Applications of Graph Neural Networks

Hanchen Wang

31/07/2023

Recap: Graph Neural Networks

- **(1) Graph Convolutional Networks (GCN)**

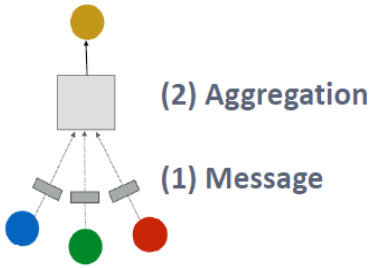
$$\mathbf{h}_v^{(l)} = \sigma\left(\mathbf{W}^{(l)} \sum_{u \in N(v)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}\right)$$

- **How to write this as Message + Aggregation?**

Message

$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}\right)$$

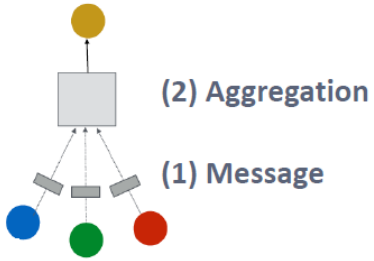
Aggregation



Recap: Graph Neural Networks

- **(1) Graph Convolutional Networks (GCN)**

$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}\right)$$



- **Message:**
  - Each Neighbor: $\mathbf{m}_u^{(l)} = \frac{1}{|N(v)|}\mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}$

    Normalized by node degree
    (In the GCN paper they use a slightly different normalization)
- **Aggregation:**
  - **Sum** over messages from neighbors, then apply activation
  - $\mathbf{h}_v^{(l)} = \sigma\left(\text{Sum}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}\right)\right)$

Recap: Graph Neural Networks

- **(2) GraphSAGE**

$$\mathbf{h}_v^{(l)} = \sigma\left(\mathbf{W}^{(l)} \cdot \mathrm{CONCAT}\left(\mathbf{h}_v^{(l-1)}, \mathrm{AGG}\left(\left\{\mathbf{h}_u^{(l-1)}, \forall u \in N(v)\right\}\right)\right)\right)$$

- **How to write this as Message + Aggregation?**
  - **Message** is computed within the $\mathrm{AGG}(\cdot)$
  - **Two-stage aggregation**
    - **Stage 1:** Aggregate from node neighbors

$$\mathbf{h}_{N(v)}^{(l)} \leftarrow \mathrm{AGG}\left(\left\{\mathbf{h}_u^{(l-1)}, \forall u \in N(v)\right\}\right)$$

    - **Stage 2:** Further aggregate over the node itself

$$\mathbf{h}_v^{(l)} \leftarrow \sigma\left(\mathbf{W}^{(l)} \cdot \mathrm{CONCAT}(\mathbf{h}_v^{(l-1)}, \mathbf{h}_{N(v)}^{(l)})\right)$$

Recap: Graph Neural Networks

- **Mean:** Take a weighted average of neighbors

$$\mathrm{AGG} = \sum_{u \in N(v)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}$$

Aggregation **Message computation**

- **Pool:** Transform neighbor vectors and apply symmetric vector function $\mathrm{Mean}(\cdot)$ or $\mathrm{Max}(\cdot)$

$$\mathrm{AGG} = \mathrm{Mean}(\{\mathrm{MLP}(\mathbf{h}_u^{(l-1)}), \forall u \in N(v)\})$$

Aggregation **Message computation**

- **LSTM:** Apply LSTM to reshuffled of neighbors

$$\mathrm{AGG} = \mathrm{LSTM}([\mathbf{h}_u^{(l-1)}, \forall u \in \pi(N(v))])$$

Aggregation

Recap: Graph Neural Networks

- **(3) Graph Attention Networks**

$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \boxed{\alpha_{vu}} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}\right)$$

Attention weights

- **In GCN / GraphSAGE**
  - $\alpha_{vu} = \frac{1}{|N(v)|}$ is the **weighting factor (importance)** of node u's message to node v
  - $\Longrightarrow$ α_{vu} is defined **explicitly** based on the structural properties of the graph (node degree)
  - $\Longrightarrow$ All neighbors $u \in N(v)$ are equally important to node v

Recap: Graph Neural Networks

- **(3) Graph Attention Networks**

$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \boxed{\alpha_{vu}} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}\right)$$

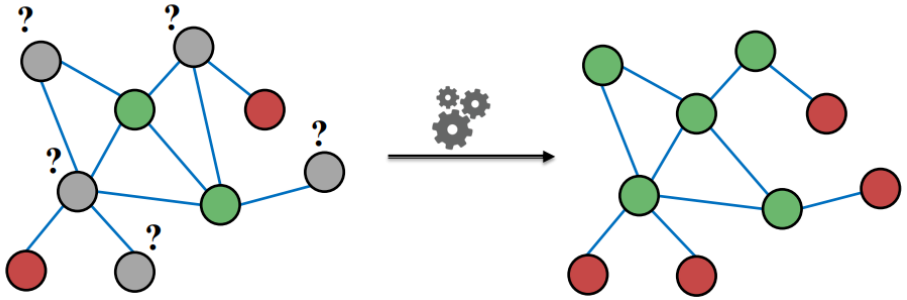
Attention weights

Not all node's neighbors are equally important

- **Attention** is inspired by cognitive attention.
- The **attention** $\boldsymbol{\alpha_{vu}}$ focuses on the important parts of the input data and fades out the rest.
  - **Idea:** the NN should devote more computing power on that small but important part of the data.
  - Which part of the data is more important depends on the context and is learned through training.

Application of Graph Neural Networks

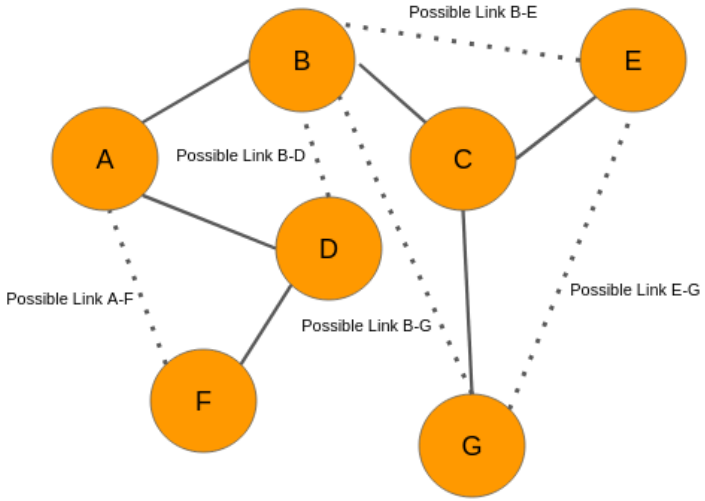
- **Node Classification**

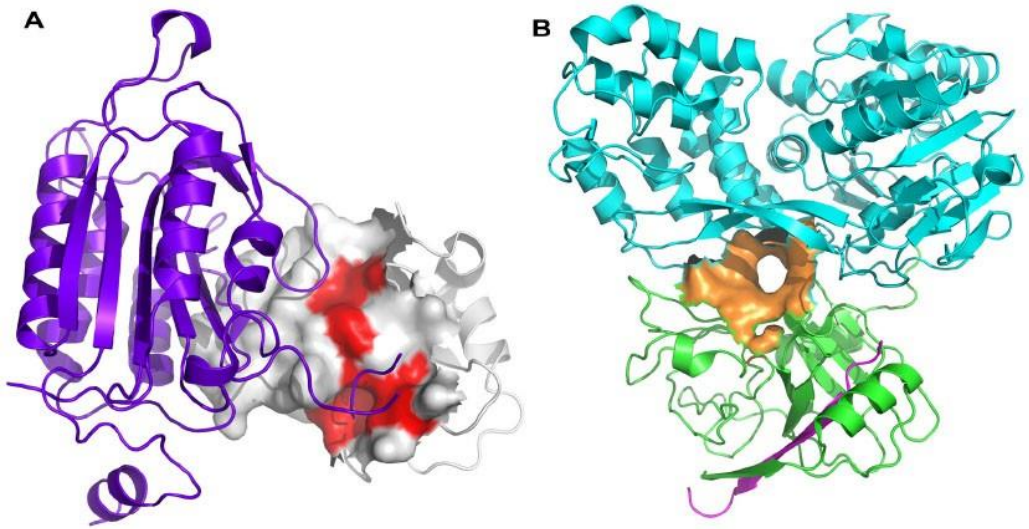


- ❑ Some labels of nodes are given
- ❑ Predict the labels of Unlabelled nodes.

Application of Graph Neural Networks

- **Link Prediction**





- With existing edges (links) between nodes, predict the existence of possible links.
- Link prediction can be used to explore potential interaction between proteins.

Applications of GNNs

- Structured Entity Analysis (Chemical, Biomedical)
- Subgraph Isomorphism (Database)
- Fraud Detection (e-Commerce)

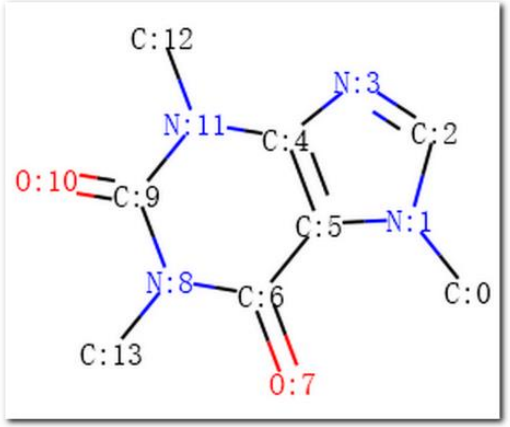
Structured Entity Analysis

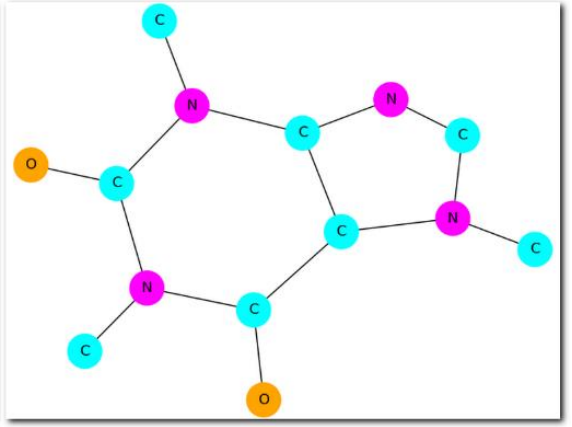
UNSW SYDNEY

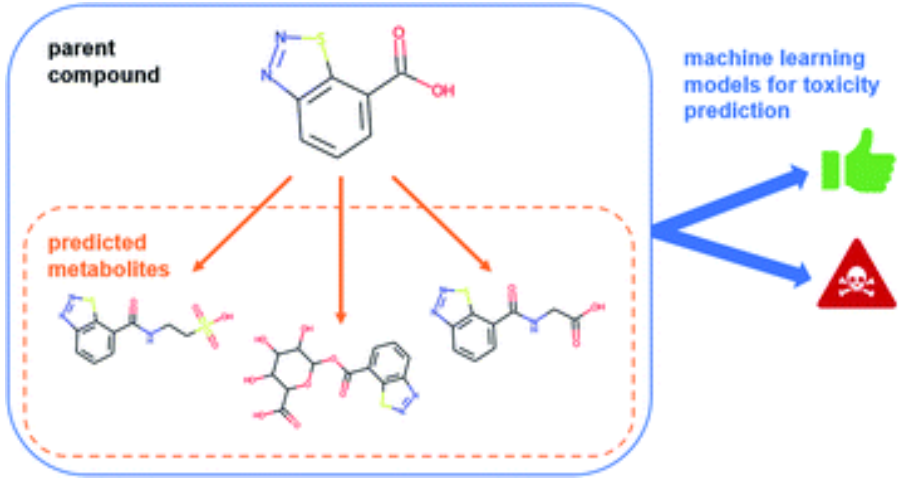
Structured Entity Analysis

- Many real-life **structured entities** can be modelled as **graphs**, such as chemical molecules and proteins.

- Graph neural networks can be used to analyse these structured entities.

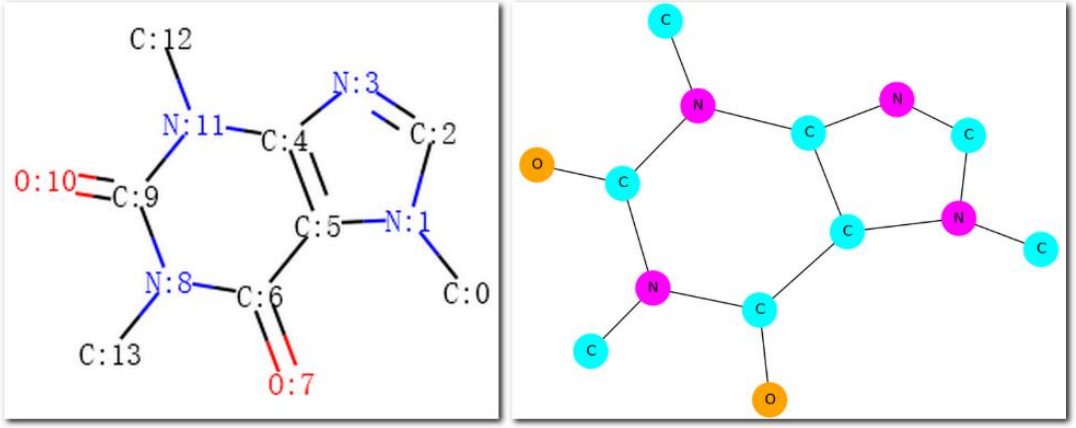


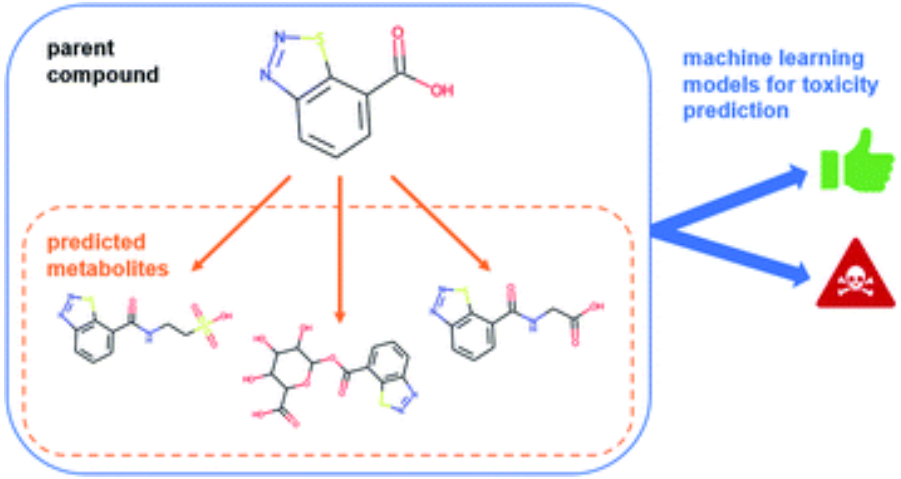




Structured Entity Analysis

- Toxicity of chemical molecules.
- Property prediction for molecules.
- **Interactions between molecules.**





Interaction between molecules

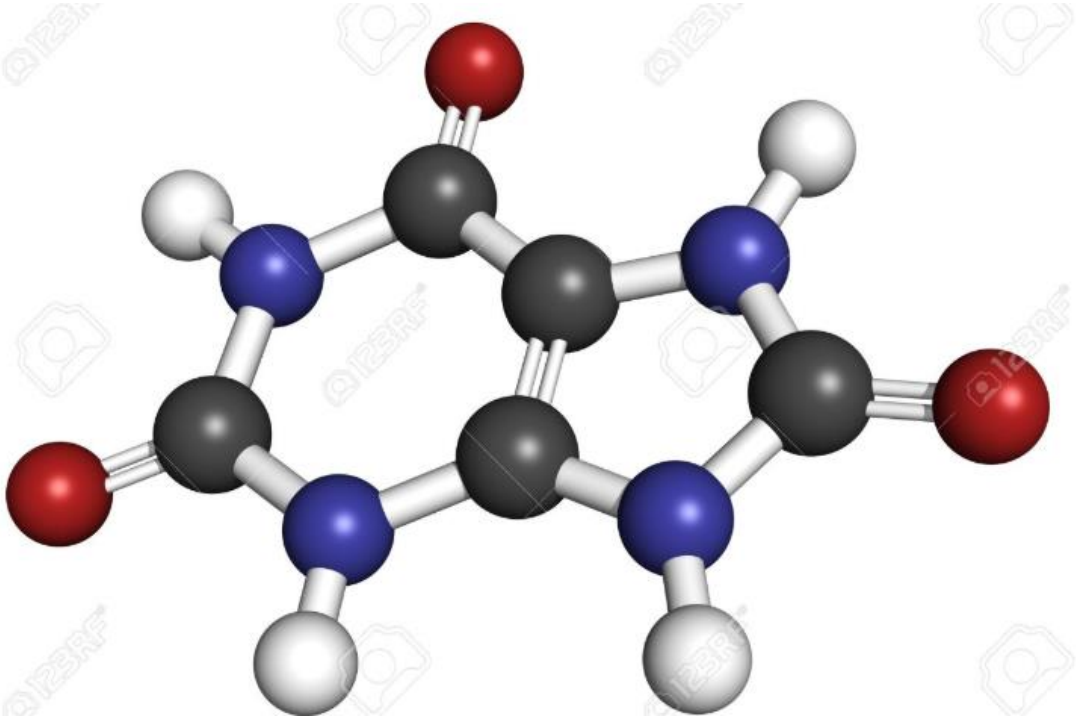
- Prediction of reaction between two molecules
- Prediction of the side effect of drug pairs.

Immediate solutions

- The immediate way to investigate the interactions between two entities (molecules or drugs) is to conduct experiments in laboratories and clinics.
- However, it is time consuming and labour intensive. It is impractical to test all entity pairs.
- Therefore, computational approaches are proposed to predict structured entity interaction effectively and efficiently.

Naïve computational solution

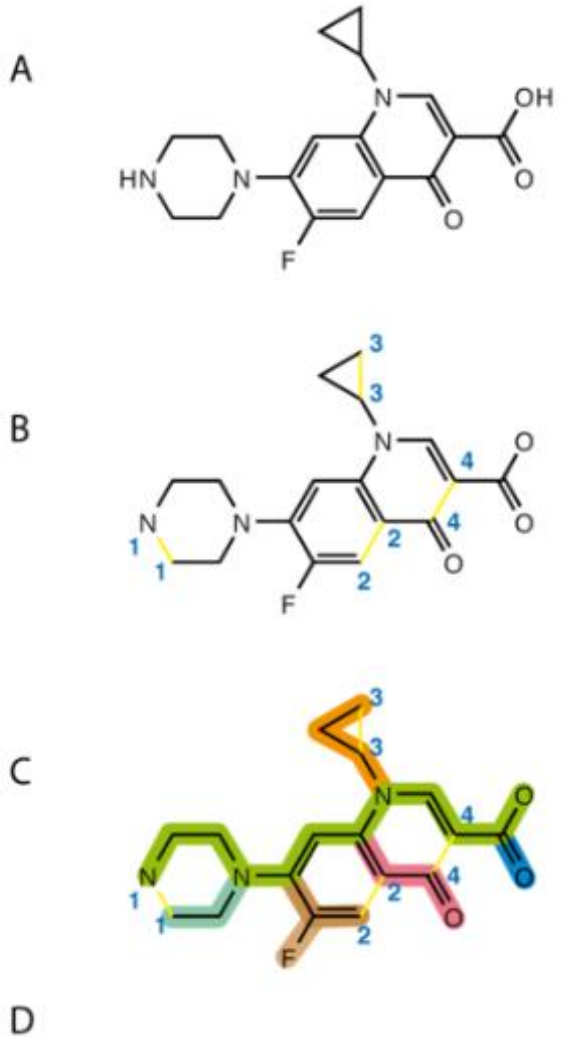
- Model the molecules as graphs
- Directly apply the graph neural networks for the predictions.
- However, such solution ignores the **interaction relation** between molecules.



Other computational methods

Simplified molecular-input line-entry system (SMILES)

- Using strings to represent molecules.
- Natural Language Processing (NLP) models are applied to capture the molecule information and produce the representations for molecules
- Downstream applications are based on the representations.
- However, these models cannot capture the structural information of the molecules.



