graph similarity learning

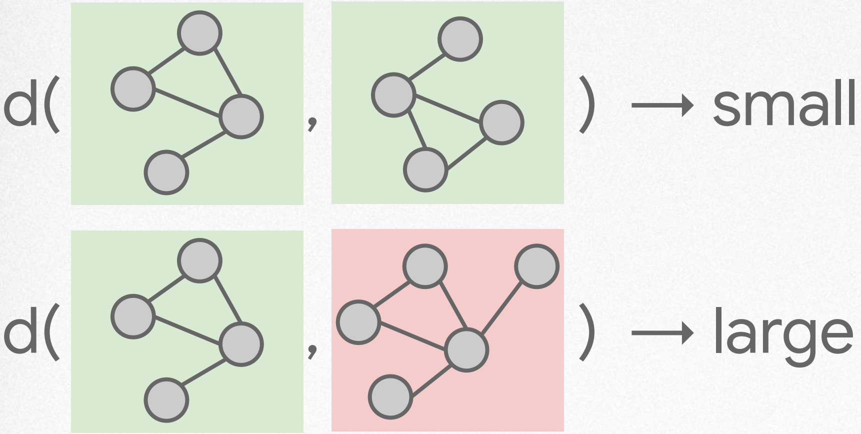
Learn a similarity (or distance) function

d( , ) ⟶ small

d( , ) ⟶ large

Graph similarity learning

Learn a similarity (or distance) function



Supervised learning on labeled **pairs** or **triplets**

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

$t = +1 \Rightarrow G_1, G_2$ similar $\Rightarrow d(G_1, G_2) \searrow$

$t = -1 \Rightarrow G_1, G_2$ not similar $\Rightarrow d(G_1, G_2) \nearrow$

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

G_1, G_2 similar, G_1, G_3 not similar

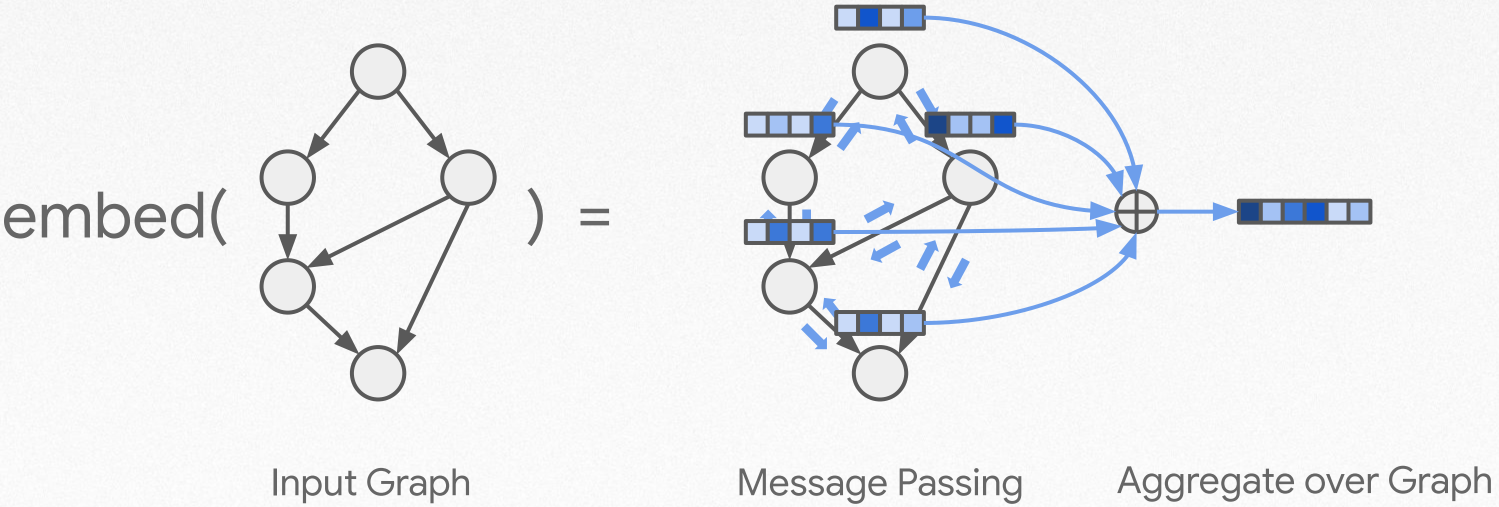
$\Rightarrow d(G_1, G_2) \searrow \quad d(G_1, G_3) \nearrow$

