Computing Graph Edit Distance via Neural Graph Matching

Chengzhi Piao[1], Tingyang Xu[2], Xiangguo Sun[1],
Yu Rong[2], Kangfei Zhao[2], Hong Cheng[1]

[1]The Chinese University of Hong Kong

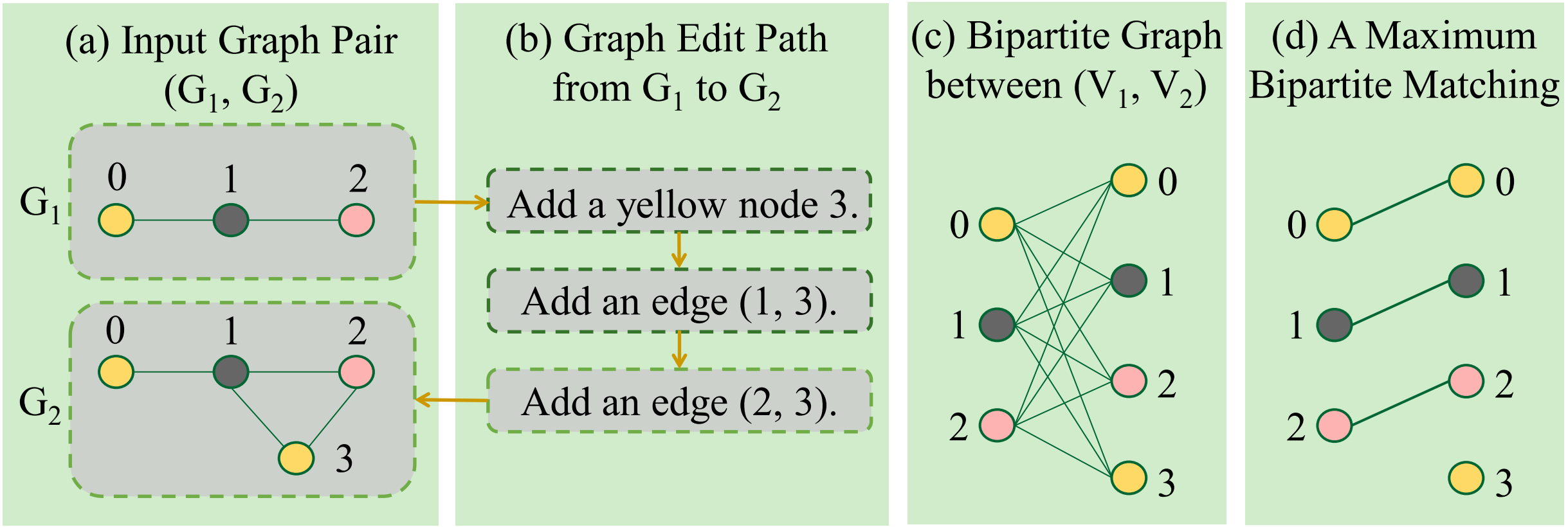
[2]Tencent AI Lab

Computing Graph Edit Distance via Neural Graph Matching

- **Graph Edit Distance**
  - Given a graph pair (G1, G2), the minimum number of edit operations that transforms G1 to G2 is called the graph edit distance, GED(G1, G2).
  - The sequence of edit operations is called an edit path.

- **Task**
  - Graph edit distance (GED) is an NP-hard problem.
  - Inexact GED computation: finding an edit path as short as possible.