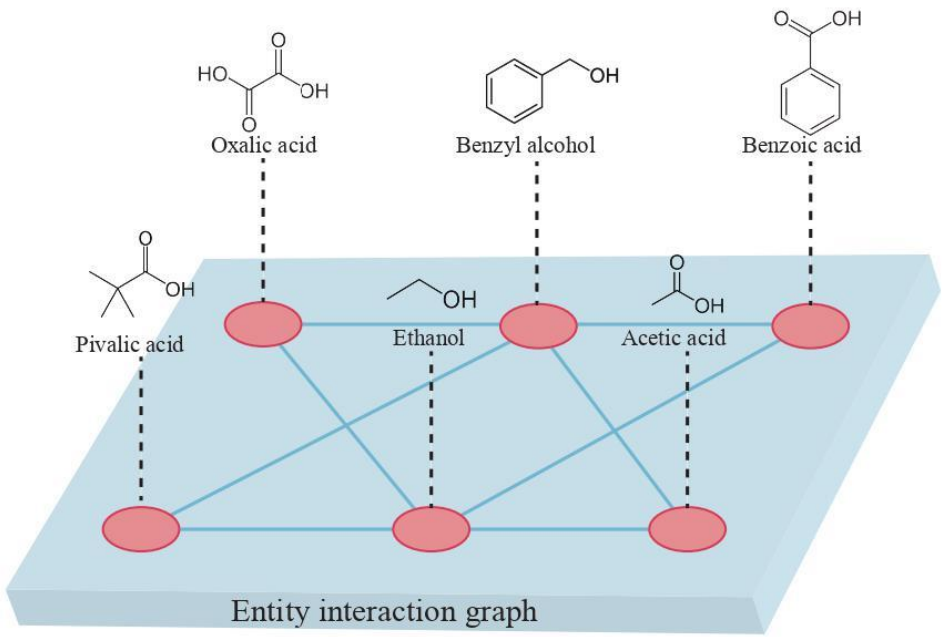
Goals

- Capture the structural information of the molecule graphs.
- Preserve the interaction relationship between the structured entities (molecules).
- Identify the important substructures within the molecules that are key to the prediction of interactions.

Graph of Graphs

- *Local graph*: the molecule graph representing the chemical structure.
- (nodes: atoms, edges: chemical bonds between the atoms.)

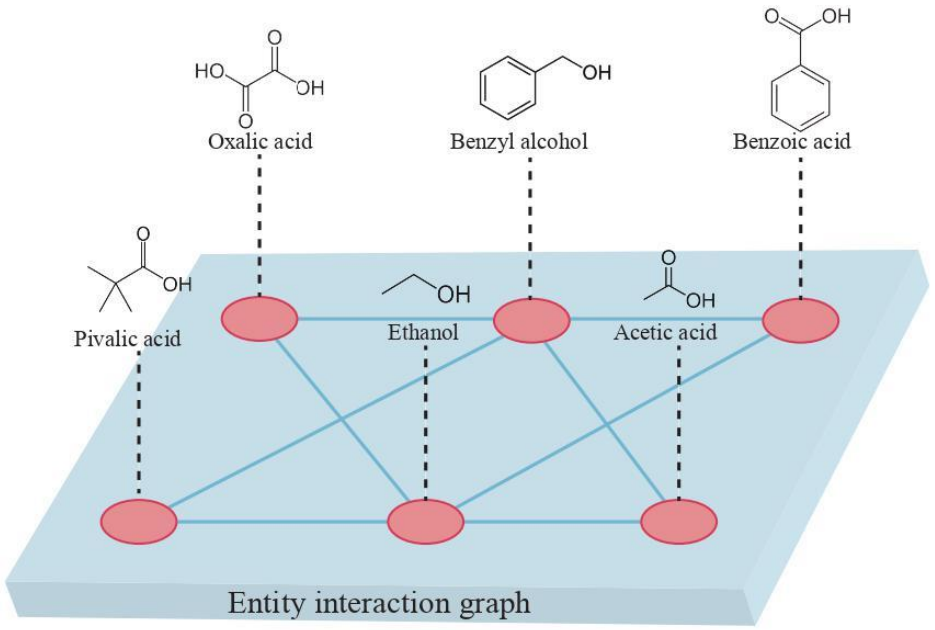
- *Global graph*: the graph of interactions between the chemicals.
- (nodes: chemical molecules, edges: interaction relations between the molecules.)



Target of the model

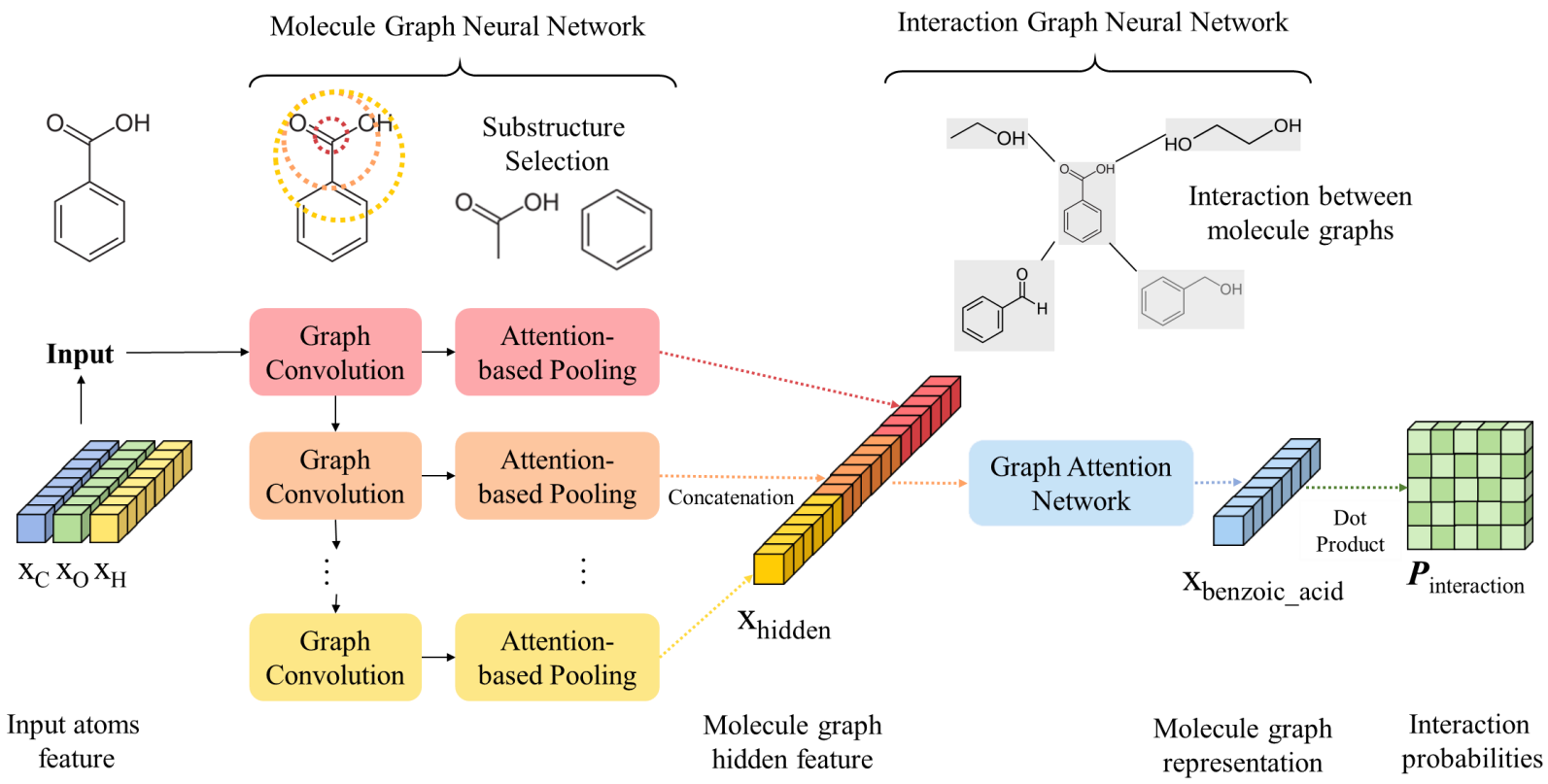
- With the input graph, the target of the model is to conduct the **link prediction** on the *global graph*.

- The graph neural network applied on the *local graph* should be able to capture the important substructure (functional groups) in the molecules.



Graph of Graphs Neural Network

- Apply graph neural networks on the graph of graphs.
- Utilize more information within the graph structures.
- Can be used for entity interaction prediction and molecule classification.

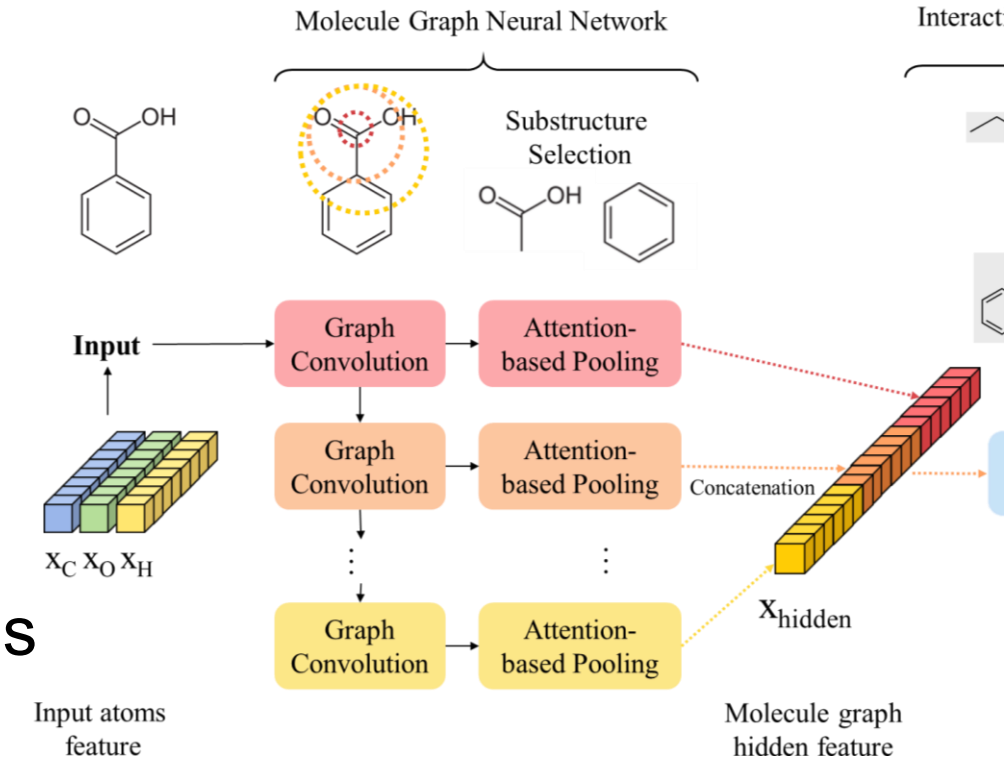


Our recent work
- “GoGNN: Graph of Graphs Neural Network for Predicting Structured Entity Interactions” IJCAI 2020

Graph of Graphs Neural Network

Multi-Resolution Graph Neural Network

- Use the concatenation of the output after different layer of GNNs to keep multi-scale (multi-hop) substructure information.
- Each layer of graph neural network captures one-hop neighbour relationship.



Graph of Graphs Neural Network

Local graph neural network:

The neural network used to learn representations for chemical molecules. We use *l*-layer graph convolutional network (GCN) as the local graph neural network:

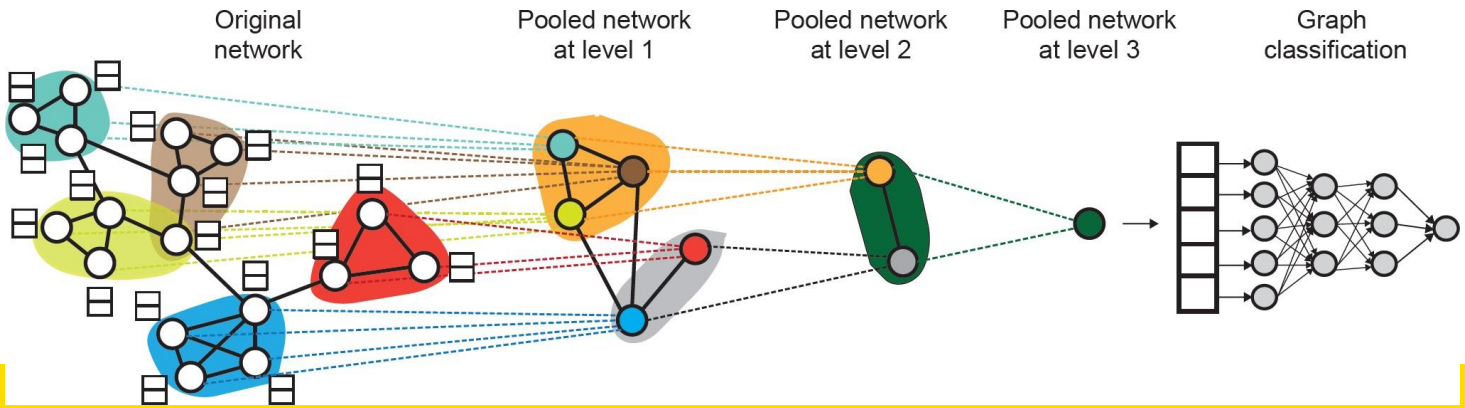
$$M_{(l+1)} = GCN_l(A, M_l)$$

$$GCN_l(A, M_l) = \sigma\left(D^{-\frac{1}{2}}\tilde{A}D^{-\frac{1}{2}}M_l W_l\right)$$

Graph of Graphs Neural Network

Graph pooling

Graph pooling is a computational strategy to reduce the number of graph nodes; in this way, one has a unified graph-level rather than node-level representation for graph-structured data while the size and topology of an individual graph are changing.



Graph of Graphs Neural Network

Graph pooling examples

- **(1) Global mean pooling**

$$\boldsymbol{y}_0 = \mathrm{Mean}(\{\mathbf{h}_\mathrm{v} \in \mathbb{R}^\mathrm{d}, \forall v \in G\})$$

- **(2) Global max pooling**

$$\boldsymbol{y}_0 = \mathrm{Max}(\{\mathbf{h}_\mathrm{v} \in \mathbb{R}^\mathrm{d}, \forall v \in G\})$$

- **(3) Global sum pooling**

$$\boldsymbol{y}_0 = \mathrm{Sum}(\{\mathbf{h}_\mathrm{v} \in \mathbb{R}^\mathrm{d}, \forall v \in G\})$$

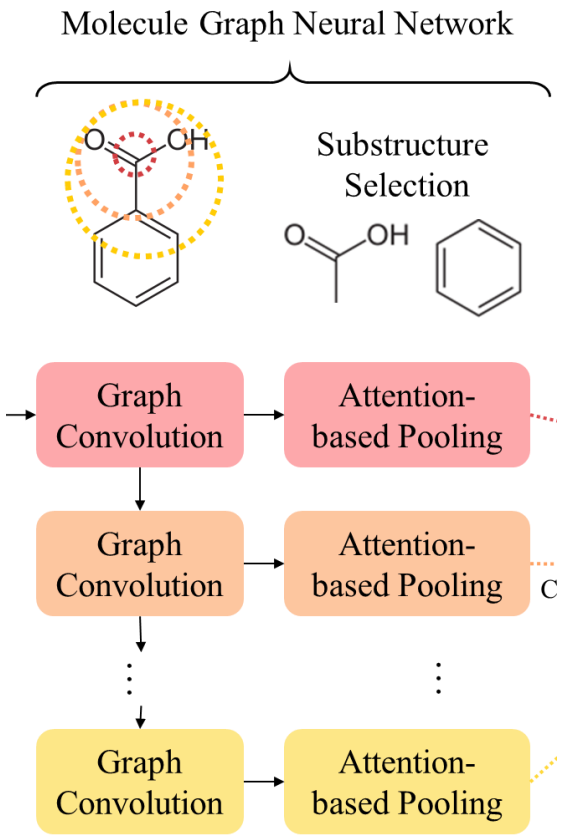
- **Issue:** Global pooling over a (large) graph will lose information

Graph of Graphs Neural Network

Graph attention pooling

- The attention-based pooling method to select **the most representative substructure** to represent the molecule graph:

$$s_l = \sigma\left(\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}M_l W_{att}^{l}\right)$$
$$idx = top(s, \lceil \gamma n \rceil),$$
$$s_{mask} = s_{idx}, M_{sel} = M \odot s_{mask}$$



Graph of Graphs Neural Network

Graph attention pooling

$$s_l = \sigma\left(\widetilde{D}^{-\frac{1}{2}}\tilde{A}\widetilde{D}^{-\frac{1}{2}}M_l W_{att}^{l}\right)$$
$$idx = top(s, \lceil \gamma n \rceil),$$
$$s_{mask} = s_{idx}, M_{sel} = M \odot s_{mask}$$

- M_{sel} is the representation matrix for the selected atoms in the molecule graph
- After selection, the combination of mean pooling and sum pooling is used to produce the representation for the molecule graph, which is also the input for interaction graph neural network.

attention pooling

$s_l = \sigma\left(l\right.$
$idx =$
$s_{mask} = s_{id}$

is the representatio
molecule graph

r selection, the com
ing is used to produ
ecule graph, which i
h neural network.

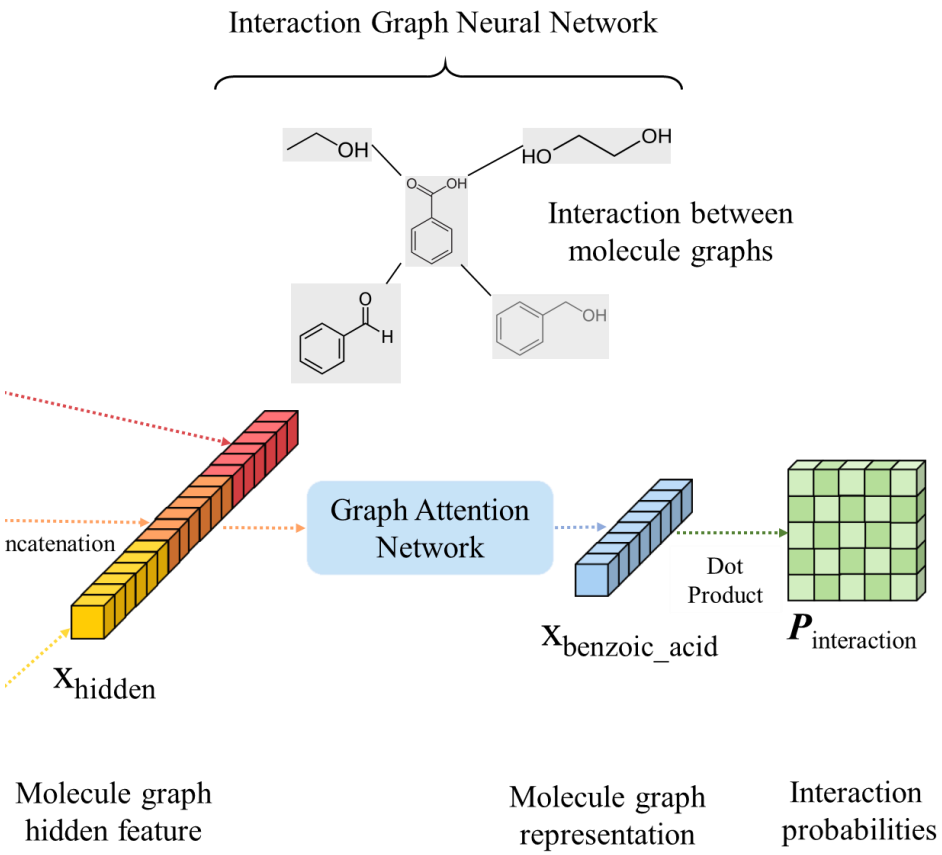
Graph of Graphs Neural Network

Graph attention network:

- With the learned molecule graph representations, the embedding is updated based on the interaction network. The multi-head attention mechanism is utilized:

$$x_{G_i}^{l+1} = \Big\|_{\kappa=1}^{K} \sigma\left(\sum_{j \in \eta_{G_i}} \alpha_{ij}^{\kappa} W_{\kappa}^{l} x_{G_j}^{l}\right)$$

$$\alpha_{ij} = \frac{exp\left(\text{LeakeyRelu}\left(a\left[Wx_{G_i}\right] || \left[Wx_{G_j}\right]\right)\right)}{\Sigma_{n \in \eta_{G_i}} exp\left(\text{LeakeyRelu}\left(a\left[Wx_{G_i}\right] || \left[Wx_{G_n}\right]\right)\right)}$$



Graph of Graphs Neural Network

Training objectives

The predicted interaction probability:

$$p_{ij} = \sigma\left(x_{G_i}^T \cdot x_{G_j}\right)$$

Cross-entropy loss function:

$$\mathcal{L}_{CCI} = \sum_{(G_i, G_j) \in G_{CCI}} -log\left(p_{ij}\right) - E_{m \sim P_j} log\left(1 - p_{im}\right)$$

Graph of Graphs Neural Network

Datasets:

- CCI tasks: The CCI dataset assigns a score from 0 to 999 to describe the interaction probability where a higher score indicates higher interaction probability. According to threshold score, we get two datasets with chemical interaction probability score over 900 and 950: **CCI900** and **CCI950**.