Learning graph embeddings with Graph Neural Nets

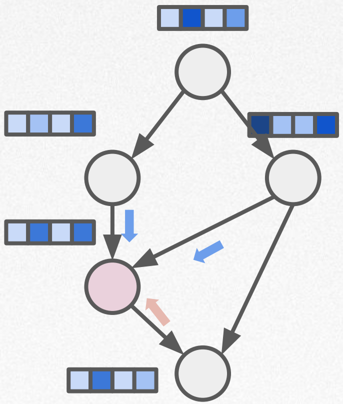
$d(G_1, G_2)$ = Euclidean/Hamming distance(**embed**(G_1), **embed**(G_2))

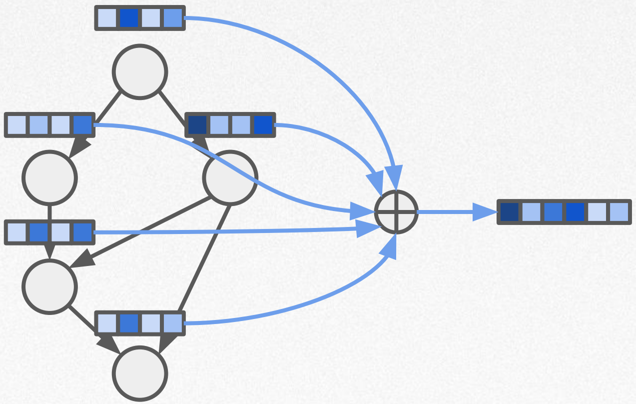
Learning graph embeddings with Graph Neural Nets

$d(G_1, G_2)$ = Euclidean/Hamming distance(**embed**(G_1), **embed**(G_2))



Graph embedding model details





Messages: $$\mathbf{m}_{u\to v} = f_{\text{message}}(\mathbf{h}_u^{(t)}, \mathbf{h}_v^{(t)}, \mathbf{e}_{uv})$$

Node updates: $$\mathbf{h}_v^{(t+1)} = f_{\text{node}}\left(\mathbf{h}_v^{(t)}, \sum_{u\in N(v)} \mathbf{m}_{u\to v}\right)$$

$$\mathbf{h}_G = \text{MLP}\left(\text{POOL}(\{\mathbf{h}_v\}_{v\in V})\right)$$

Aggregation:
sum pooling, attention pooling etc.

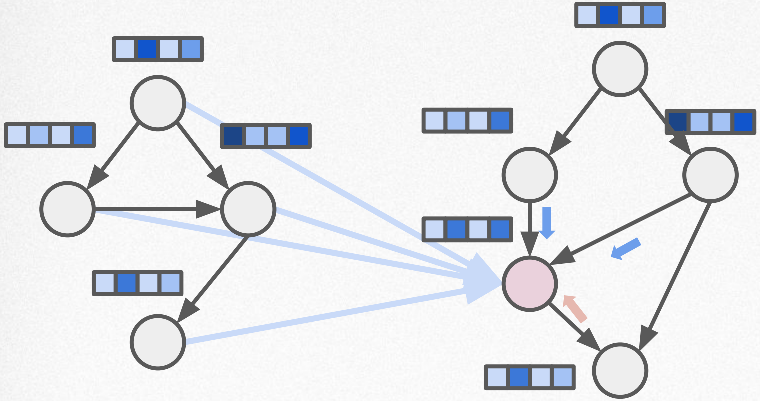
Graph Matching Networks

$h_1, h_2 =$ **embed-and-match**(G_1, G_2)

$d(G_1, G_2) =$ Euclidean/Hamming distance(h_1, h_2)

Graph Matching Networks

$h_1, h_2 =$ **embed-and-match**(G_1, G_2)
$d(G_1, G_2) =$ Euclidean/Hamming distance(h_1, h_2)