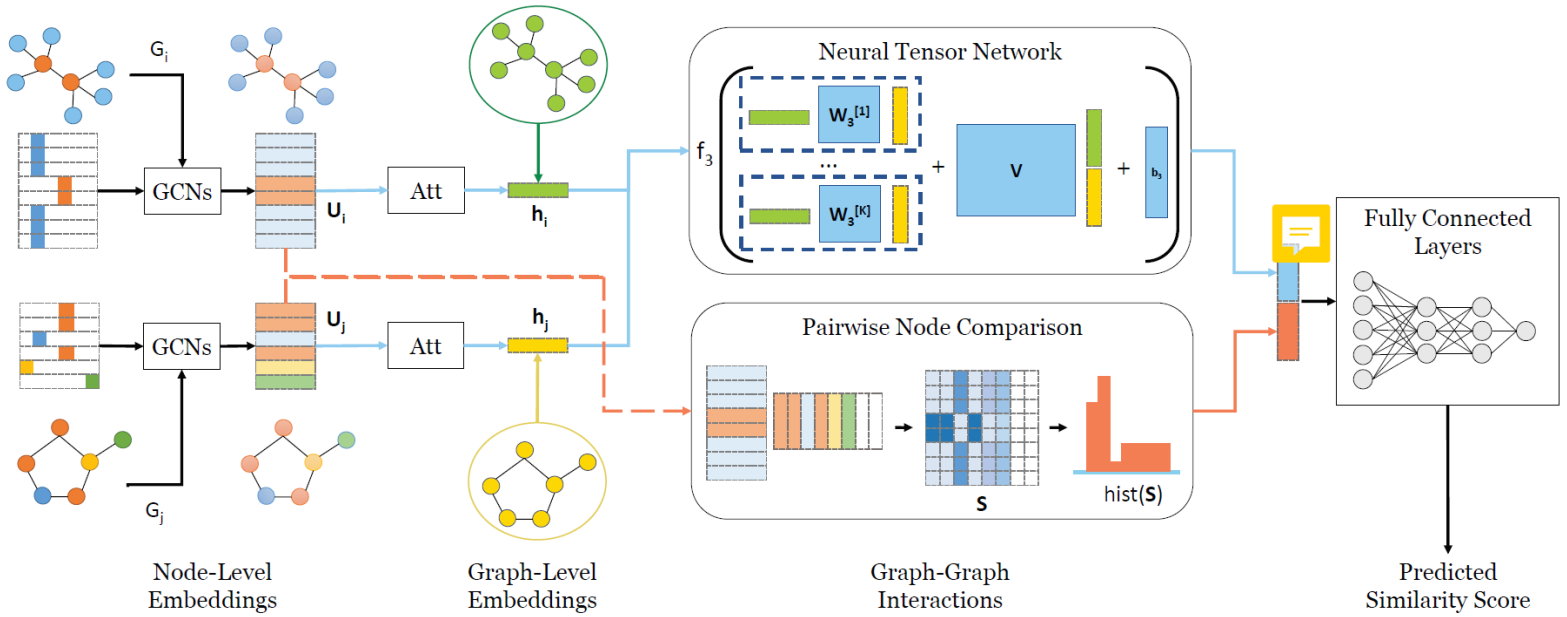
Example



- a, b: An instance of graph edit path.
- c, d: Solving GED via bipartite matching.

Existing Methods

	Time Complexity	Solution Quality	Finding an Edit Path
A*-beam Search	Exponential	Depending on *beamsize*	Yes
Greedy Algorithms	$O(n^3)$	Low	Yes
Learning Models	$O(n+m)$	High	No