- DDI tasks. For the drug-drug interaction prediction problem, **DDI** dataset and the side effect dataset **SE** are used. A vector representation (attribute) is assigned to each side effect type produced by pre-trained BERT model

Graph of Graphs Neural Network

Experiment results:

	CCI900		CCI950	
	AUC	AP	AUC	AP
DeepCCI	0.925	0.918	0.957	0.957
DeepDDI	0.891	0.886	0.916	0.915
MR-GNN	0.927	0.921	0.934	0.924
MLRDA	0.922	0.907	0.959	0.948
SEAL	0.894	0.886	0.941	0.937
GoGNN	**0.937**	**0.932**	**0.963**	**0.962**
GoGNN-M	0.914	0.909	0.938	0.931
GoGNN-I	0.921	0.898	0.929	0.912
GoGNN-noPool	0.931	0.930	0.958	0.954
GoGNN-noAttn	0.909	0.905	0.956	0.948

Table 1: Result of chemical-chemical interaction prediction task.

	DDI		SE	
	AUC	AP	AUC	AP
DeepCCI	0.862	0.856	0.819	0.806
DeepDDI	0.915	0.912	0.827	0.809
MR-GNN	0.932	0.922	0.769*	0.752*
MLRDA	0.931	0.926	0.847*	0.825*
Decagon	-	-	0.872	0.832
SEAL	0.925	0.921	N/A	N/A
GoGNN	**0.943**	**0.933**	**0.930**	**0.927**
GoGNN-M	0.905	0.902	0.862	0.817
GoGNN-I	0.922	0.917	0.860	0.834
GoGNN-noPool	0.900	0.891	0.912	0.909
GoGNN-noAttn	0.925	0.921	0.897	0.883

* indicates that the result is the output of the baselines after two weeks' training.

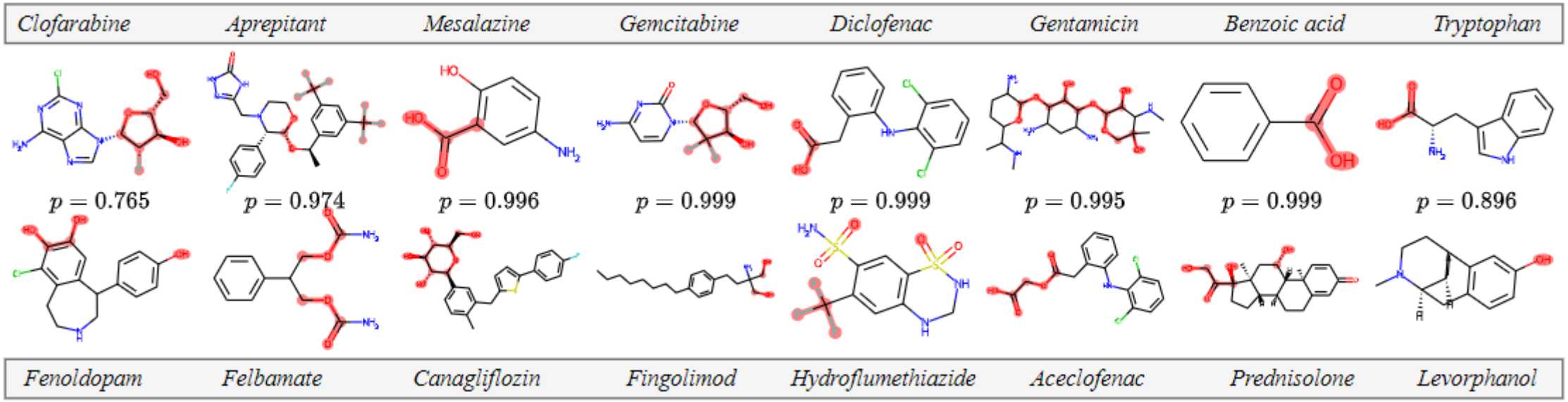
\- DDI dataset has no protein data which is required by Decagon

Table 2: Result of drug-drug interaction prediction task.

Graph of Graphs Neural Network

Case study:

Red: the substructures that are responsible of the interactions.



Our recent work
- “Denoising Variational Graph of Graphs Auto-Encoder for Predicting Structured Entity Interactions” -- TKDE 2023

Subgraph Isomorphism

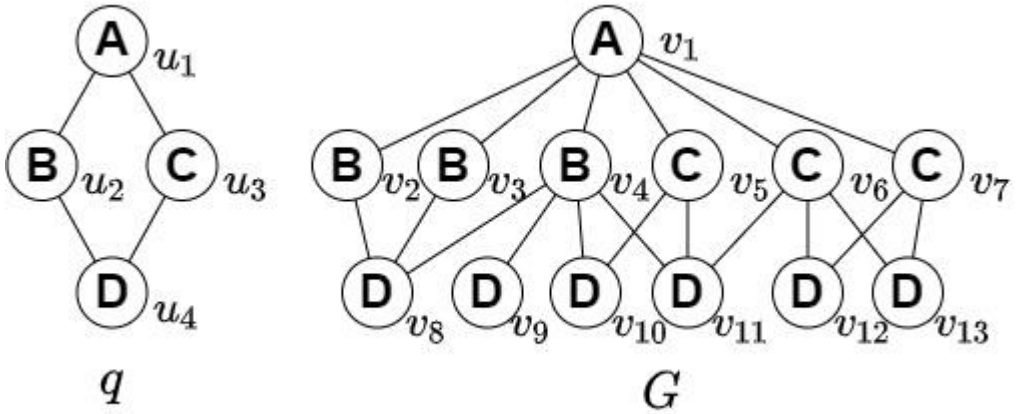
Subgraph Counting

Subgraph Isomorphism

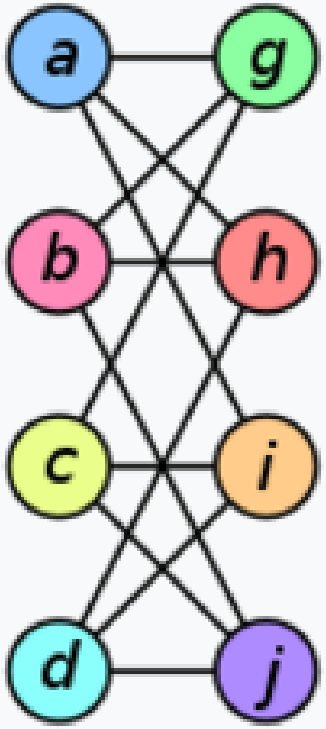
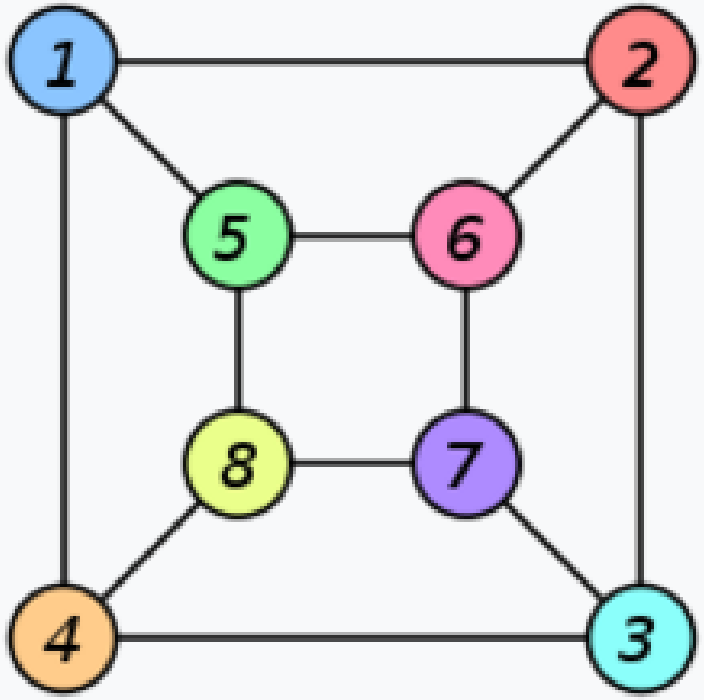
- Query graph $q = (V, E, f_l)$
- Data graph $G = (V', E', f_l)$

- Subgraph Isomorphism: injective function $f_{iso}: V \rightarrow V'$:
  - $\forall u \in V, f_l(u) = f_l(f_{iso}(u))$
  - $\forall e(u, u') \in E, e(f_{iso}(u), f_{iso}(u')) \in E'$

- Determining the existence of subgraph isomorphism is NP-complete.



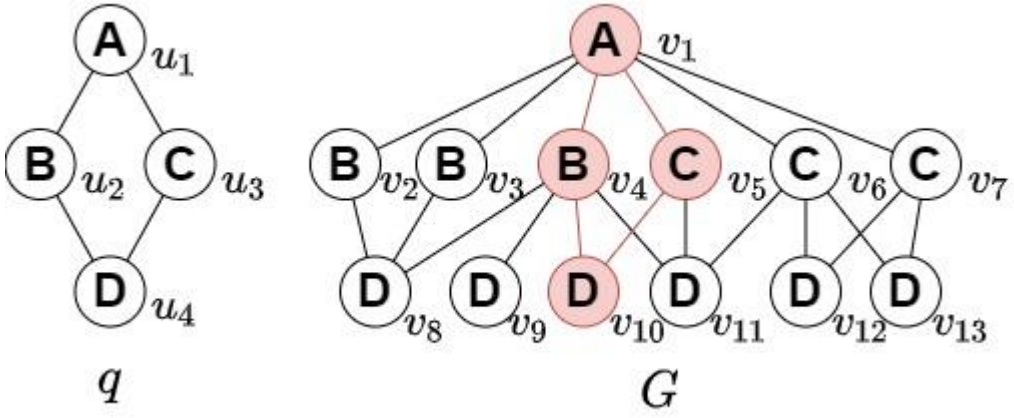
Graph isomorphism (a more complicated example)

Graph G	Graph H	An isomorphism between G and H
		$f(a) = 1$ $f(b) = 6$ $f(c) = 8$ $f(d) = 3$ $f(g) = 5$ $f(h) = 2$ $f(i) = 4$ $f(j) = 7$

https://en.wikipedia.org/wiki/Graph_isomorphism

Subgraph Counting

Subgraph Counting: Given a query graph q and a data graph G, the problem is to count the number of subgraphs in the data graph that match the query graph by subgraph <u>isomorphism</u>.

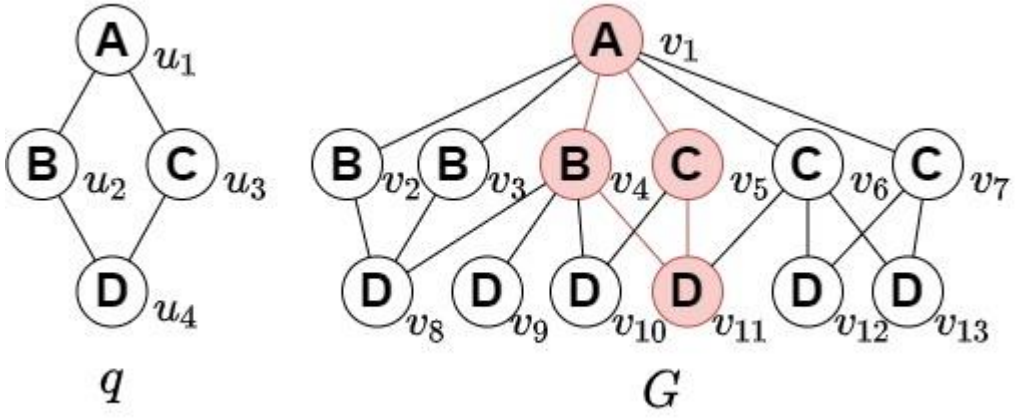


Subgraph isomorphisms

1. $(u_1, u_2, u_3, u_4) \rightarrow (v_1, v_4, v_5, v_{10})$

Subgraph Counting

Subgraph Counting: Given a query graph q and a data graph G, the problem is to count the number of subgraphs in the data graph that match the query graph by subgraph <u>isomorphism</u>.

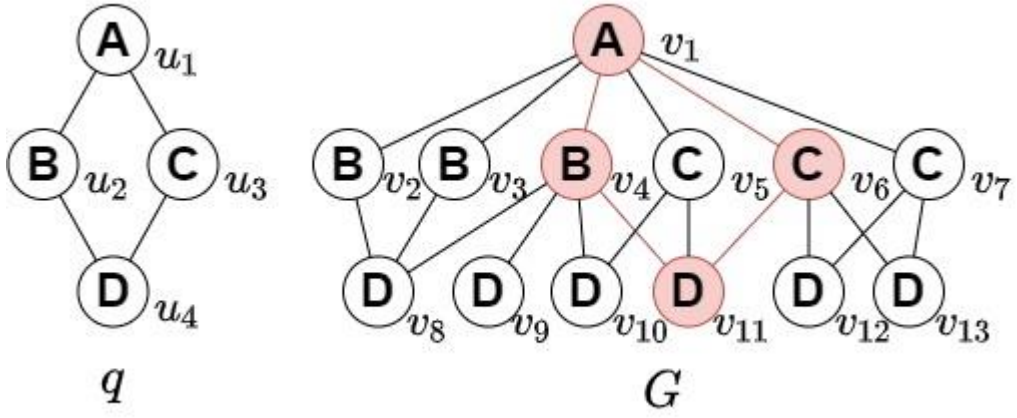


Subgraph isomorphisms

1. $(u_1, u_2, u_3, u_4) \rightarrow (v_1, v_4, v_5, v_{10})$
2. $(u_1, u_2, u_3, u_4) \rightarrow (v_1, v_4, v_5, v_{11})$

Subgraph Counting

Subgraph Counting: Given a query graph q and a data graph G, the problem is to count the number of subgraphs in the data graph that match the query graph by subgraph <u>isomorphism</u>.



Subgraph isomorphisms

1. $(u_1, u_2, u_3, u_4) \rightarrow (v_1, v_4, v_5, v_{10})$
2. $(u_1, u_2, u_3, u_4) \rightarrow (v_1, v_4, v_5, v_{11})$
3. $(u_1, u_2, u_3, u_4) \rightarrow (v_1, v_4, v_6, v_{11})$

Why Subgraph Counting?

Applications

Analysis on Social Networks:

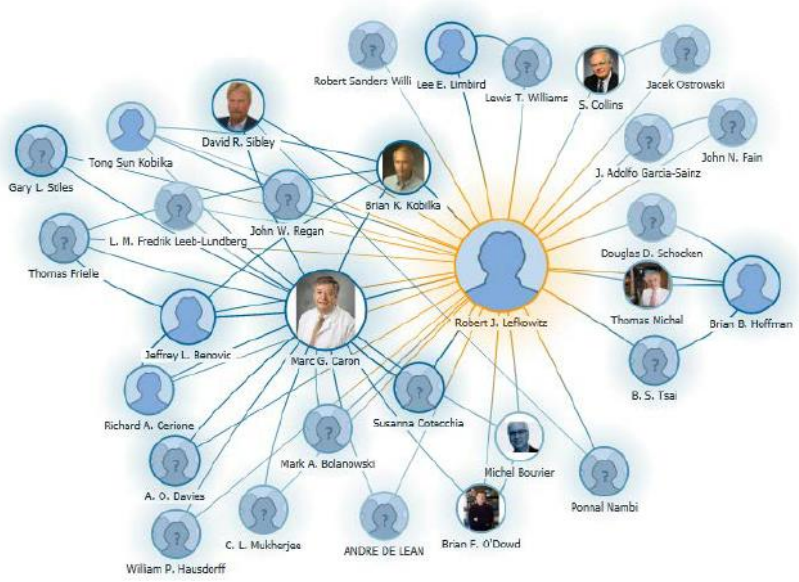
- Find co-authorships, community detection.

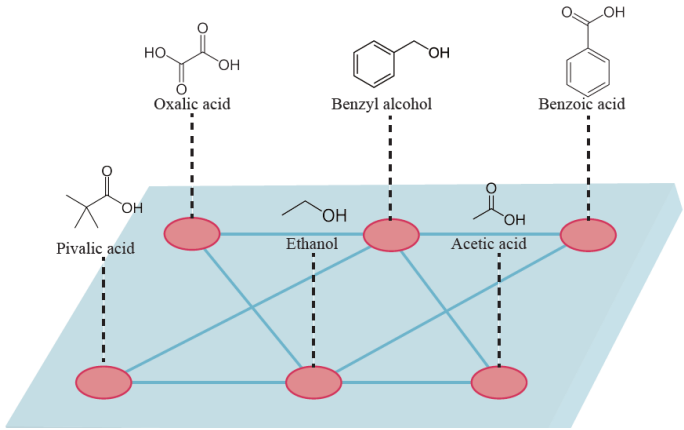
Analysis on Biological Networks:

- On brain, regulation, protein and molecule graphs.
- Summarize the structural patterns for the biological graphs.

Query Optimization for Subgraph Matching Queries:

- Cardinality estimation for multi-way join.





Existing Subgraph Counting Methods

Algorithmic Methods:

Enumeration-based methods:

- Computational complexity.

Sampling-based methods:

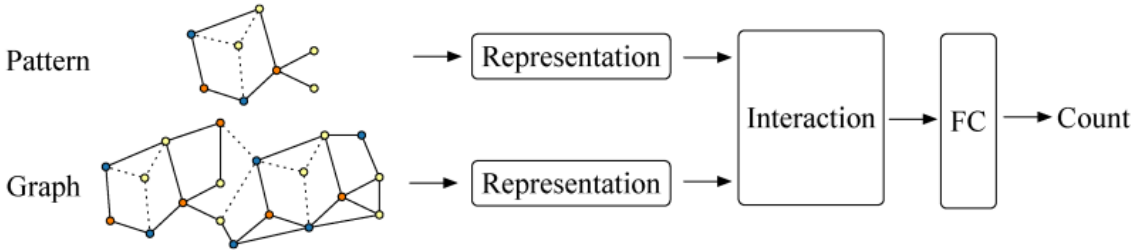
- Sampling failure.

Summary-based methods:

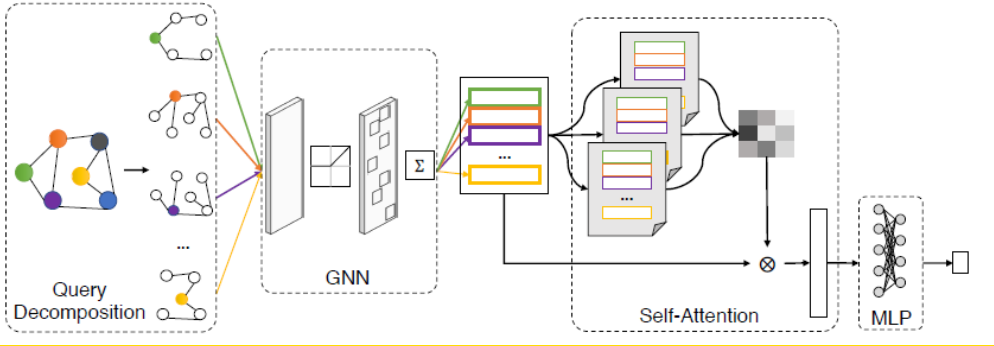
- Independent assumption.

Learning-based Methods:

Neural Subgraph Isomorphism Counting



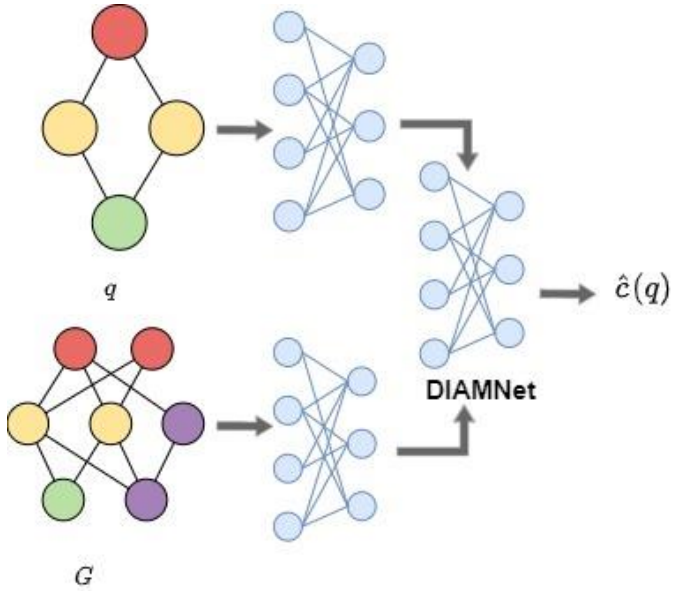
A Learned Sketch for Subgraph Counting



Existing Subgraph Counting Methods

Neural Subgraph Isomorphism Counting*:*

- The query graph and data graph are input into the graph neural networks for representation learning.
- The learned representations are input into the RNN-based network named DIAMNet to predict the subgraph counts.