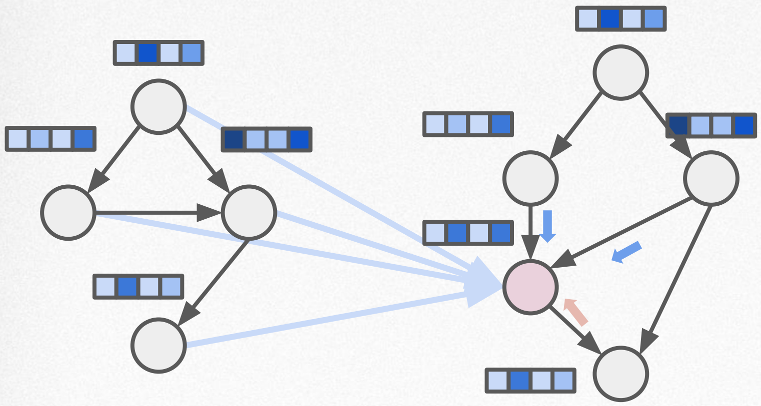
Attention: $a_{j \to i} = \text{Softmax}_j \left(s(\mathbf{h}_i^{(t)}, \mathbf{h}_j^{(t)}) \right)$

Weighted difference: $\boldsymbol{\mu}_{j \to i} = a_{j \to i}(\mathbf{h}_i^{(t)} - \mathbf{h}_j^{(t)})$

Graph Matching Networks

h_1, h_2 = **embed-and-match**(G_1, G_2)

$d(G_1, G_2)$ = Euclidean/Hamming distance(h_1, h_2)



Total cross-graph message

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

Effectively: match node i to the closest node in the other graph and take the difference.