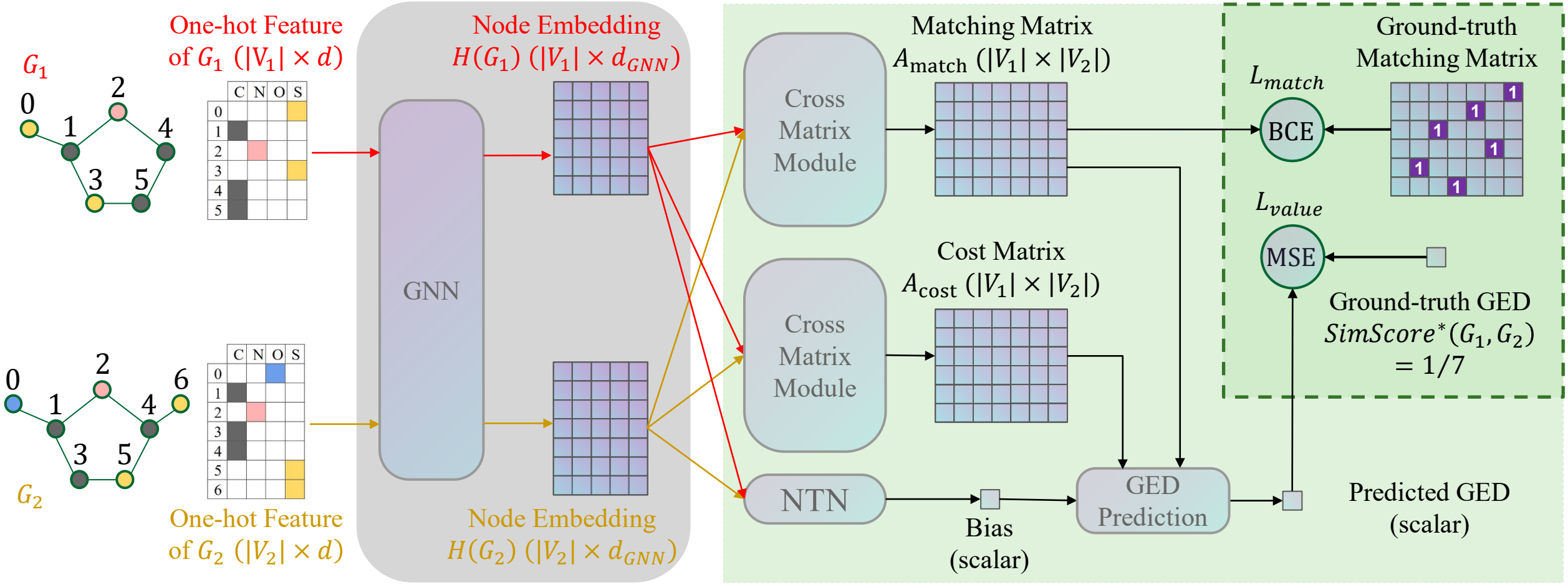


- Edit Path:
  - Interpretability
  - Usability

Solution Overview

- A Two-step Framework:
  - Using GNN to predict a GED and generate a node matching matrix.
  - Post-processing the node matching matrix to find a short edit path.

Step 1: GEDGNN Architecture



Cross Matrix Module

- Input:
  - node embedding H_1 and H_2 of the given graph pair
- Output:
  - A matrix of size $|V_1| \times |V_2|$
- Target:
  - Capture the node-to-node correspondence between H_1 and H_2;
  - Further generate node matching (edit path).
- Structure

$$A = H_1 W H_2^\top.$$

c hidden layers

$$A = [H_1 W_1 H_2^\top, H_1 W_2 H_2^\top, \cdots, H_1 W_c H_2^\top],$$

MLP(c -> 1)

The Matching Matrix

- A matching matrix A_{match} is a prediction of the ground-truth matching matrix and reflects the extent of matching of different node pairs.
- Its generation is supervised by the ground-truth matching matrix M^* . During the training process, a binary cross entropy loss is used to minimize the difference between A_{match} and M^*.

$$\mathcal{L}_{match} = \text{BCELoss}(A_{match}, M_{01}^*)$$

$$= \frac{1}{|V_1| \cdot |V_2|} \sum_{u,v} (M_{01}^*[u][v] \cdot \log A_{match}[u][v]$$

$$+ (1 - M_{01}^*[u][v]) \cdot \log(1 - A_{match}[u][v])).$$

The Cost Matrix

- The cost matrix A_{cost} has size $|V_1| \times |V_2|$ in which an element $A_{cost}[u][v]$ denotes the cost of edit operations for matching $u \in V_1$ with $v \in V_2$.

- With these two matrices, we predict the GED value by:

$$GED(G_1, G_2) = f\left(Softmax(A_{match}) \cdot A_{cost} + bias\right)$$

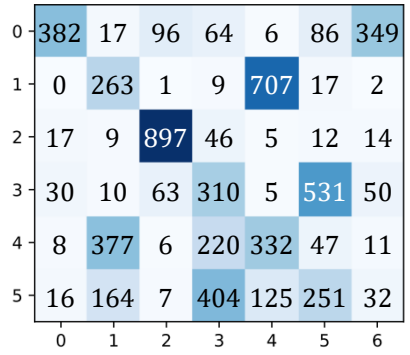
Step 2: Post-processing

Extracting the node matching from A_{match}.

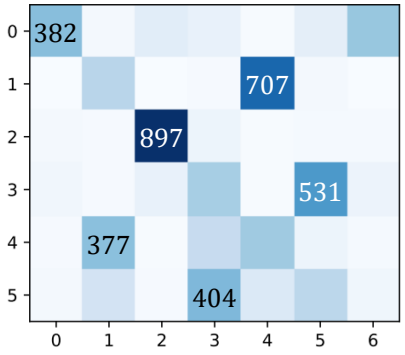
From A_{match} to Node Matching

- Extracting the node matching from A_{match} can be formulated as a weighted bipartite matching problem.
- We can construct a bipartite graph G=(V_1, V_2, V_1xV_2, A_{match}) and then find the maximum weight matching on G.
- To further increase the chance of finding a short edit path, we explore **k-best matching** instead of only the maximum weight one.