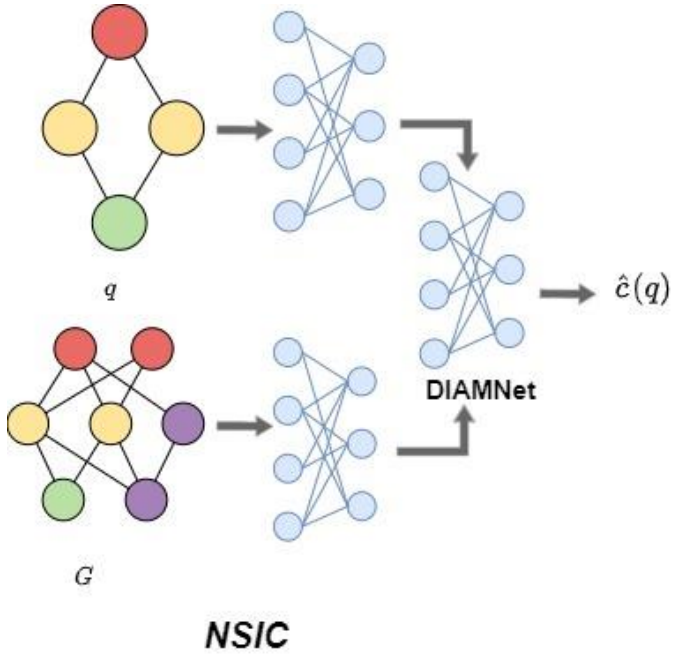


NSIC

Existing Subgraph Counting Methods

Neural Subgraph Isomorphism Counting*:*

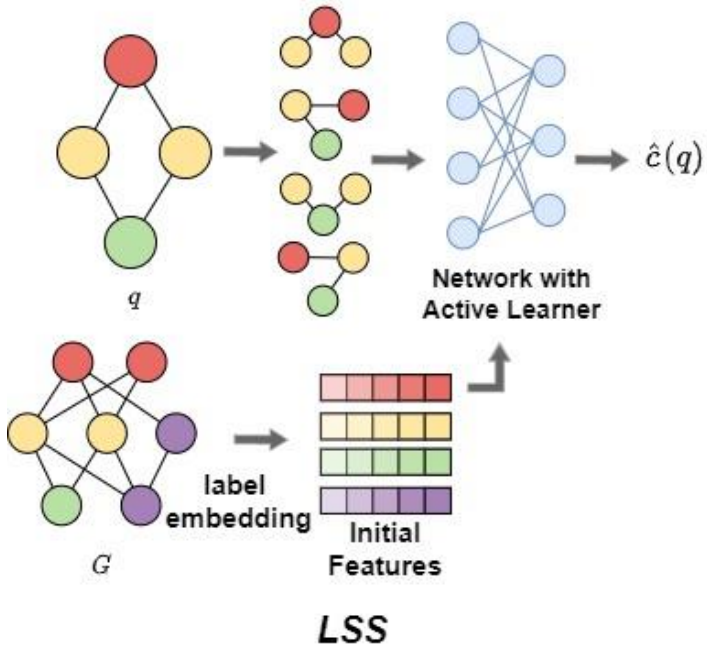
- The data graph is usually large-scale.
- The model will face the efficiency and scalability issue.
- Since the data graph contains more information, it is hard to distinguish the counting results of different query graphs when the data graph is large. The representation of data graph will dominate the computation in this case.



Existing Subgraph Counting Methods

A Learned Sketch for Subgraph Counting*:*

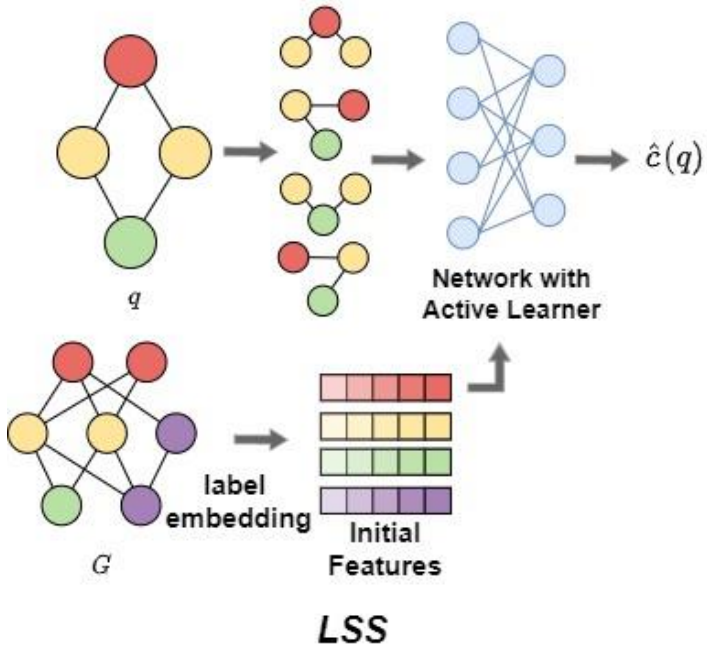
- The initial features of query nodes are computed based on the labels and structure of data graph by the graph embedding methods like DeepWalk.
- The query graph is decomposed into small substructures and fed into the graph neural network with active learner to predict the subgraph counts.



Existing Subgraph Counting Methods

A Learned Sketch for Subgraph Counting*:*

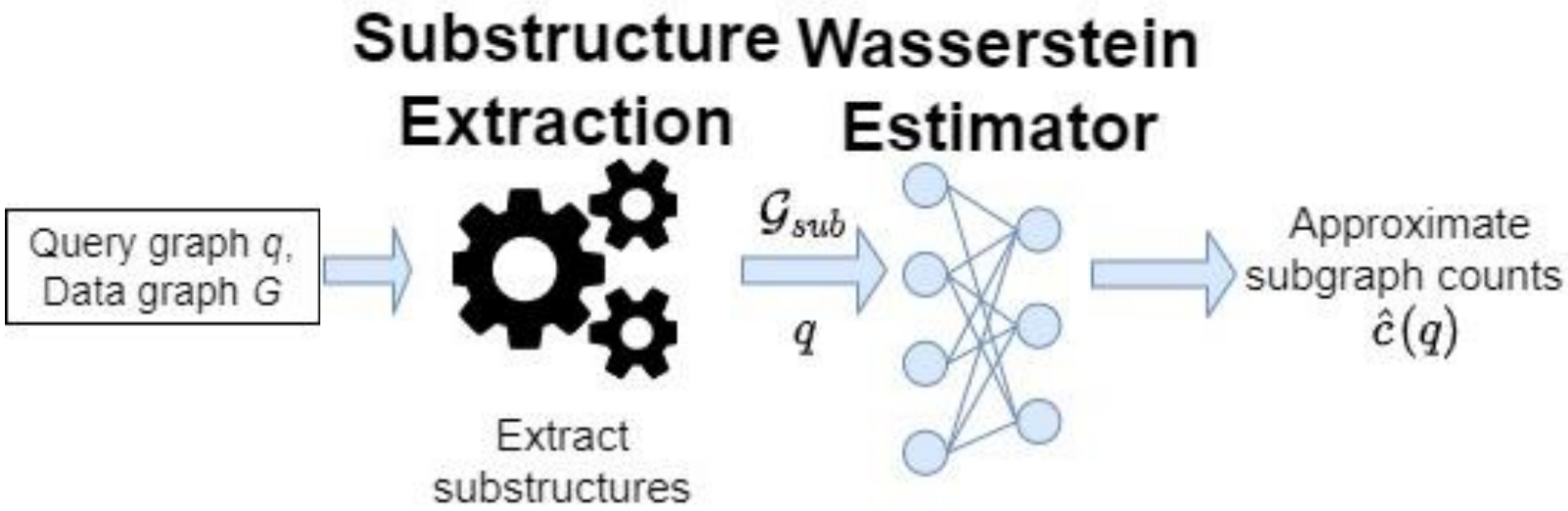
- Cannot fully utilize the data graph information. The topological information of data graph is somehow ignored.
- Consequently, the model has limited robustness, i.e., the result can be easily affected by minor modification on the query graph.



Goal

- Utilize both query and data graph information.
- Avoid efficiency and scalability issue.
- Improve the subgraph counting accuracy.

Neural Subgraph Counting method: NeurSC



Our recent work
- “Neural Subgraph Counting with Wasserstein Estimator” -- SIGMOD 2022

NeurSC

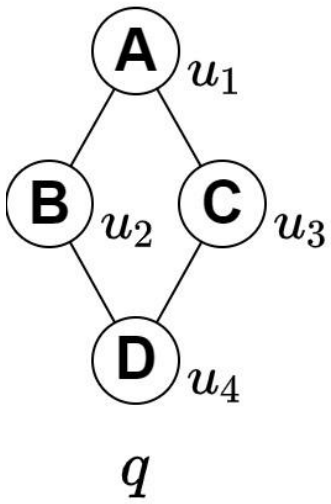
Substructure Extraction

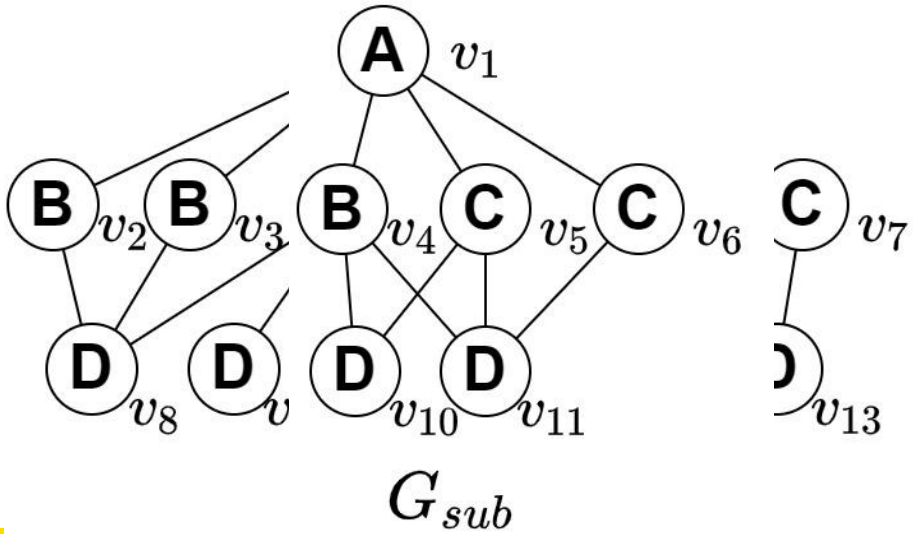
- **Complete Candidate Vertex Set** (CS):
  - $CS(u)$ for query vertex $u \in V$ is a set of data vertices $v \in V'$
  - If (u, v) exists in a match from q to G, then $v \in CS(u)$

- First, we determine the complete candidate vertex set for all query vertices using *local pruning* and *global refinement*.
- Based on neighboring and label information

- Candidate set of query q: $CS(\mathrm{q}) = \cup_{u \in V} CS(u)$

- Induced subgraph of G with vertices $CS(\mathrm{q})$ is used as the candidate substructures, denoted as G_{sub}





NeurSC

Feature Initialization

$$\boldsymbol{x}_v = f_b(deg_v)||f_b\big(f_l(v)\big)||_{i=1}^{k} MP_{\forall v' \in N^{(i)}(v)} f_b(deg_{v'})||f_b\big(f_l(v')\big)$$

Degree information **Label information** **Neighbor information**

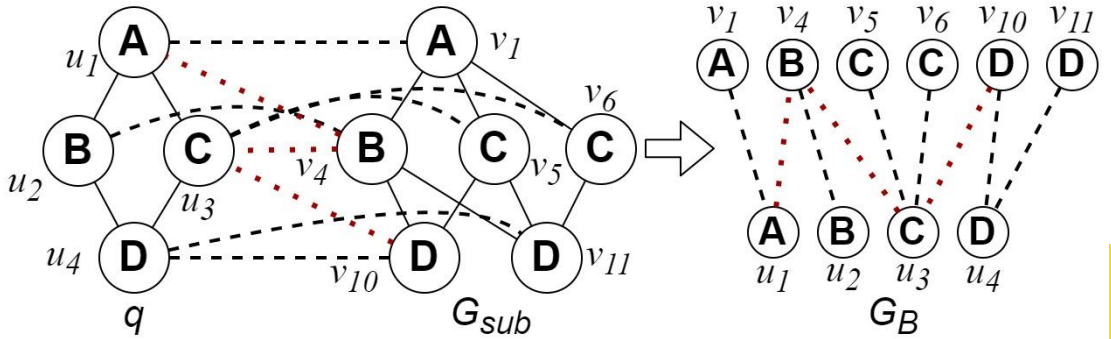
- $||$ denotes the concatenation.
- f_b denotes the binary encoding that converts the decimal digits into binary numbers.
- MP denotes the mean pooling.
- $N^{(i)}(v)$ denotes the i-hop neighbors of v.

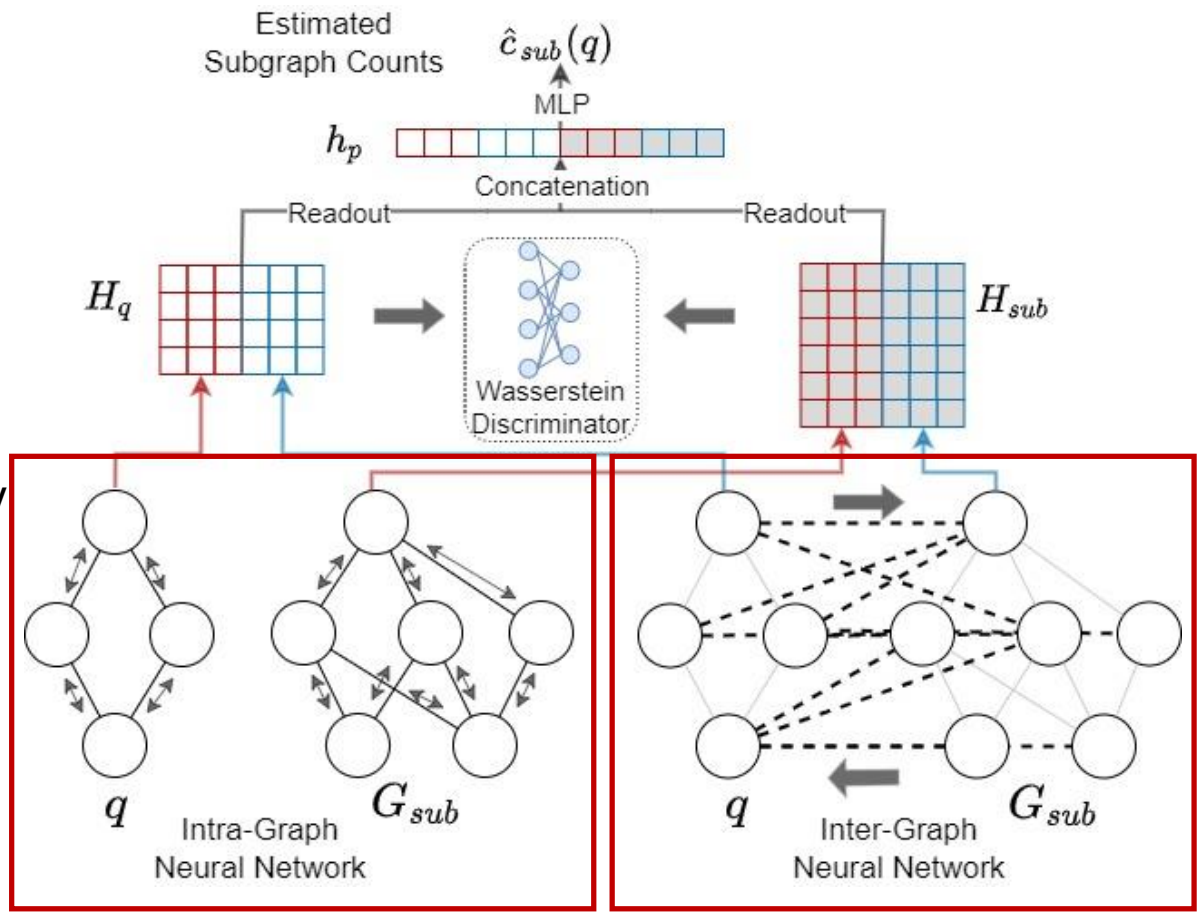
NeurSC

Wasserstein Estimator

- **Intra-Graph Neural Network**
  - For both query graph and substructure.
  - Capture structural and attribute information.
  - $\boldsymbol{h_u^{(k)}} = \boldsymbol{MLP^{(k)}}\left(\left(\boldsymbol{1+\epsilon^{(k)}}\right)\boldsymbol{h_u^{(k-1)}}, \sum_{\boldsymbol{u' \in N_q(u)}} \boldsymbol{h_{u'}^{(k)}}\right)$

- **Inter-Graph Neural Network**
  - Construct a bipartite graph for inter-relationship.
  - Capture the mapping relationship between query vertices and corresponding candidate vertices
  - $\boldsymbol{h_u^{(k)}} = \boldsymbol{\sigma}\left(\boldsymbol{a_{uu}^{(k)}\theta^{(k)}h_u^{(k-1)}}, \sum_{\boldsymbol{v \in N_{G_B}(u)}} \boldsymbol{a_{uv}^{(k)}\theta^{(k)}h_v^{(k)}}\right)$

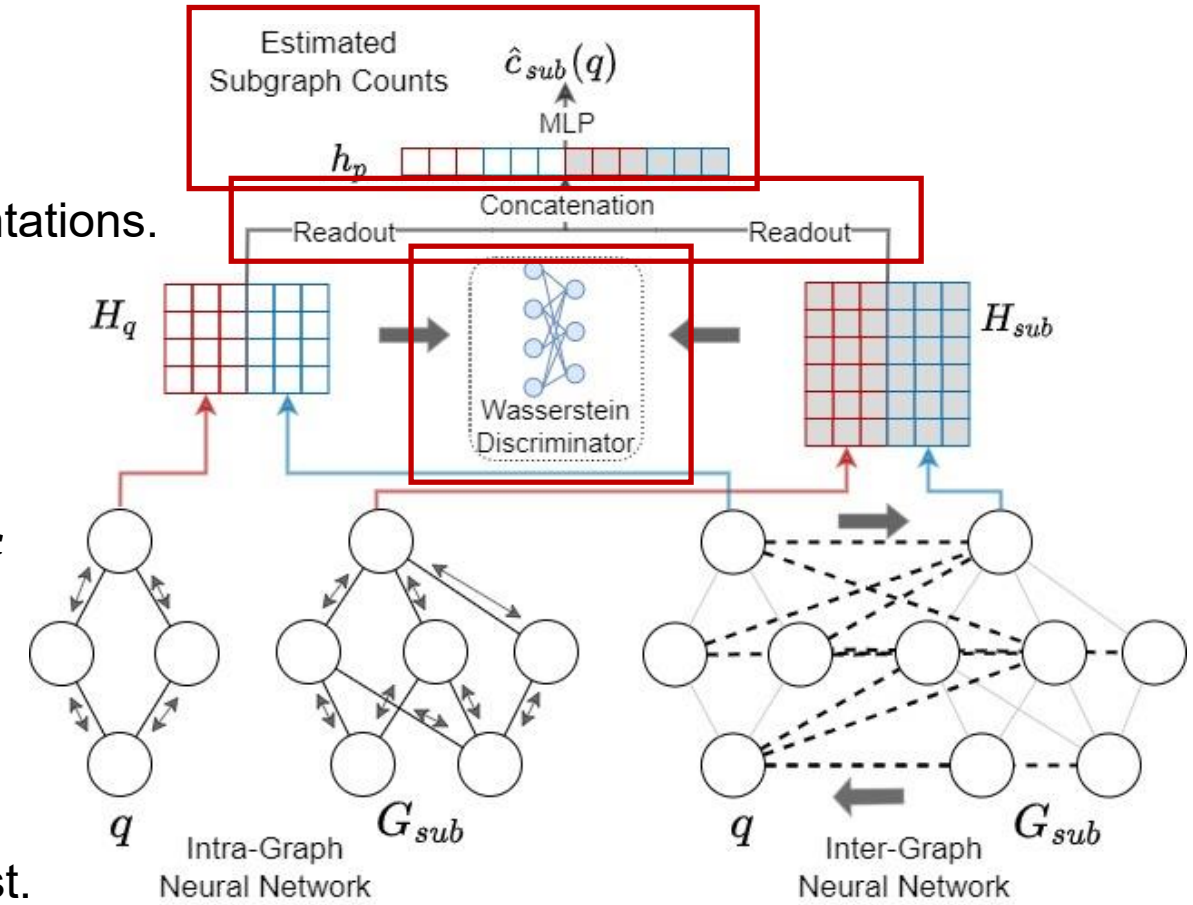




NeurSC

Wasserstein Estimator

- **Readout**
  - Sum Pooling
  - Concatenation of intra- and inter-graph representations.
- **Prediction**
  - Multi-layer perceptron.
- **Wasserstein Discriminator**
  - Minimize Wasserstein distance between q and G
  - Further utilize the vertex correspondence information between q and G
  - $L_w(q, G_{sub}) = \sum_{u \in V'(q)} f_{\omega(h_u)} - \sum_{v \in V'(G_{sub})} f_{\omega(h_v)}$
- **Expressive Power**
  - *WEst* is as powerful as 1-Weisfeiler-Lehman test.