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

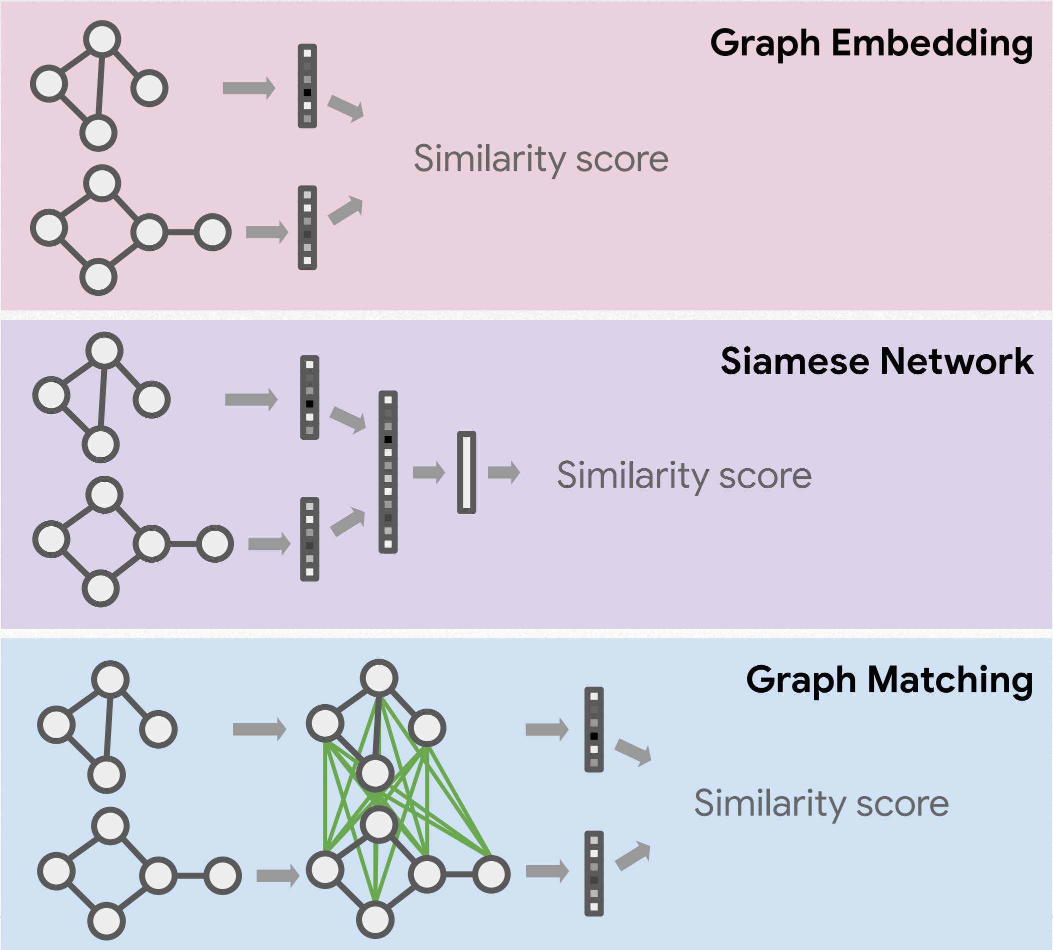
Other variants

Other variants of GNNs for embedding:

- e.g. Graph Convolutional Networks (GCNs), which is a simpler variant without modeling edge features

Siamese networks:

- instead of using Euclidean or Hamming distance, learn a distance score through a neural net
- $d(G_1, G_2) = \mathbf{MLP}(\text{concat}(\mathbf{embed}(G_1), \mathbf{embed}(G_2)))$
- learn the embedding model and the scoring MLP jointly


Graph Embedding
Similarity score
Siamese Network
Similarity score
Graph Matching
Similarity score

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

Synthetic task: graph edit distance learning

Training and evaluating on graphs of size n, and edge density (probability) p

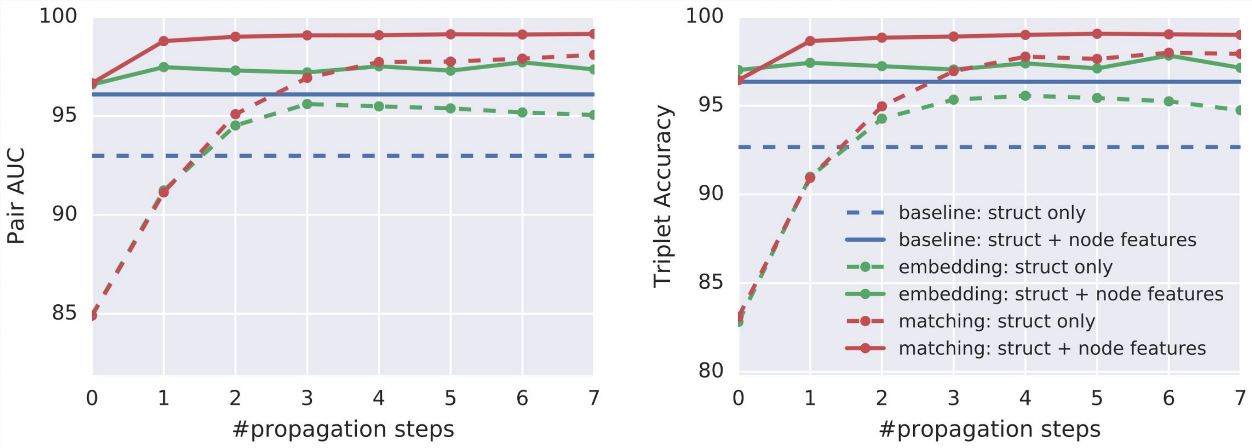
Measuring **pair classification AUC** / **triplet prediction accuracy**.

Graph Spec.	WL kernel	embedding model	matching model
$n = 20, p = 0.2$	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**

Learned models do better than WL kernel.

Matching model better than embedding model.