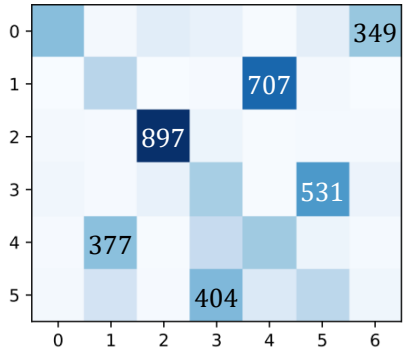
Post-processing Algorithm: k-best Matching



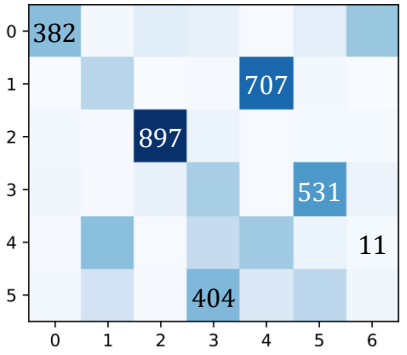
Matching Matrix A_{match}



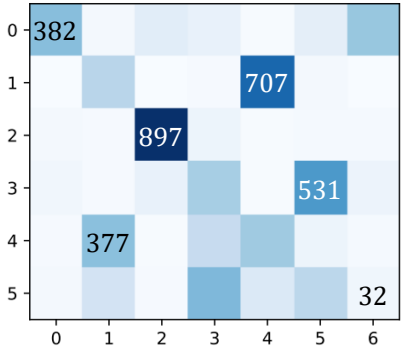
Maximum Weight Matching M_1
$W(M_1) = 3298$ $GED(M_1) = 5$



2nd-best Matching M_2
$W(M_2) = 3265$ $\mathbf{GED(M_2) = 2}$



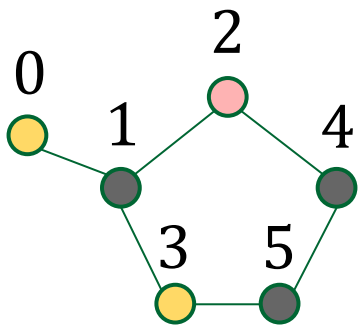
3rd-best Matching (M_3)
$W(M_3) = 2932$ $GED(M_3) = 10$



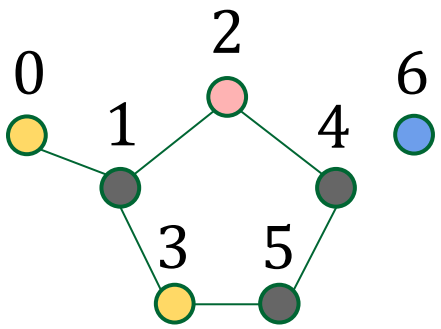
4th-best Matching (M_4)
$W(M_4) = 2926$ $GED(M_4) = 10$

V_1	V_2
0	6
1	4
2	2
3	5
4	1
5	3

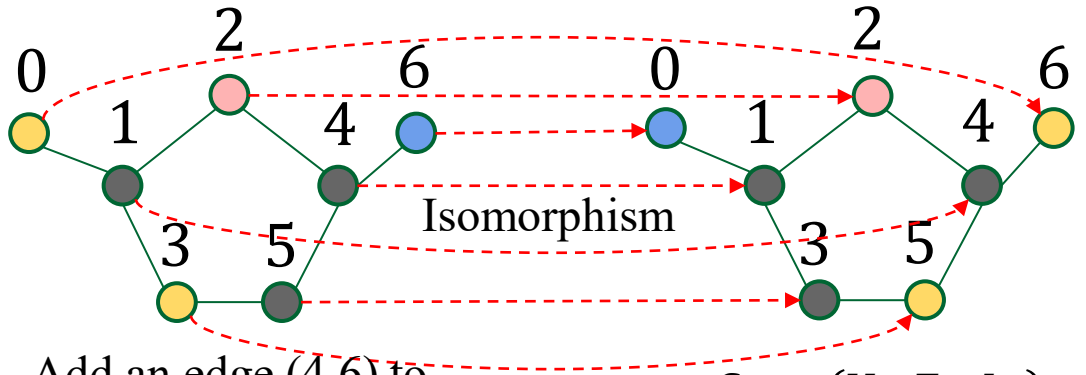
M_2



$G_1 = (V_1, E_1, L_1)$



Add an extra node (6)
to match node 0 of V_2.