NeurSC

Learning objective and training procedure

- **q-error loss**
  - $L_c(q) = \max\left(\frac{c(q)}{\hat{c}(q)+\varepsilon}, \frac{\hat{c}(q)}{c(q)}\right)$
  - Optimize the model by reducing the prediction error.
- **Wasserstein loss**
  - $L_w(q, G_{sub}) = \sum_{u \in V'(q)} f_{\omega(h_u)} - \sum_{v \in V'(G_{sub})} f_{\omega(h_v)}$
  - Minimize the Wasserstein distance.
- **Overall loss**
  - $L(q) = (1-\beta)L_c(q) - \frac{\beta}{|G_{sub}|}\sum_{g \in G_{sub}} L_w(q, g)$

Algorithm 3: Training Procedure of *WEst*

Input: training query graph set Q_t, data graph G, estimation network f_θ , discriminator f_ω, learning rates α_θ, α_ω, batch size n_{batch}, number of training iterations $iter_\omega$.

Initialize optimizers opt_θ, opt_ω with learning rates α_θ, α_ω.
Separate Q_t into batches $\{Q_b = \{q^{(i)}\}\}$ with n_{batch} query graphs.
for $Q_b \in Q_t$ **do**
  for $q^{(i)} \in Q_b$ **do**
    Generate $\mathcal{G}_{sub}^{(i)}$ for $q^{(i)}$
    $X_q^{(i)} \leftarrow$ initial features of vertices in query graph $q^{(i)}$
    for $j = 1, \ldots, |\mathcal{G}_{sub}^{(i)}|$ **do**
      $X_{sub}^{(j)} \leftarrow$ initial features of vertices in $G_{sub}^{(j)}$
      $H_q^{(i)}, H_{sub}^{(j)}, \hat{c}_j(q^{(i)}) \leftarrow f_\theta(q^{(i)}, G_{sub}^{(j)}, X_q^{(i)}, X_{sub}^{(j)})$
      for $iter_\omega$ **do**
        Sample $V'(q^{(i)})$ and $V'(G_{sub}^{(j)})$
        Update ω by opt_ω minimizing $-\mathcal{L}_w$ in Eq. 9
      Compute $\mathcal{L}_w(q^{(i)}, G_{sub}^{(j)})$
    $\hat{c}(q^{(i)}) = \sum_{j=1}^{|\mathcal{G}_{sub}|} \hat{c}_j(q^{(i)})$
    Compute $\mathcal{L}_c(q^{(i)})$ using Eq. 10
  Update θ by opt_θ with $\sum_{q^{(i)} \in Q_b} \mathcal{L}(q^{(i)})$ defined in Eq. 11

Experiment

Experimental Setup

- **Dataset**
  - *7 data graphs + 5 query sets.*
- **Compared methods**
  - 5 non-learning methods.
  - 2 learning-based methods with 2 variants.
  - 2 variants of *NeurSC* for ablation study.
- **Parameter settings**
  - Initial dimension: 64
  - Hidden and output dimension 128
  - Intra- and Inter-GNN have 2 layers.
  - Prediction network is a 4-layer MLP.
- **Implementation**
  - Substructure extraction: C++
  - *WEst: Python + Pytorch Geometric*

Table 2: Statistics of Data Graphs

Dataset	$\|V\|$	$\|E\|$	$\|L\|$	d
Yeast	3,112	12,519	71	8.0
Human	4.674	86,282	44	36.9
HPRD	9,460	34,998	307	7.4
Wordnet	76,853	120,399	5	3.1
DBLP	317,080	1,049,866	15	6.6
EU2005	862,664	16,138,468	40	37.4
Youtube	1,134,890	2,987,624	25	5.3

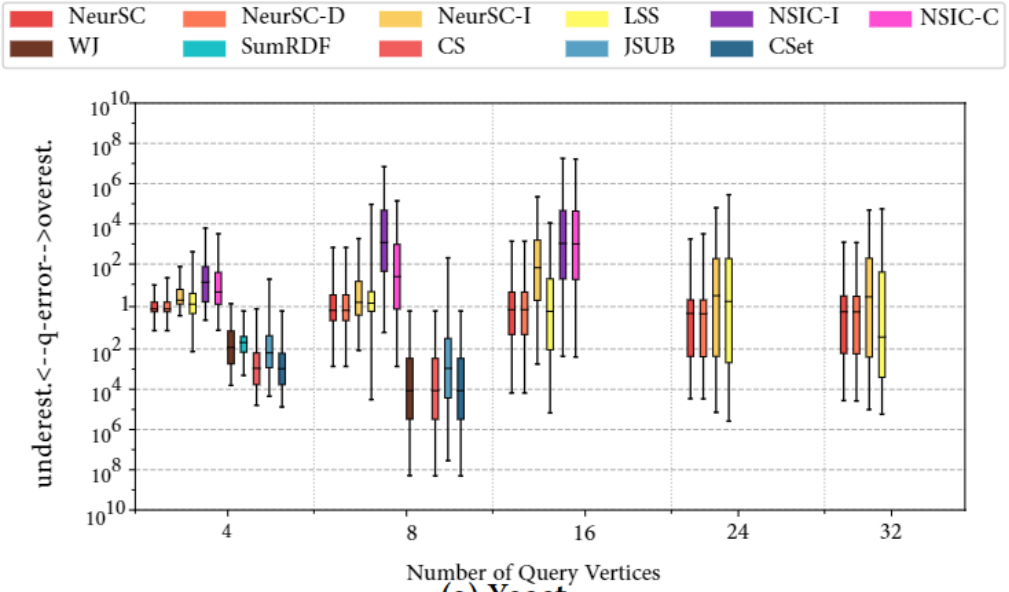
Table 3: Details of Query Graphs

Dataset	# Queries	Query Sizes	Counts Range
Yeast	1,632	{4, 8, 16, 24, 32}	$[10^0, 10^{11}]$
Human	339	{4, 8, 16}	$[10^0, 10^{10}]$
HPRD	1,000	{4, 8, 16}	$[10^0, 10^4]$
Wordnet	600	{4, 8}	$[10^1, 10^9]$
DBLP	600	{4, 8}	$[10^3, 10^8]$
EU2005	372	{4, 8}	$[10^4, 10^9]$
Youtube	811	{4, 8, 16}	$[10^0, 10^{11}]$

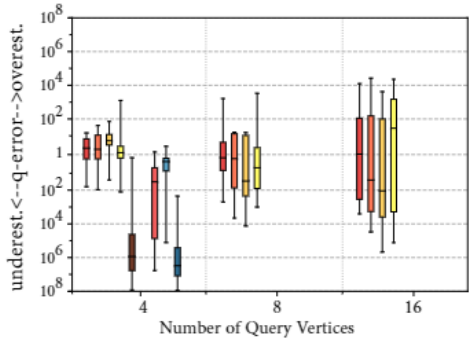
UNSW SYDNEY

Experiment Results

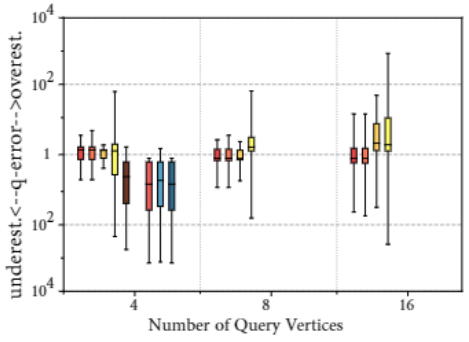
Accuracy Evaluation



(a) Yeast

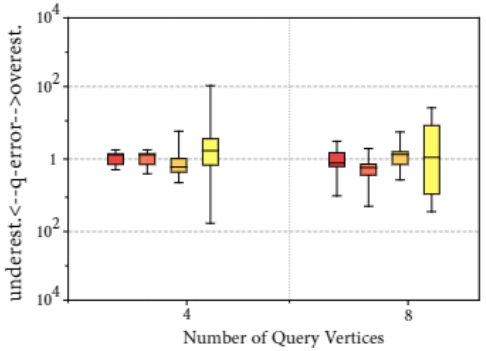


(b) Human

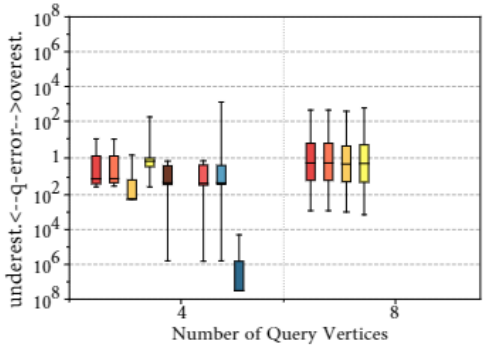


(c) HPRD

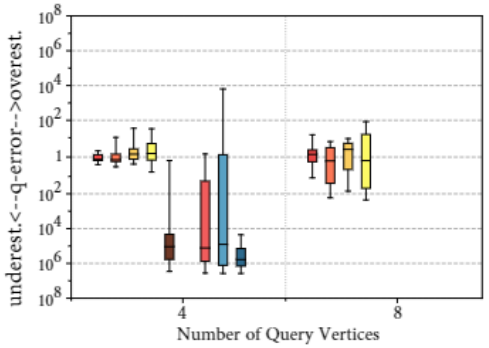
- **Evaluation metric**
  - *q-error:* $\max\left(\frac{\max(1,c)}{\max(1,\hat{c})}, \frac{\max(1,\hat{c})}{\max(1,c)}\right)$



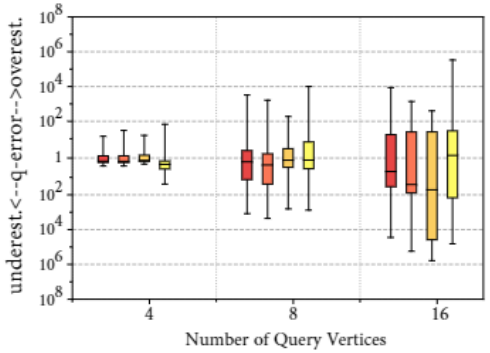
(d) DBLP



(e) Wordnet



(f) EU2005



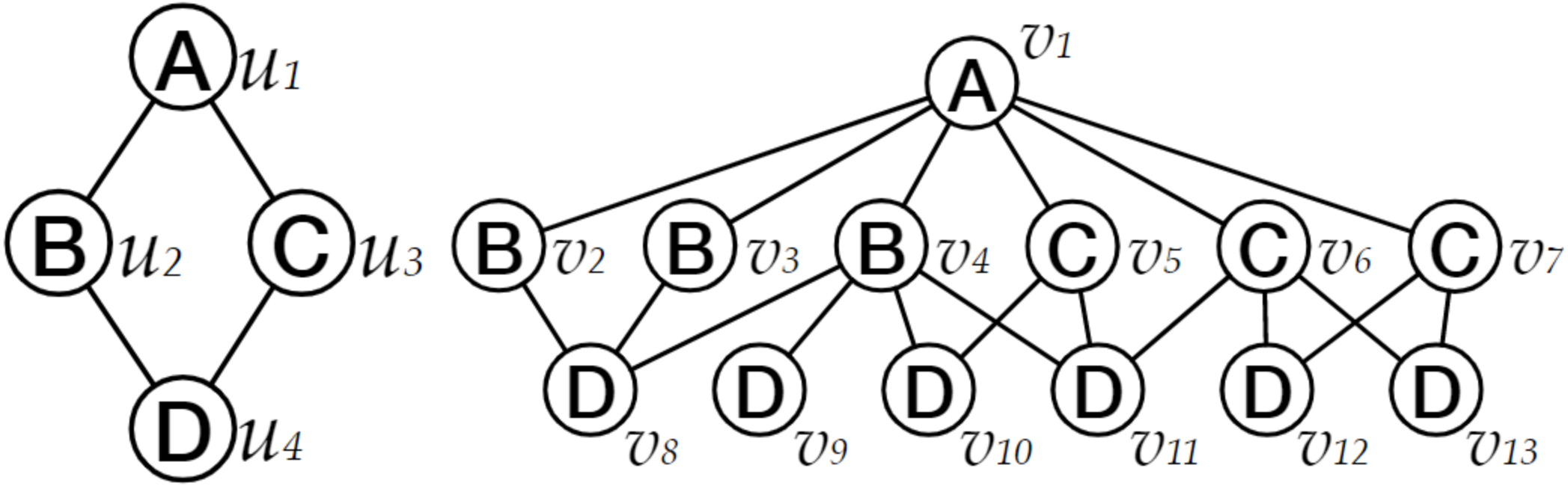
(g) Youtube

Subgraph Matching

The objective of the *subgraph matching* is searching for all *subgraph isomorphisms* from query graph q to data graph G

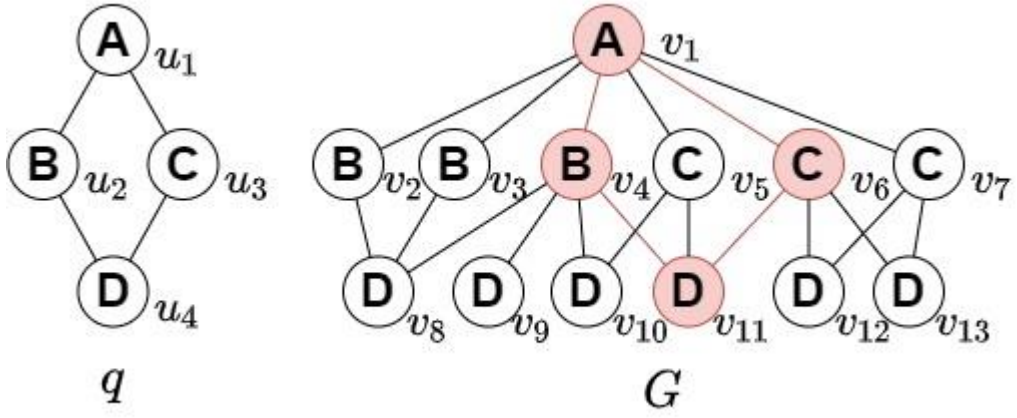
Definition II.1 (Subgraph Isomorphism)**.** Given a query graph $q = (V, E)$ and a data graph $G = (V', E')$, a subgraph isomorphism is an injective function f_{iso} from V to V' such that (1) $\forall v \in V, f_l(v) = f_l(f_{iso}(v))$; and (2) $\forall e_{(u,v)} \in E, e_{(f_{iso}(u), f_{iso}(v))} \in E'$.

Subgraph Isomorphism



(a) Query Graph q (b) Data Graph G

Subgraph Isomorphism



Subgraph isomorphisms

1. $(u_1, u_2, u_3, u_4) \rightarrow (v_1, v_4, v_5, v_{10})$
2. $(u_1, u_2, u_3, u_4) \rightarrow (v_1, v_4, v_5, v_{11})$
3. $(u_1, u_2, u_3, u_4) \rightarrow (v_1, v_4, v_6, v_{11})$

Existing Subgraph Matching Methods

There are two major categories of subgraph matching methods:

- Backtracking-based methods
- Join-based methods

In this work, we focus on the backtracking-based methods.

Existing Subgraph Matching Methods

The backtracking-based methods can be partitioned in three main phases:

1. The complete candidate vertex set generation.
2. Matching order generation.
3. Matching enumeration.

Backtracking-based Methods

Complete candidate vertex set generation is to filter out the unpromising vertices, and hence reduce the search space before the enumeration process begins.

Definition II.2 (Complete Candidate Vertex Set $\mathcal{C}$). Given q and G, a complete candidate vertex set $C(u)$ of $u \in V(q)$ is a set of data vertices such that for each $v \in V(G)$, if (u, v) exists in a match from q to G, then $v \in C(u)$.

Backtracking-based Methods

Matching order generation phase generates the matching order φ to guide the enumeration of matched subgraphs.

Definition II.3 (Matching Order). A matching order ϕ is a permutation (i.e., sequence) of query graph's vertex set $V(q)$.

Backtracking-based Methods

The enumeration procedure finds all matches of the query subgraph q in the data graph G with given matching order φ.

Definition II.5 (Enumeration Procedure). An enumeration procedure is performed recursively to find subgraph matches f_{iso} with given matching order ϕ and candidate vertex set C.

Subgraph Matching

Subgraph matching has wide applications such as query in graph database and biological graph analytics.

However, it has been proven that the subgraph matching is *NP-complete*. We cannot optimize the worst-case time cost.

In this work, we aim to reduce the enumeration time *on the average case* by proposing a novel query vertex ordering method.

- Background
- Motivation
- Framework
- Feature Representations
- Query Vertex Ordering as Markov Decision Process
- Policy Training
- Experiments

Limitations of Existing Order Generation Methods

The existing subgraph matching methods usually generate the matching order based on the heuristic values, here are some examples:

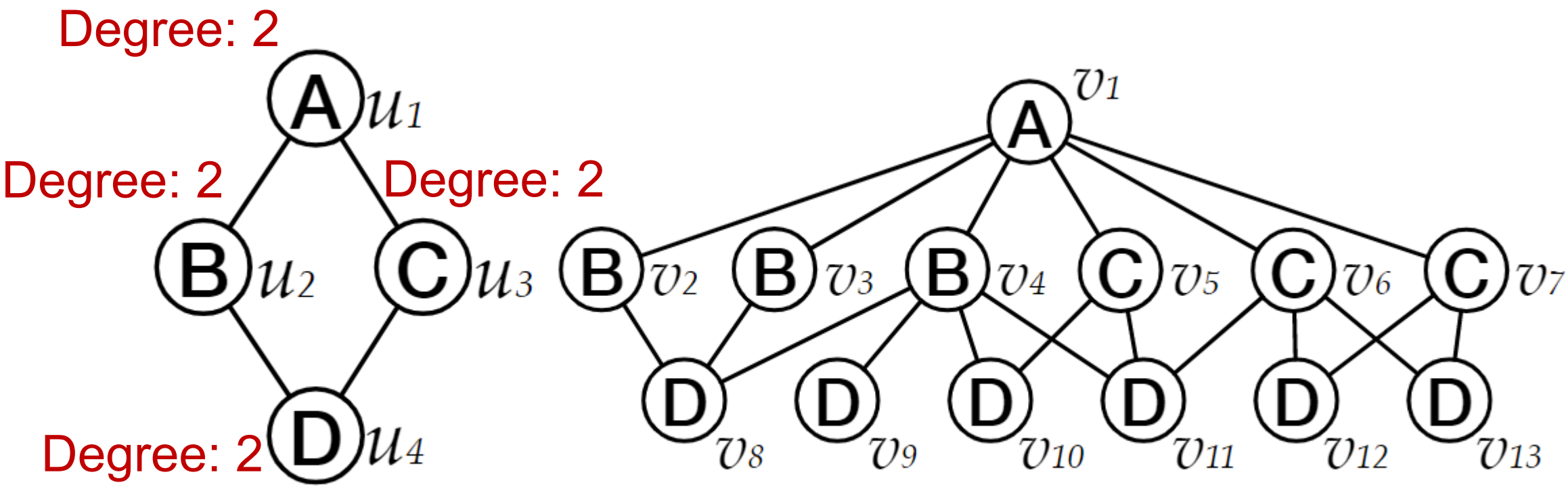
- Infrequent edge first ordering
- Infrequent label first ordering
- Path-based ordering.

Limitations of Existing Order Generation Methods

Two major limitations:

- Cannot fully use the graph information.
- Greedy heuristics can lead to local optimum.

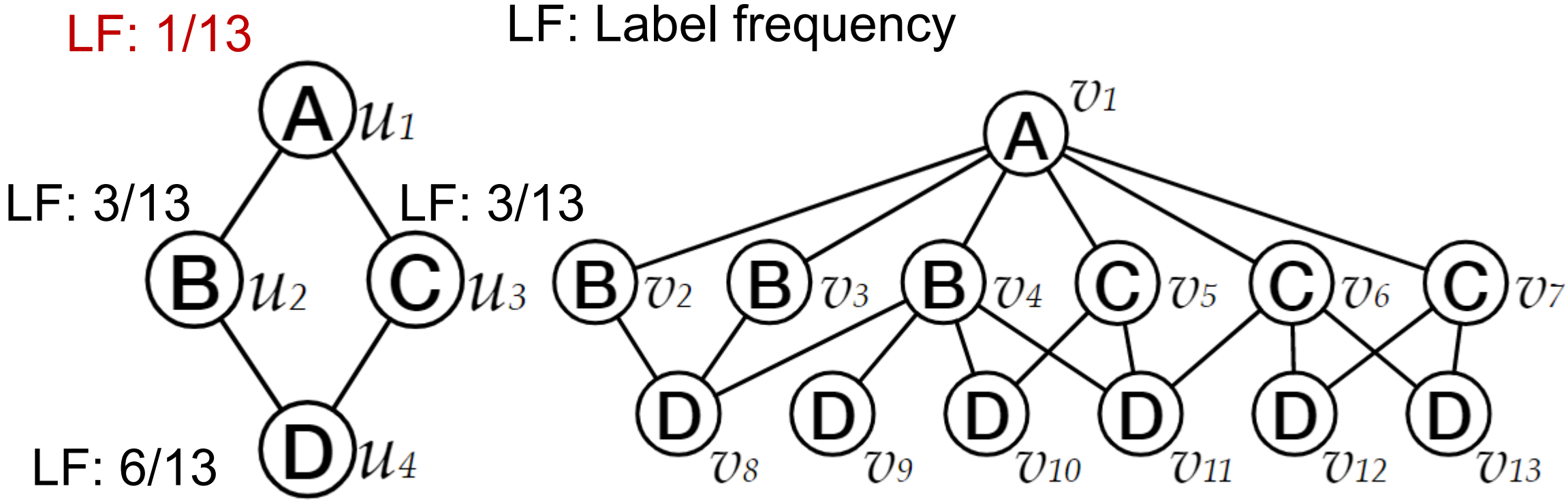
If ordering based on degree (RI)



(a) Query Graph q (b) Data Graph G

If ordering based on label frequency

LF: Label frequency



(a) Query Graph q (b) Data Graph G

Greedy heuristics can lead to local optimum

The heuristic-based greedy methods can reduce the most redundant intermediate results.