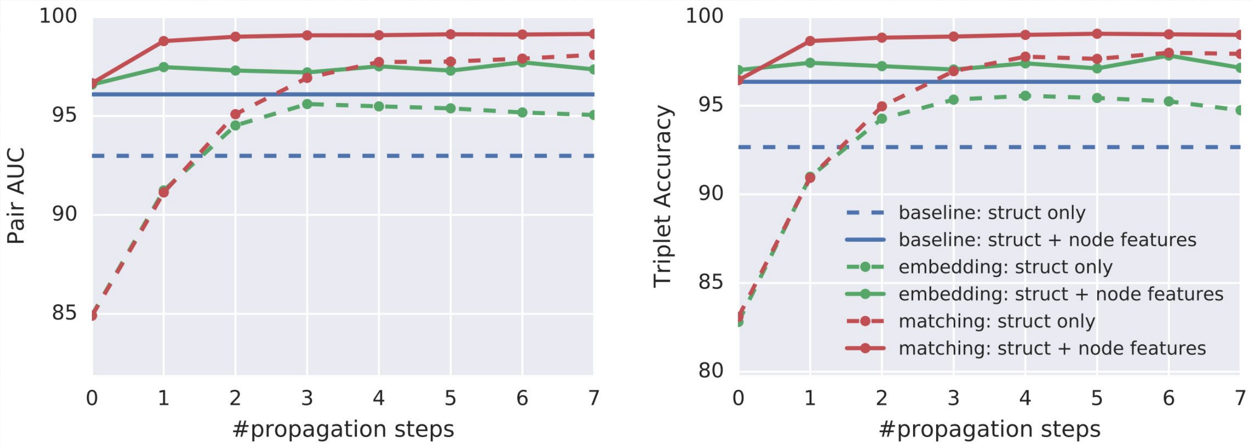
Results on binary function similarity search



Hand-engineered baseline (graph hashing + locality sensitive hashing) vs **GNN embedding** vs **GMN**.

Graph topology only vs jointly over **structures and features.**

Results on binary function similarity search



1) **learned approaches** better than **hand-engineered solution**
2) **matching** better than **embedding** alone
3) joint modeling of **structure and features** better than **structure alone**
4) performance better with **more graph propagation steps**

More ablation studies

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**

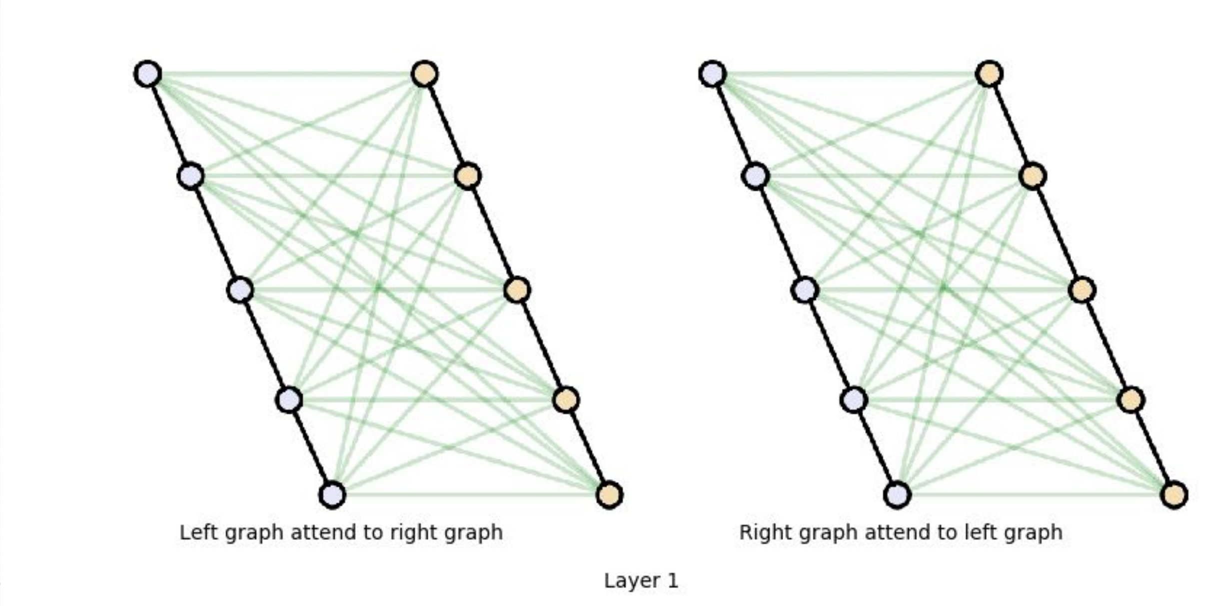
COIL-DEL

GMNs consistently better than alternatives.