Add an edge (4,6) to
match edge (1,0) of E_2.

$G_2 = (V_2, E_2, L_2)$

k-best Matching Algorithm

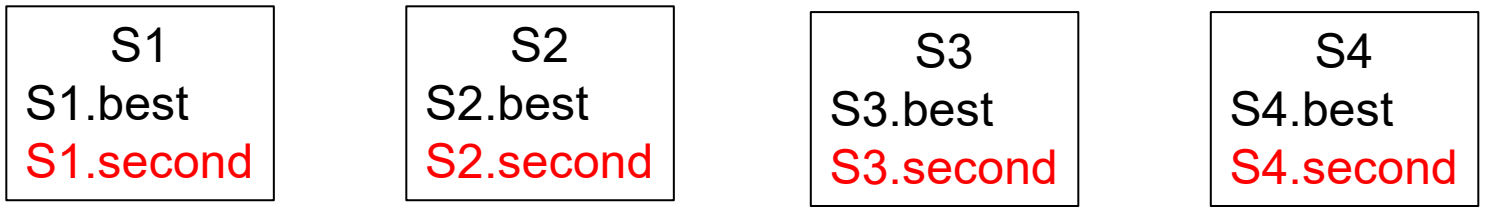
- Solution Space:
  - The set of possible matchings.

- Splitting
  - Whole Solution Space S: the set of all node matchings
  - We can easily split S into two parts by branching on a node pair.
  - For a given node pair (u, v), $u \in V1$ and $v \in V2$:
    - Subspace S1: {M | M ∈ S and M contains (u, v)}
    - Subspace S2: {M | M ∈ S and M does not contain (u, v)}

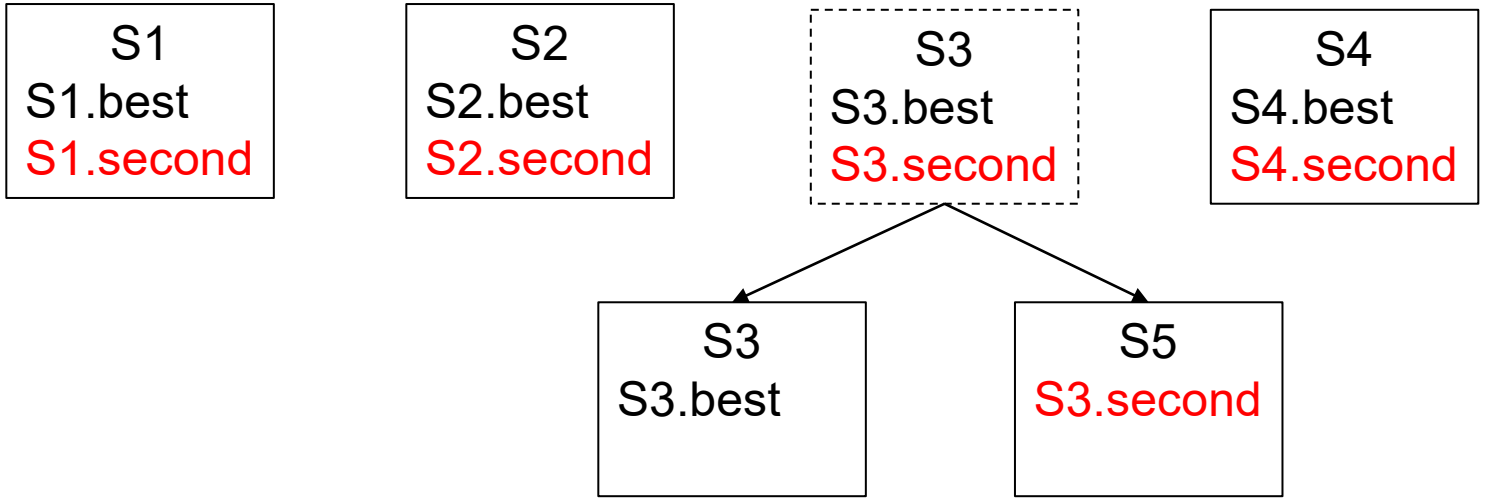
k-best Matching Algorithm

- Suppose we have a solution space partition:
  - $S = S_1 \cup S_2 \cup \cdots \cup S_k$
  - The best matching of S_i is exactly the i-th best matching of the whole solution space S.
  - The (k+1)-th best matching is the best one among the second-best matchings of each subspace.