However, these methods cannot consider the long-term query time cost.

The exact optimal order can only be found after all possible order permutations are evaluated.

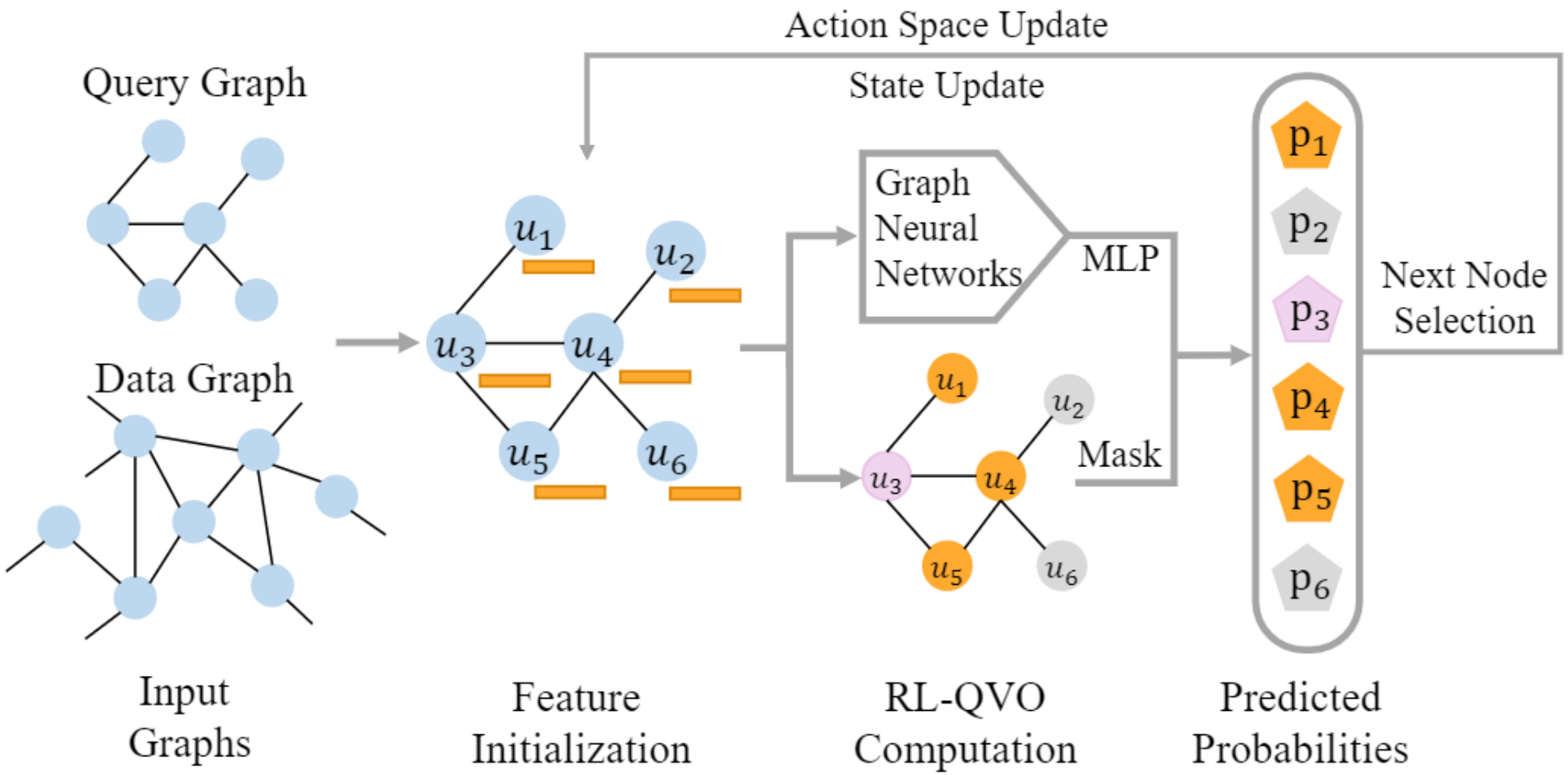
Our solutions

Motivated by the aforementioned limitations, we proposed the following approaches:

- Capture the graph information with graph neural network.
- Try to approach the global optimal with reinforcement learning.

- Background
- Motivation
- Framework
- Feature Representations
- Query Vertex Ordering as Markov Decision Process
- Policy Training
- Experiments

Framework



Our recent work
- “Reinforcement learning based query vertex ordering model for subgraph matching” -- ICDE 2022

- Background
- Motivation
- Framework
- Feature Representations
- Query Vertex Ordering as Markov Decision Process
- Policy Training
- Experiments

Feature Initialization

We use the important statistical heuristics of query vertices to initialize the input query representations.

Based on the input features, our model can fully exploit the information within the features while preserving the relations between the query and data graphs.

Feature Initialization

- Degree of node:

$$\boldsymbol{h}_u^{(0)}(1) = degree(u)/\alpha_{degree},$$

- Label of node:

$$\boldsymbol{h}_u^{(0)}(2) = label(u)$$

- Query node ID:

$$\boldsymbol{h}_u^{(0)}(3) = id(u)$$

Feature Initialization

- Frequency of data vertices with greater degree than the query node

$$\boldsymbol{h}_u^{(0)}(4) = |\{v \in G | d(u) < d(v)\}| \,/\, (|V(G)| \times \alpha_d);$$

- Frequency of data vertex with same label as query vertex:

$$\boldsymbol{h}_u^{(0)}(5) = |\{v \in G | L(u) = L(v)\}| \,/\, (|V(G)| \times \alpha_l);$$

Feature Initialization

Lastly, we put two indicator variables in the initial feature:

- Number of unordered vertices:

$$\boldsymbol{h}_u^t(6) = |V(q)| - t + 1$$

- Trailing indicator that shows whether the node has been ordered:

$$\boldsymbol{h}_u^t(7) = \mathbb{1}(u \in \phi_{t-1})$$

- Background
- Motivation
- Framework
- Feature Representations
- Query Vertex Ordering as Markov Decision Process
- Policy Training
- Experiments

Query Vertex Ordering as Markov Decision Process

To exploit the reinforcement learning, we need to model our query vertex ordering problem as a Markov decision process (MDP).

Specifically, we need to define:

- State space
- Action space
- Action probability
- Reward

State Space

In this work, the state space is defined as the all (partial) vertex order sequence.

Specifically, we use ϕ_t (the order at time step t) to denote the state at time step t.

Action Space

With the order ϕ_t (state of the MDP), we define the action space as $N(\phi_t) = \{N(u) | \forall u \in \phi_t, N(u) \notin \phi_t\}$ which is the neighbor vertices set of the ordered vertices in ϕ_t .

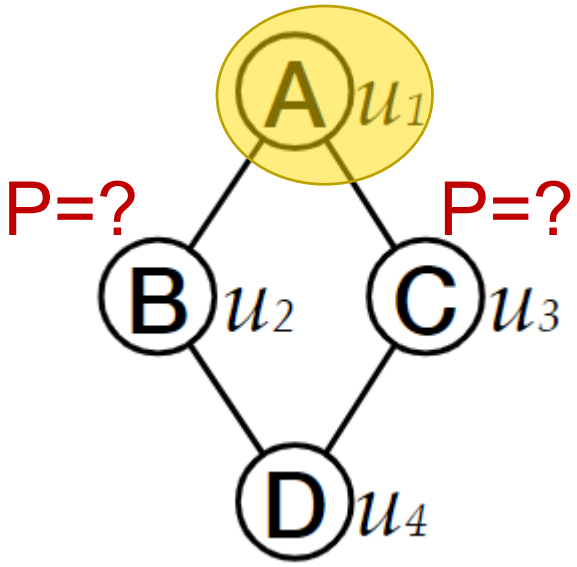
Consequently, the action at each time step t is to select a node from the action space (neighbor set $N(\phi_t)$).

Action Probability

With the state and corresponding action space.

The key problem is to compute the probabilities for each action in the action space.

To this end, we design a Graph Neural Network (GNN)-based policy network.



Policy Network

Graph convolutional network:

$$\boldsymbol{H}^{(l+1)} = \sigma(\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)})$$

Action probability with the state at time step t:

$$\mathbb{P}_{u'}^{(t)} = \pi(\cdot|S^{(t)}) = Softmax(mask_{u' \in AS(t)}(\boldsymbol{W}_2 \cdot \sigma(\boldsymbol{W}_1 \boldsymbol{h}_{u'}^{(t)})))$$

Reward Design

Three main rewards:

- Enumeration reward
- Step-wise validate reward
- Entropy reward

Enumeration Reward

Definition II.6 (Enumeration Number)**.** An enumeration number $\#_{enum}$ is the number of *recursive* calls of the enumeration procedure to find all matches with given q, G, ϕ and C.

The reduced enumeration number:

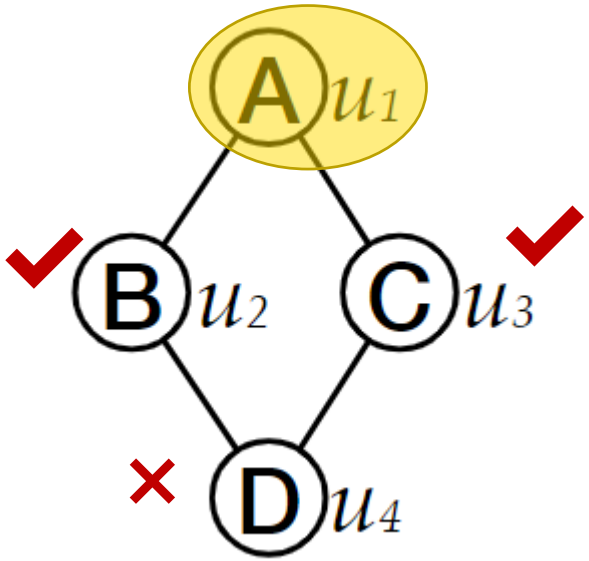
$$\Delta\#_{enum} = \#_{enum}(\phi) - \#_{enum}(\phi_{base})$$

Enumeration reward:

$$r_{enum} = f_{enum}(\Delta\#_{enum}).$$

Other rewards

Step-wise validation: $r_{val,t}$



Entropy reward:

$$r_{h,t} = H(P_{\pi_\theta}(\phi_t, N(\phi_t)))$$

Ensure the model can perform more actions.

Reward Design

With the aforementioned rewards, our overall rewards at time step t is as follows:

$$R_t = r_{enum} + \beta_{val} \cdot r_{val,t} + \beta_h \cdot r_{h,t}$$

The overall reward is as follows:

$$R_{q,\theta} = \sum_{t=1}^{|V(q)|} \gamma^t R_t,$$

- Background
- Motivation
- Framework
- Feature Representations
- Query Vertex Ordering as Markov Decision Process
- Policy Training
- Experiments

Policy Training

In this work, we use the proximal policy optimization (PPO) to train our policy network with the following loss function:

$$J_r^{(t)}(\theta) = \sum_{(a_t, s_t)} min(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta'}(a_t|s_t)} r_t(\theta), clip(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta'}(a_t|s_t)}, 1-\epsilon, 1+\epsilon) r_t(\theta))$$

$$J(\theta) = \sum_{t=1}^{|V(q)|} J_r^{(t)}(\theta)$$

- Background
- Motivation
- Framework
- Feature Representations
- Query Vertex Ordering as Markov Decision Process
- Policy Training
- Experiments

Experiment Setup

Compared methods:
QuickSI
RI
VF2++
VEQ
Hybrid
RL-QVO

Dataset Statistics

Dataset	$\|V\|$	$\|E\|$	$\|L\|$	d
Citeseer	3,327	4,732	6	1.4
Yeast	3,112	12,519	71	8.0
DBLP	317,080	1,049,866	15	6.6
Youtube	1,134,890	2,987,624	25	5.3
Wordnet	76,853	120,399	5	3.1
EU2005	862,664	16,138,468	40	37.4

Evaluation Metrics

- Query processing time
- Enumeration time

Average Query Processing Time

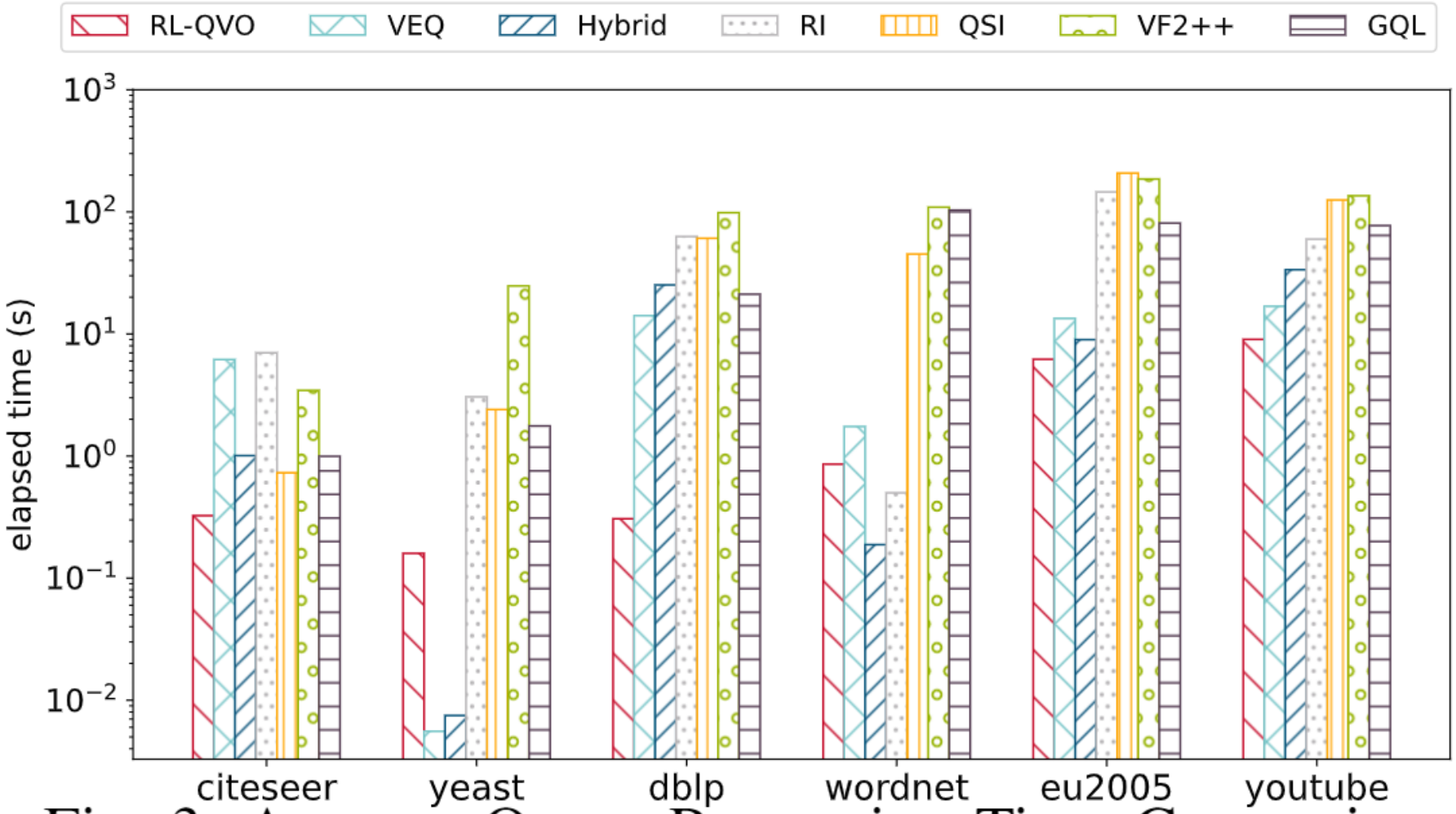
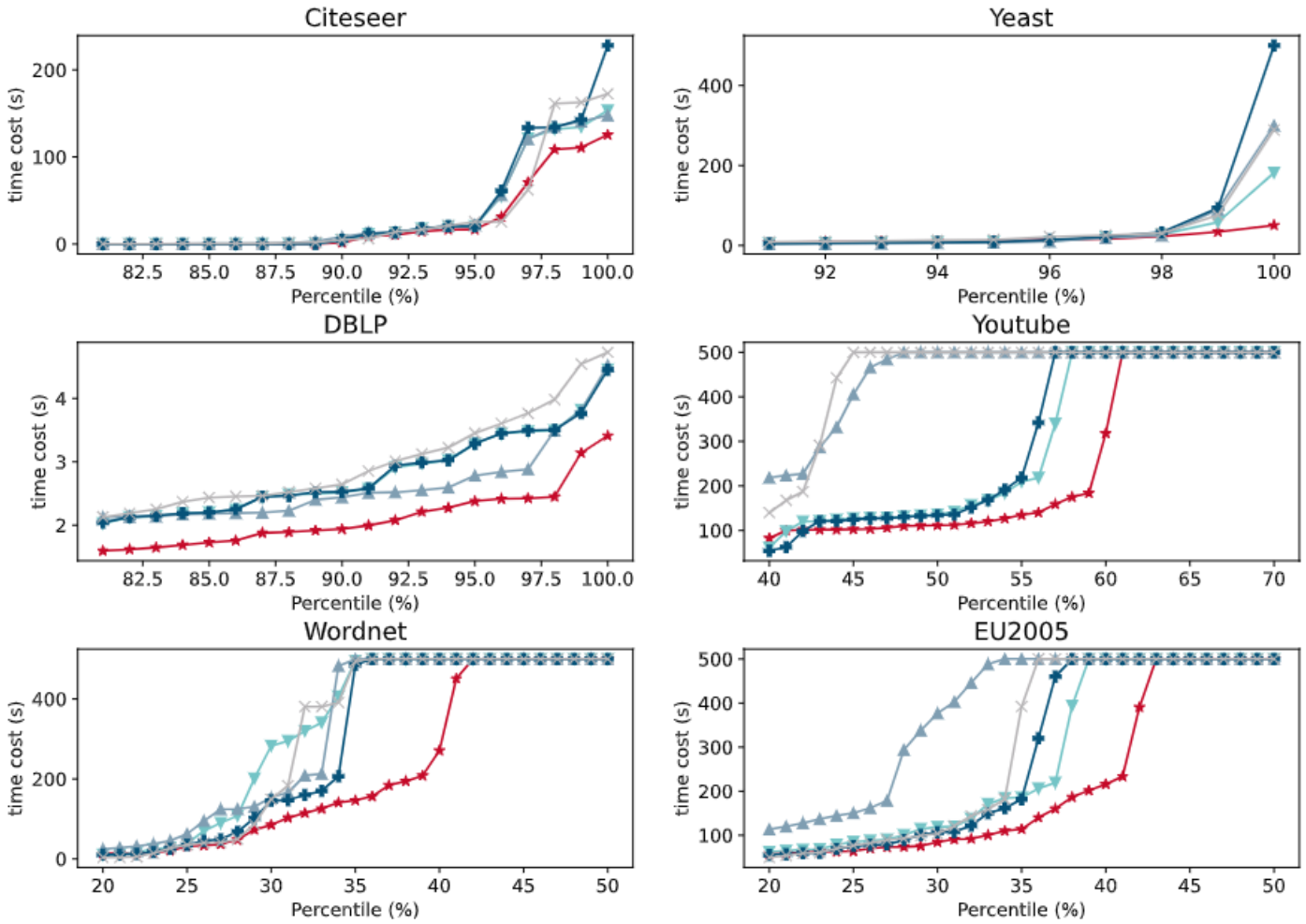
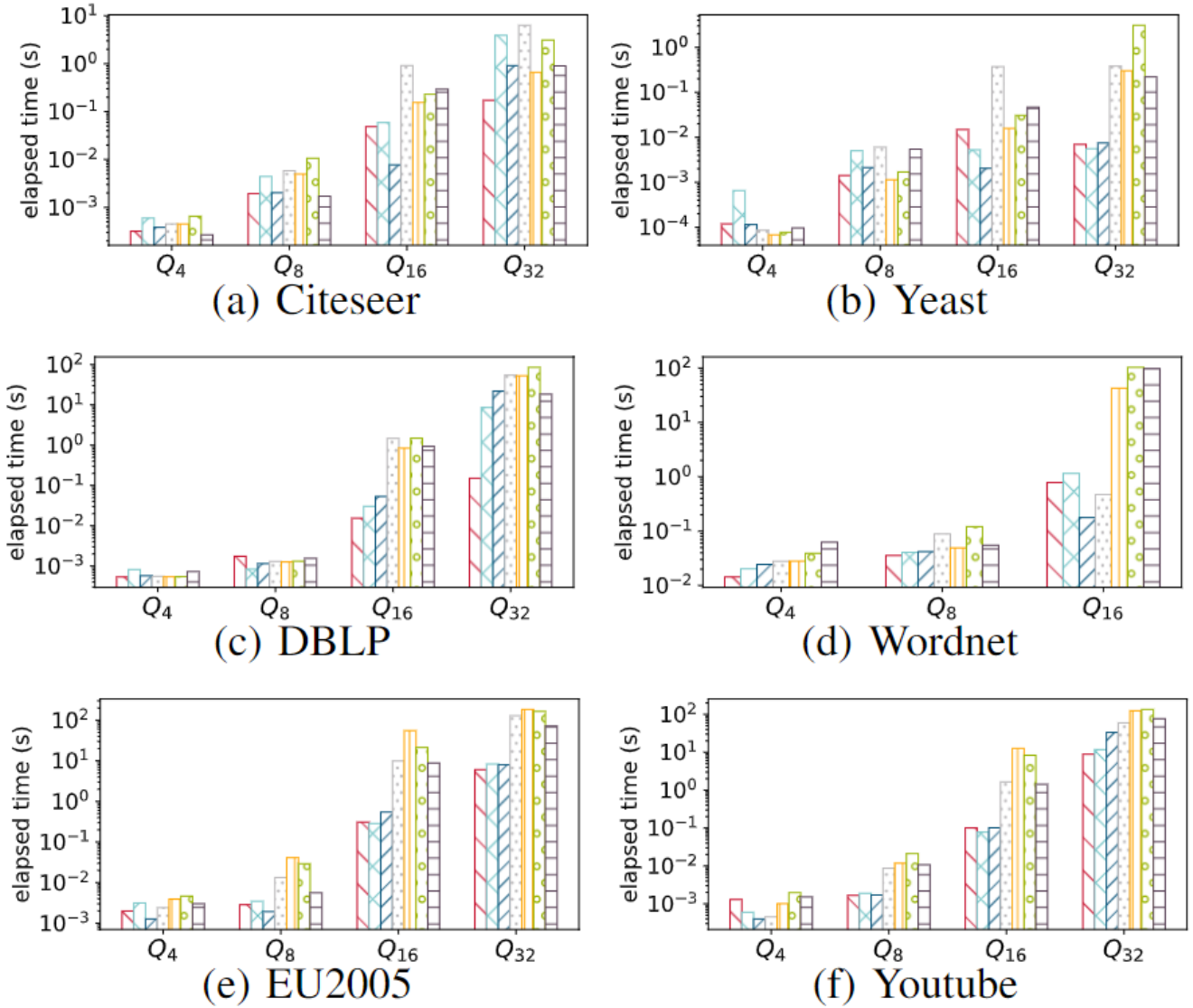