Siamese vs **matching**: fusing two graphs early better than only at the end.

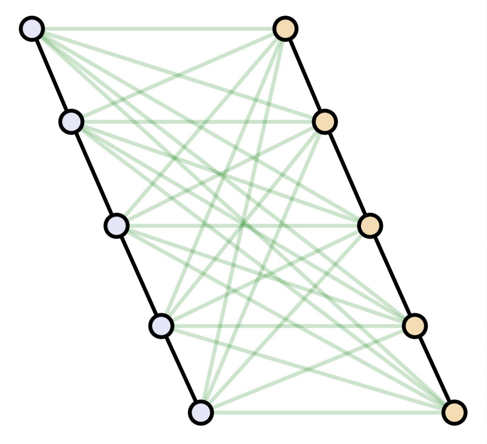
Learned attention patterns

We never supervise the cross-graph attention, but the model still learns some interesting attention patterns.

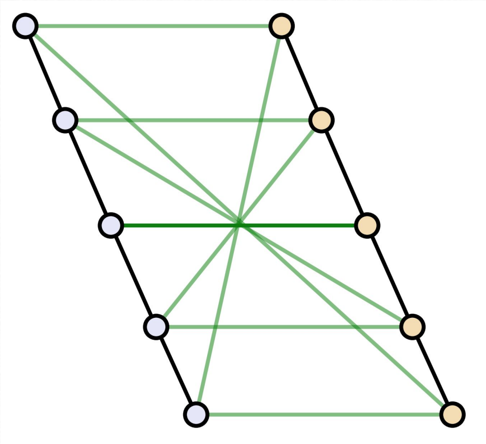


Learned attention patterns

When the two graphs are identical, the learned attention pattern may (not always) correspond to node matching.



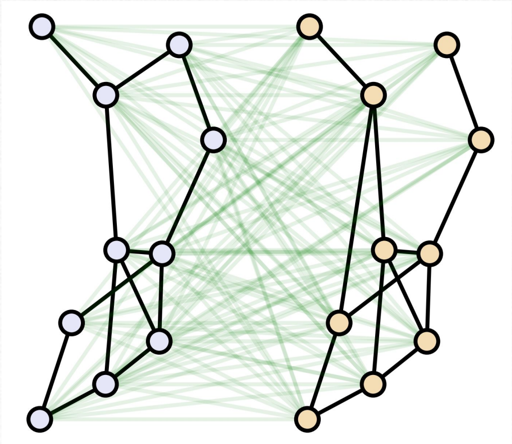
After 10 message passing steps



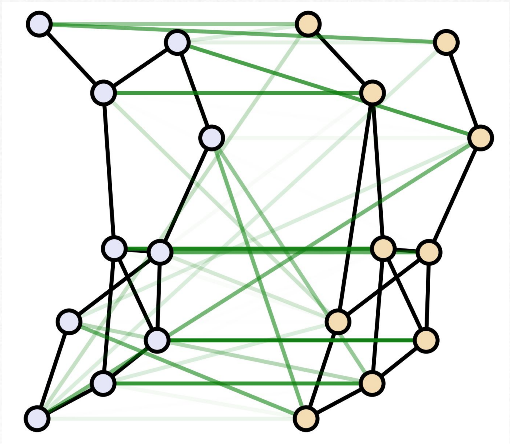
Model trained on the edit distance learning task.

Learned attention patterns

Otherwise the attention pattern is less interpretable.



After 10 message passing steps



Model trained on the edit distance learning task.

Conclusions and future directions

Takeaways:

- graph similarity can be learned.
- learned graph embedding models are good and efficient models for this.
- graph matching networks are even better.

Future directions:

- make cross-graph attention and matching more efficient
- explore new architectures that can utilize the new capability of learned graph similarity

Graph Matching Networks for Learning the Similarity of Graph Structured Objects

Yujia Li, Chenjie Gu, Thomas Dullien*, Oriol Vinyals, Pushmeet Kohli

DeepMind

*Google