- Maintain s sequence of subspaces.



- The best one of {S1.second, S2.second, S3.second, S4.second} becomes S5.best.



Solution Space Splitting

- Find K-best matchings
  - Iteratively find second-best matchings in solution subspaces
  - One iteration: second-best matching $O(n^3)$
  - K iterations: $O(Kn^3)$

Experimental Studies

Tested our method on three real graph data sets (AIDS, Linux and IMDB) and synthetic power-law graphs.

- AIDS:
  - Chemical compound.
  - Node label: C, N, O, Cu, …
- Linux:
  - Program dependence graphs of Linux kernel functions
- IMDB
  - Ego-networks of co-star relations.

Statistics of Graph Data Sets

	#graphs	$\overline{\lvert V \rvert}$	$\overline{\lvert E \rvert}$	$\lvert V \rvert_{max}$	$\lvert E \rvert_{max}$
AIDS	700	8.9	8.8	10	14
Linux	1000	7.6	6.9	10	13
IMDB	1500	13	65.9	89	1467
IMDB-small	148	8.1	25.2	10	45
IMDB-large	152	19.1	117.1	54	858

- The synthetic graphs contain 25 ~ 400 nodes.

Groundtruth data Generation

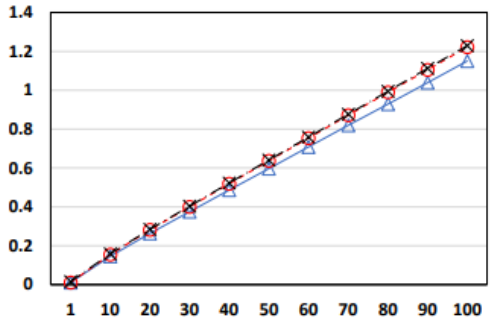
- Small Graphs (less than 10 nodes)
  - Exhaustive search for an arbitrary graph pair.
  - Input: (G1, G2)
  - Output: GED(G1, G2) and corresponding edit path
- Large Graphs (more than 10 nodes)
  - Generate synthetic pairs for each large graph.
  - Input: (G1, t)
  - Output: G2 (randomly conduct t edit operations on G1)

Overall Performance

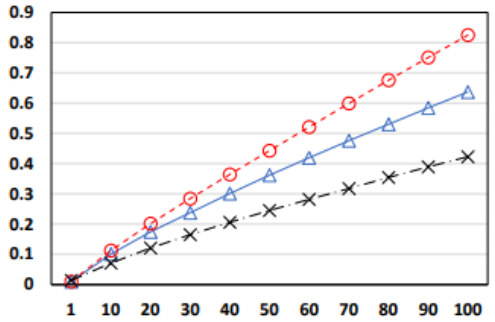
AIDS	GED		Ranking				Graph Edit Path			Feasible	Time
	MAE	Accuracy	ρ	τ	p@10	p@20	Precision	Recall	F1	Rate	(s/100p)
SimGNN	0.914	33.8%	0.832	0.693	62.4%	72.0%	-	-	-	67.6%	0.283
TaGSim	0.841	36.6%	0.850	0.715	64.6%	74.6%	-	-	-	66.2%	0.123
GPN	0.902	35.3%	0.822	0.684	58.6%	70.4%	-	-	-	66.8%	0.326
GEDGNN-value	**0.773**	**39.7%**	**0.876**	**0.751**	**71.6%**	**77.9%**	-	-	-	62.2%	0.408
Greedy	9.227	1.3%	0.464	0.362	50.2%	57.2%	32.5%	59.6%	41.2%	100.0%	110.000
Noah	3.078	6.3%	0.730	0.610	70.9%	75.1%	49.9%	62.0%	54.7%	100.0%	168.390
GEDGNN-matching	**1.427**	**44.3%**	**0.806**	**0.704**	**85.5%**	**85.0%**	**68.5%**	**74.3%**	**71.0%**	100.0%	122.900