Fig. 3: Average Query Processing Time Comparison

Query Processing Time Percentile Comparison



Enumeration Time Comparison

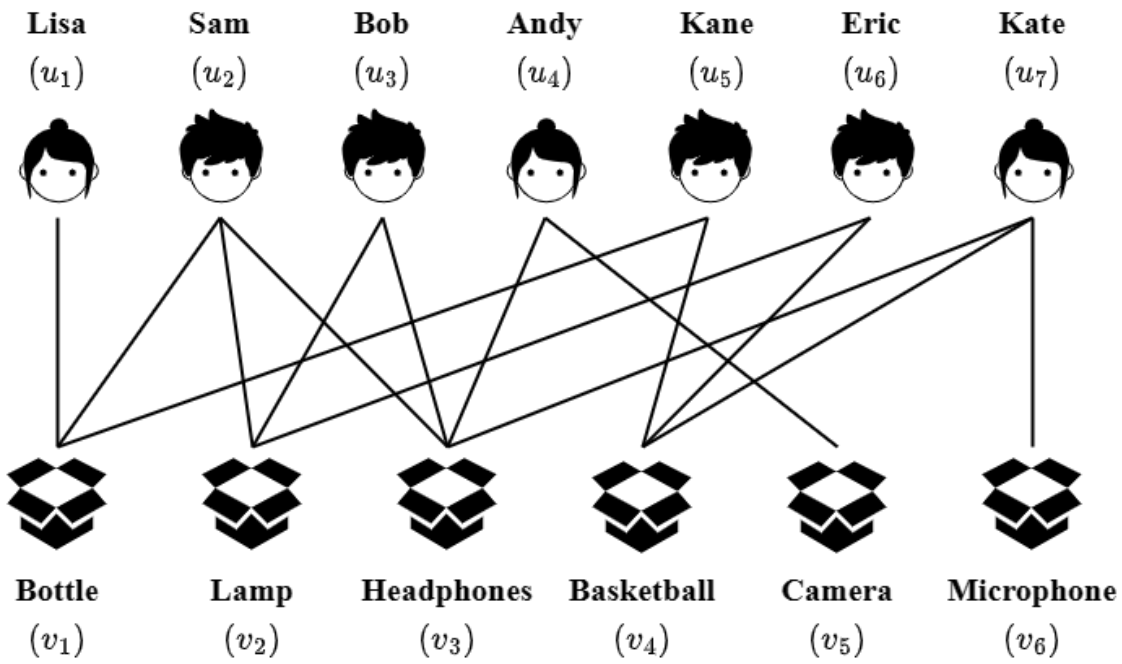


Fraud Detection

Background

Attributed Bipartite Graph

An attributed bipartite graph is a type of graph which consists of two sets of vertices that are linked by edges. The vertices have additional attributes, making this graph particularly useful for **representing information in the field of e-commerce**.



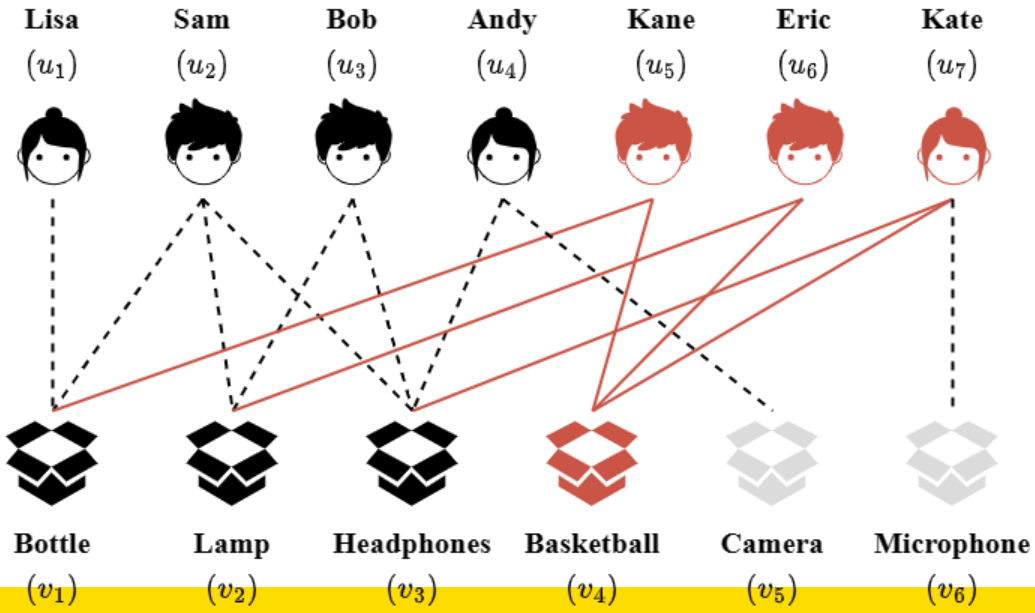
Background

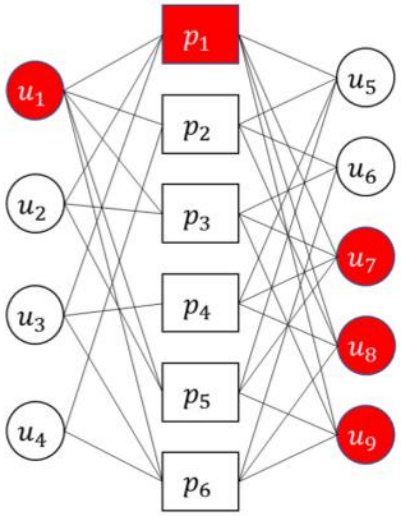
Group-based Frauds on Attributed Bipartite Graphs

Group-based fraud is becoming increasingly common:

"Ride Item's Coattails" attack (edge classification)

Sockpuppet-based Targeted Attack on Reviewing Systems (STARS attack) (vertex classification)





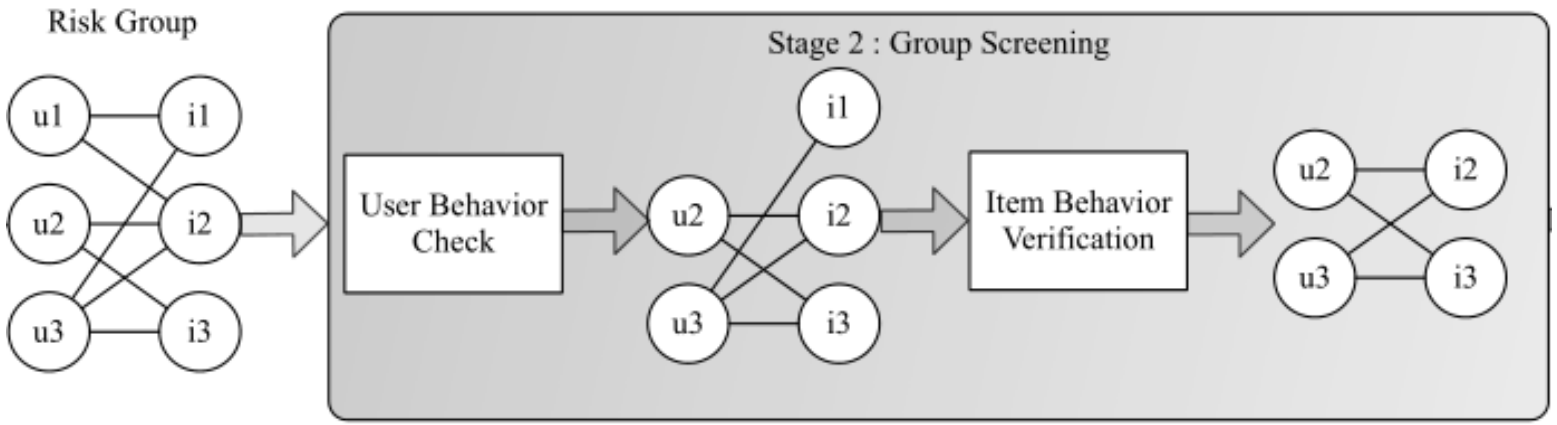
Rating	Score	Rating	Score
(u_1, p_1)	1	(u_5, p_5)	0.5
(u_1, p_2)	-1	(u_6, p_2)	0
(u_1, p_3)	0.5	(u_6, p_3)	1
(u_1, p_5)	0.5	(u_6, p_6)	1
(u_1, p_6)	0.5	(u_7, p_1)	1
(u_2, p_1)	0	(u_7, p_3)	1
(u_2, p_3)	1	(u_7, p_4)	1
(u_2, p_5)	1	(u_7, p_6)	0.5
(u_3, p_1)	0	(u_8, p_1)	1
(u_3, p_4)	-0.5	(u_8, p_2)	0
(u_3, p_6)	0.5	(u_8, p_4)	0.5
(u_4, p_2)	-1	(u_8, p_6)	1
(u_4, p_6)	1	(u_9, p_1)	1
(u_5, p_1)	-1	(u_9, p_3)	0.5
(u_5, p_2)	-1	(u_9, p_5)	0.5
(u_5, p_4)	0	(u_9, p_6)	0.5

Background

SOTA method for “Ride Item’s Coattails” attack

RICD ((α, $k1$, $k2$)-biclique): **fraud detection method** for “Ride Item’s Coattails” attack. Can only utilize structural information.

Tianchi competition winner’s algorithm: **classification method**. Can only use attribute information.



Background

SOTA method for STARS attack

RTV: fraud detection method for Sockpuppet-based Targeted Attack on Reviewing Systems (STARS).
Unable to make good use of label information.

	Algorithm RTV **Input:** Rating graph $G = (\mathcal{U} \cup \mathcal{P}, \mathcal{R}, \mathrm{sc})$, weights $\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma_1, \gamma_2, \gamma_3, \gamma_4$, threshold ϵ **Output:** $\mathrm{fair}(u)\ \forall u \in \mathcal{U}$, $\mathrm{good}(p)\ \forall p \in \mathcal{P}$, $\mathrm{rel}(u,p)\ \forall (u,p) \in \mathcal{R}$
1	**for each** $u \in \mathcal{U}$, $\mathrm{fair}_0(u) \leftarrow \mathrm{norm}(u)$
2	**for each** $p \in \mathcal{P}$, $\mathrm{good}_0(p) \leftarrow \mathrm{norm}(p)$
3	**for each** $(u,p) \in \mathcal{R}$, $\mathrm{rel}_0(u,p) \leftarrow \mathrm{norm}(u,p)$
4	$\mu_f \leftarrow \frac{\sum_{u \in \mathcal{U}} \mathrm{fair}_0(u)}{\lvert\mathcal{U}\rvert}$, $\mu_g \leftarrow \frac{\sum_{p \in \mathcal{P}} \mathrm{good}_0(p)}{\lvert\mathcal{P}\rvert}$
5	$t \leftarrow 1$
6	**for each** $u \in \mathcal{U}$, $\mathrm{fair}_t(u) \leftarrow$ value computed as specified in Section 4.1, with $\mathrm{rel}(u,p) = \mathrm{rel}_{t-1}(u,p)$
7	**for each** $p \in \mathcal{P}$, $\mathrm{good}_t(p) \leftarrow$ value computed as specified in Section 4.1, with $\mathrm{rel}(u,p) = \mathrm{rel}_{t-1}(u,p)$
8	**for each** $(u,p) \in \mathcal{R}$, $\mathrm{rel}_t(u,p) \leftarrow$ value computed as specified in Section 4.1, with $\mathrm{fair}(u) = \mathrm{fair}_t(u)$
9	$\Delta \leftarrow \max\left(\sum_{u \in \mathcal{U}} \lvert\mathrm{fair}_t(u) - \mathrm{fair}_{t-1}(u)\rvert, \sum_{p \in \mathcal{P}} \lvert\mathrm{good}_t(p) - \mathrm{good}_{t-1}(p)\rvert, \sum_{(u,p) \in \mathcal{R}} \lvert\mathrm{rel}_t(u,p) - \mathrm{rel}_{t-1}(u,p)\rvert\right)$
10	**if** $\Delta > \epsilon$ **or** $t = 1$ **then** $t \leftarrow t + 1$ and go to Line 6
11	**return** $\mathrm{fair}_t(u)\ \forall u \in \mathcal{U}$, $\mathrm{good}_t(p)\ \forall p \in \mathcal{P}$, $\mathrm{rel}_t(u,p)\ \forall (u,p) \in \mathcal{R}$

Background

Existing methods

Classification Methods:

- Imbalanced labeled vertices, community information.

Cohesive Subgraph Mining Methods:

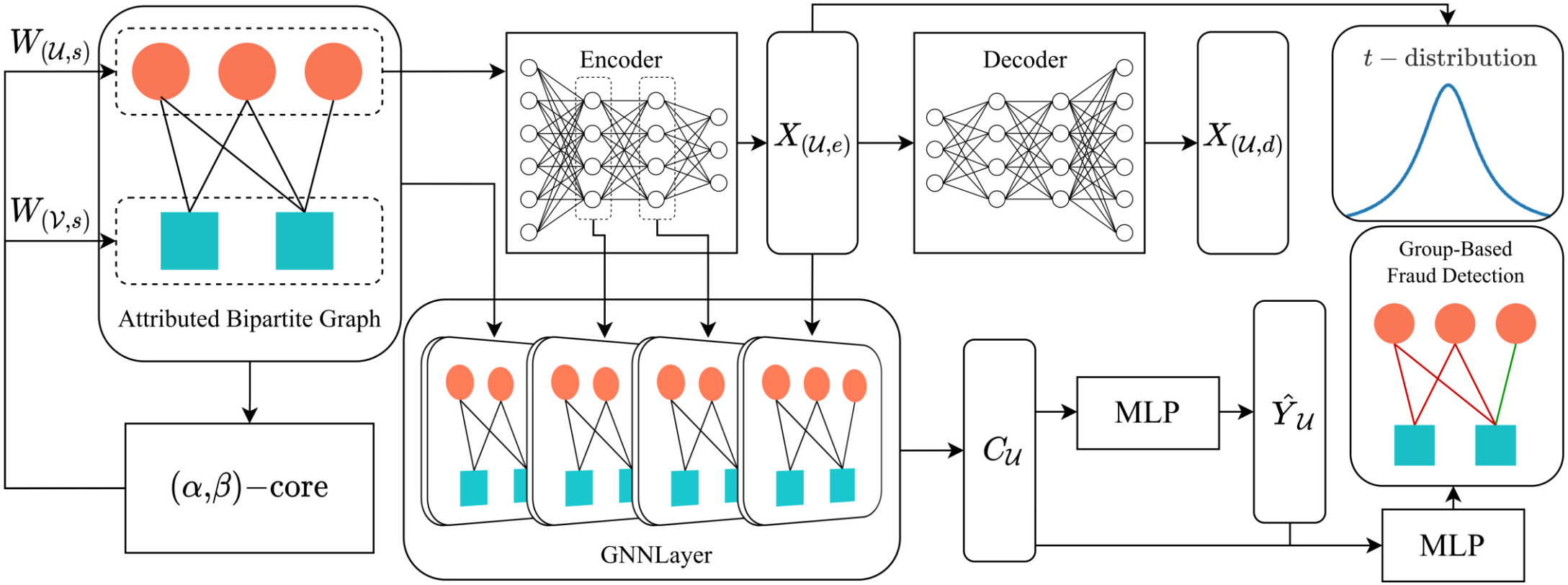
- Attribute and label information, suffer from NP-completeness.

Fraud Detection Methods:

- Global topological and attribute information, label information, manual parameter setting.