IMDB-large	GED		Ranking				Graph Edit Path			Feasible	Time
	MAE	Accuracy	ρ	τ	p@10	p@20	Precision	Recall	F1	Rate	(s/100p)
SimGNN	1.470	23.2%	0.480	0.364	57.8%	63.0%	-	-	-	54.1%	0.258
TaGSim	3.752	7.4%	0.115	0.090	43.3%	50.3%	-	-	-	20.3%	0.113
GPN	1.591	**27.7%**	0.551	0.456	54.6%	59.0%	-	-	-	57.5%	0.331
GEDGNN-value	**1.398**	26.9%	**0.629**	**0.500**	**69.6%**	**71.5%**	-	-	-	68.5%	0.414
Greedy	24.961	40.4%	0.671	0.602	73.8%	72.4%	42.3%	75.8%	45.3%	100.0%	4.000
Noah	18.596	49.5%	0.547	0.525	60.3%	65.8%	45.5%	80.4%	48.9%	100.0%	10132.777
GEDGNN-matching	**6.078**	**67.9%**	**0.795**	**0.751**	**86.6%**	**84.8%**	**74.6%**	**91.4%**	**77.3%**	99.9%	264.800

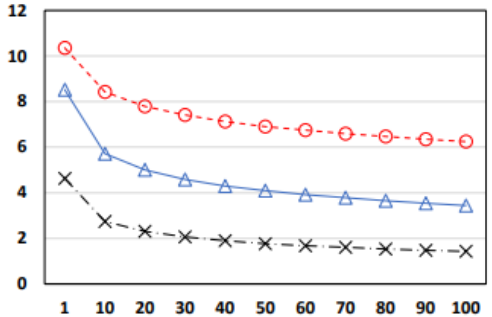
Evaluation of k-best Matching Framework



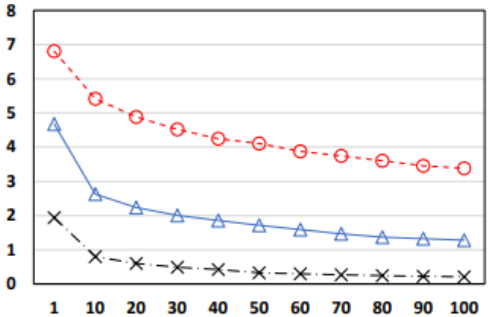
(a) AIDS - Time (s/graph pair)



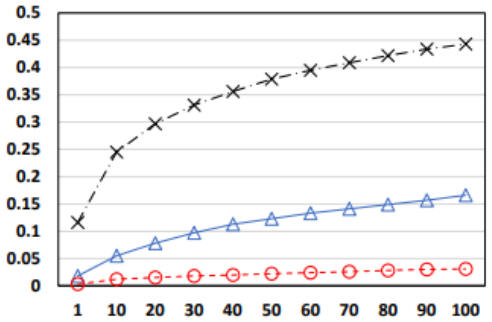
(b) Linux - Time (s/graph pair)



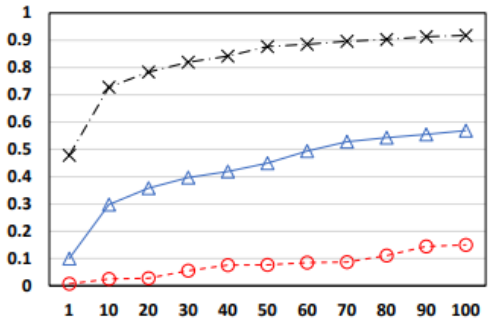
(c) AIDS - GED MAE



(d) Linux - GED MAE



(e) AIDS - Accuracy



(f) Linux - Accuracy

Greedy GEDGNN untrained GEDGNN matching

Summary

- Proposed a novel deep learning framework that solves the GED problem in a two-step manner:
  - 1) The graph neural network GEDGNN is in charge of predicting the GED value and a matching matrix;
  - 2) A post-processing algorithm based on k-best matching is used to extract multiple node matchings from the matching matrix generated by GEDGNN. The best of them will finally lead to a high-quality edit path.

- The post-processing algorithm is a key innovation that makes up the gap between what graph neural networks can produce (i.e., the matching matrix) and an actual solution to the GED problem.

Thank you!