Overview

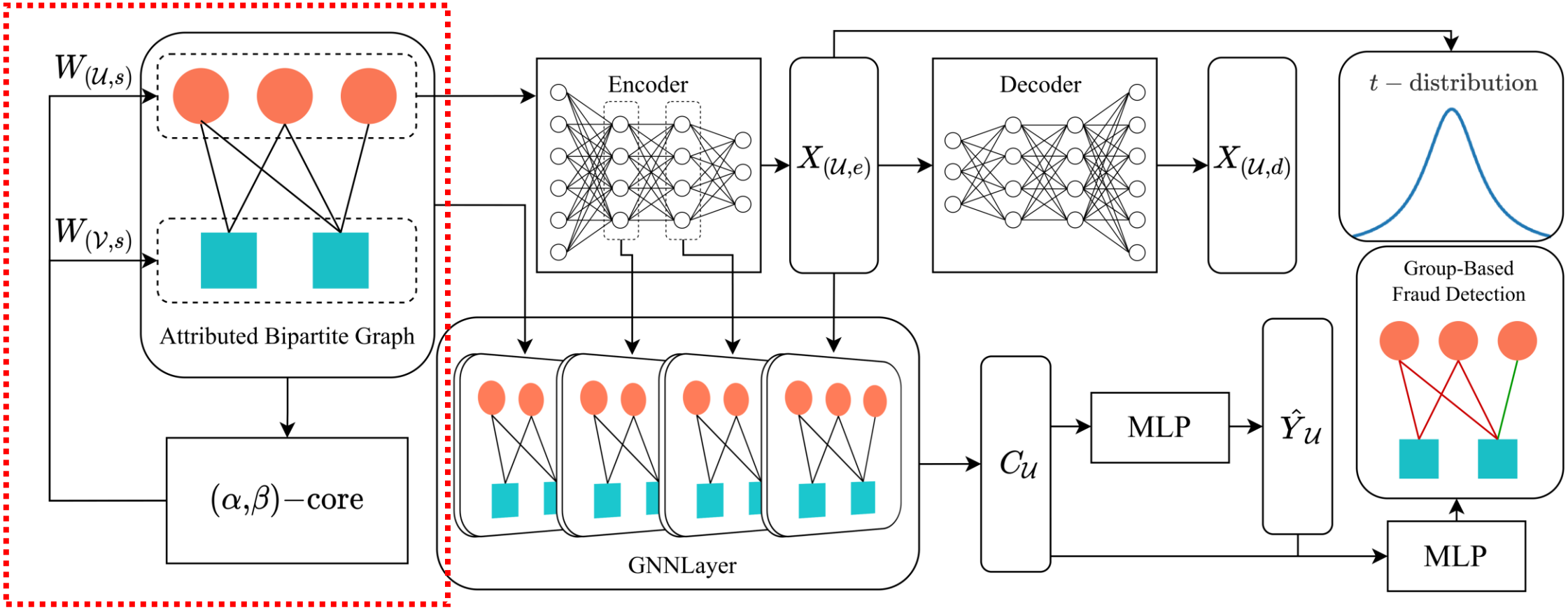
Group-based Fraud Detection method: GFDN



Our recent work
- “Group-based Fraud Detection Network on e-Commerce Platforms” -- KDD 2023

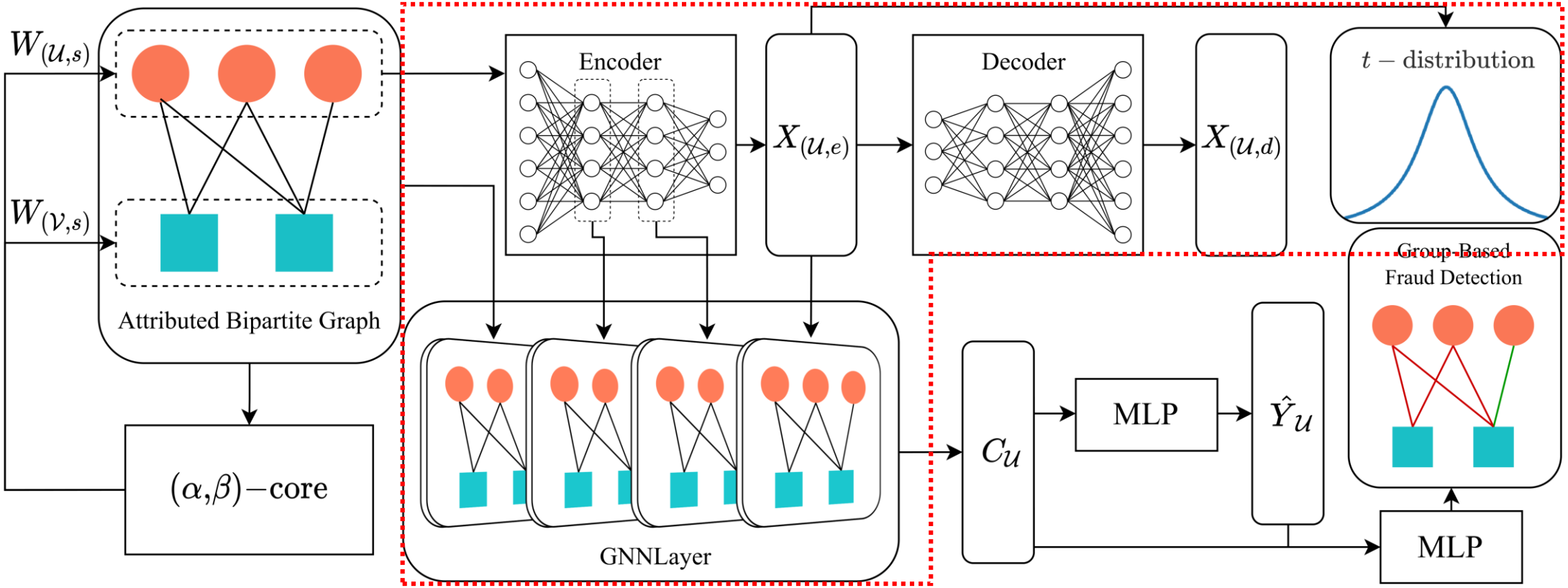
Overview

Group-based Fraud Detection method: GFDN



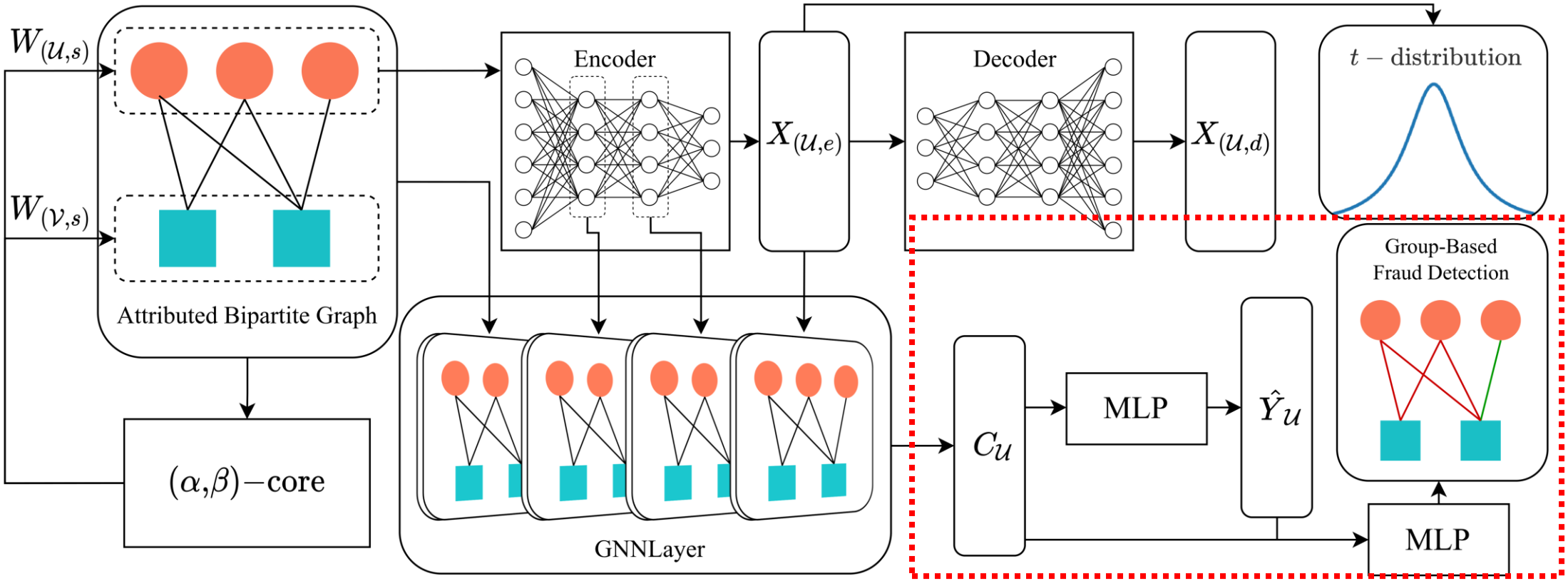
Overview

Group-based Fraud Detection method: GFDN



Overview

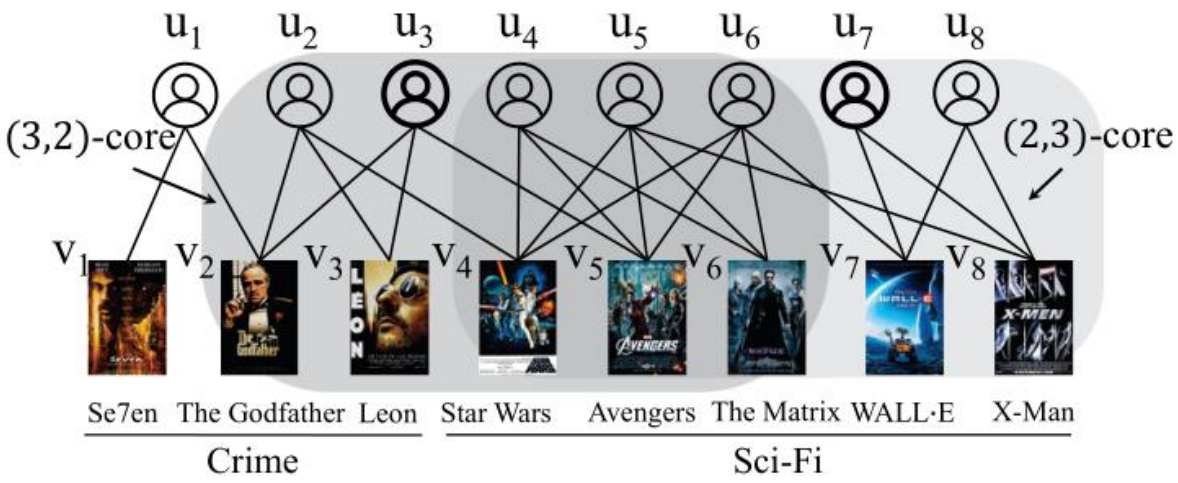
Group-based Fraud Detection method: GFDN



GFDN

Structural Feature Initialization

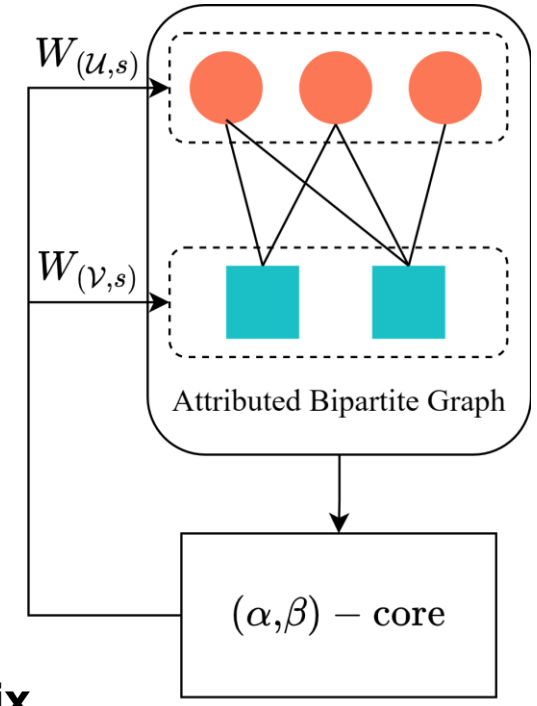
(α, β)-core:
Given a bipartite graph G and integers α, $\beta \in Z^+$, (α, β)-core of G is denoted as G ′ which consists of two vertex sets U′ ⊆ U and V′ ⊆ V.
The (α, β)-core G ′ is a maximal bipartite subgraph induced by U′ ∪ V′ from G in which all the vertices in U′ have degrees at least α and all the vertices in V′ have degrees at least β.



GFDN

Structural Feature Initialization

GFDN will generate structural features for vertices based on their existence in different (α, β)-core.



$$\hat{X}_{(\mathcal{U},s)} = X_{(\mathcal{U},s)} \odot (I_{\mathcal{U}} W_{(\mathcal{U},s)}),\ \hat{X}_{(\mathcal{V},s)} = X_{(\mathcal{V},s)} \odot (I_{\mathcal{V}} W_{(\mathcal{V},s)})$$

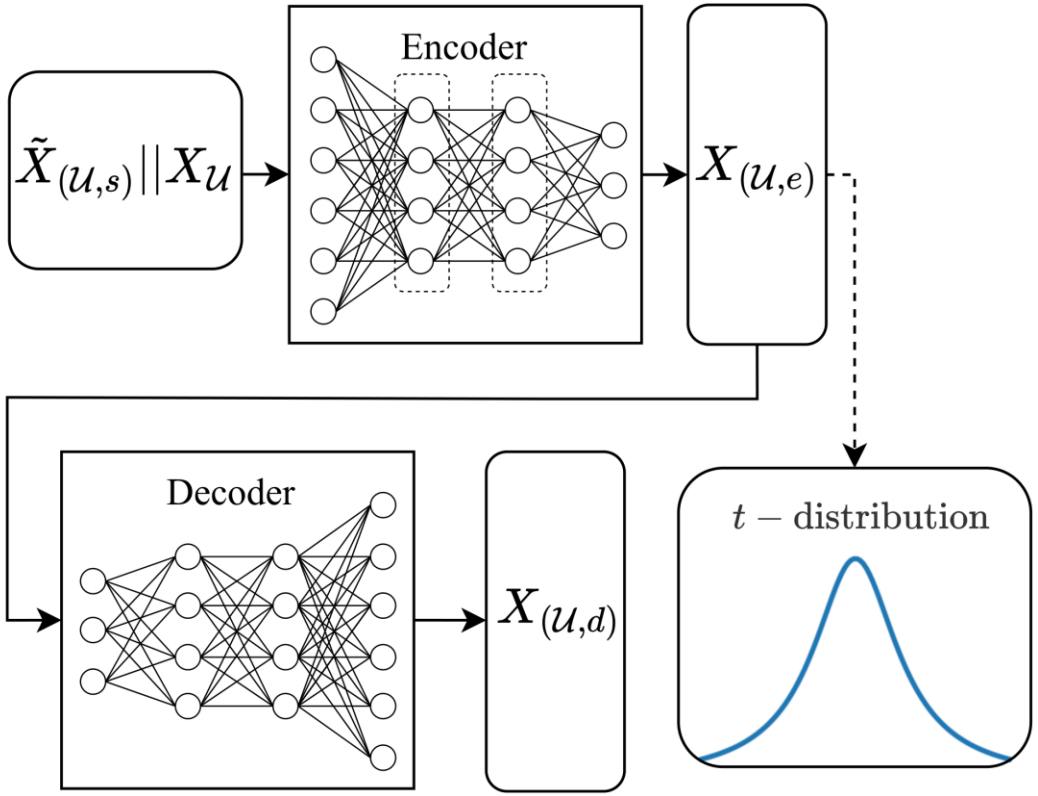
Structural Features **Element-wise Product** **All-ones Vector** **Weight Matrix**

GFDN

Fraudster Community Detection

BDCN - Autoencoder:
Autoencoder in Bipartite Deep Clustering Network (BDCN) can:
1. preserving both structural and attribute information from the input features.
2. Generate high-quality community representation for customer vertices.

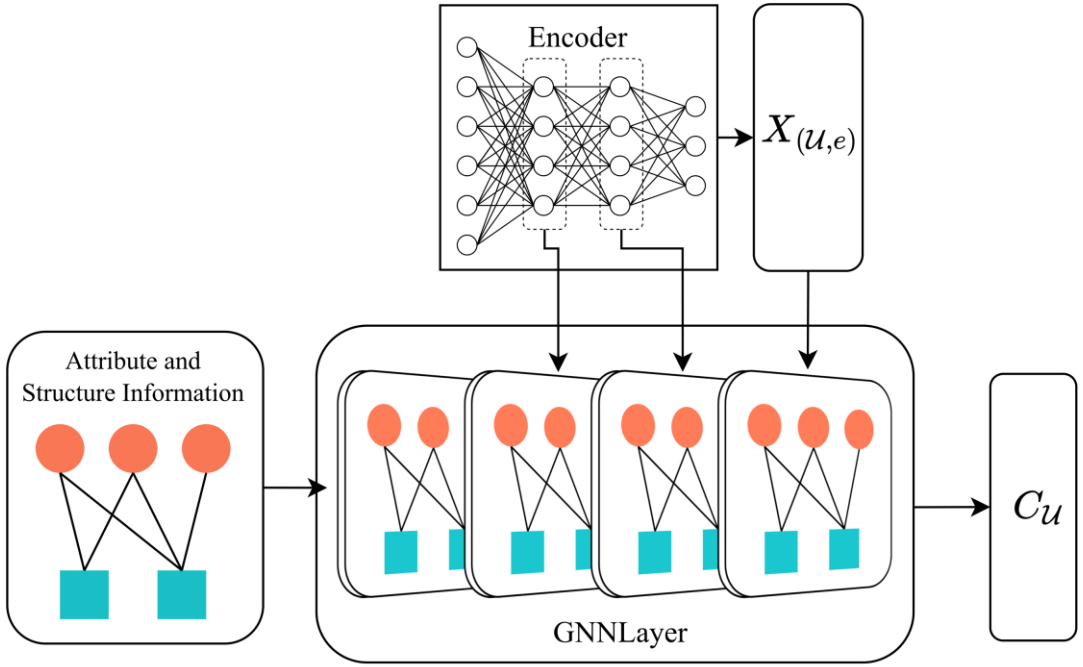
It can achieve self-supervised fraud **community detection** using a loss function measures with Student's t-distribution kernel.



GFDN

Fraudster Community Detection

BDCN - GNN:
GNN in BDCN can aggregate on attribute bipartite graph and preserve the attribute information and structural information of the graph. The output of encoding layer will be used.

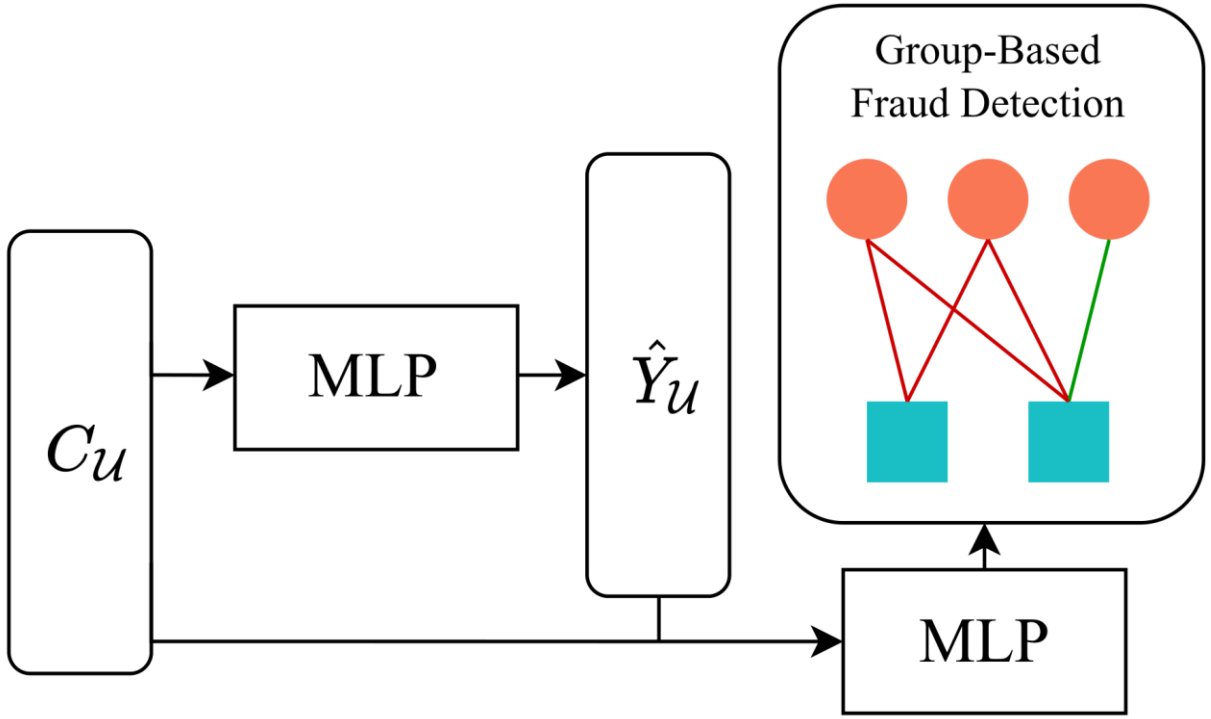


GFDN

Training Objective

"Ride Item's Coattails" Attack:
In "Ride Item's Coattails" attack, not all edges related to fraudsters necessarily have attack implications. GFDN will perform **multi-task training** on this issue, predicting both **fraudsters** and **fraudulent attack**.

STARS Attack:
STARS attack detection aims to **detect fraudsters**, in which case GFDN only needs to perform the vertex classification task.



GFDN

Training Objective

The final loss function will be composed of the loss functions of the aforementioned modules, including **autoencoder**, **community prediction**, **fraudster prediction**, and **fraudulent prediction**. The sum of the weights of all parts of them is 1.

$$\mathcal{L} = \omega_{ae}\mathcal{L}_{ae} + \omega_{c}\mathcal{L}_{c} + \omega_{l}\mathcal{L}_{l} + \omega_{e}\mathcal{L}_{e}$$

Autoencoder **Community** **Fraudster** **Fraudulent**

Experiments

Experimental Setup

- **Dataset**
  - 4 real-life datasets.
- **Compared methods**
  - 5 learning-based methods.
  - 2 pattern-based methods.
  - 4 fraud detection methods.
  - A naïve model and four ablated GFDNs
- **Parameter settings**
  - The number of GNN layer: 4.
  - The number of community: 32.
  - Hidden dimension: 128.
  - The selected GNN is GraphSAGE.
- **Implementation**
  - Structure information extraction: C++
  - Other Parts of the Model :Python + Pytorch Geometric.

Table 1: Datasets for "Ride Item's Coattails" Attack Detection

Dataset	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{V}\rvert$	% Fraudulent	% Legitimate
TB	3,085,653	996,090	381,611	0.62%	3.53%
TC	1,050,000	532,345	239,840	2.86%	11.43%

Table 2: Datasets for STARS Attack Detection

Dataset	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{V}\rvert$	% Fraudulent	% Legitimate
Alpha	24,186	3,286	3,754	3.10%	4.20%
OTC	35,592	4,814	5,858	3.70%	2.80%

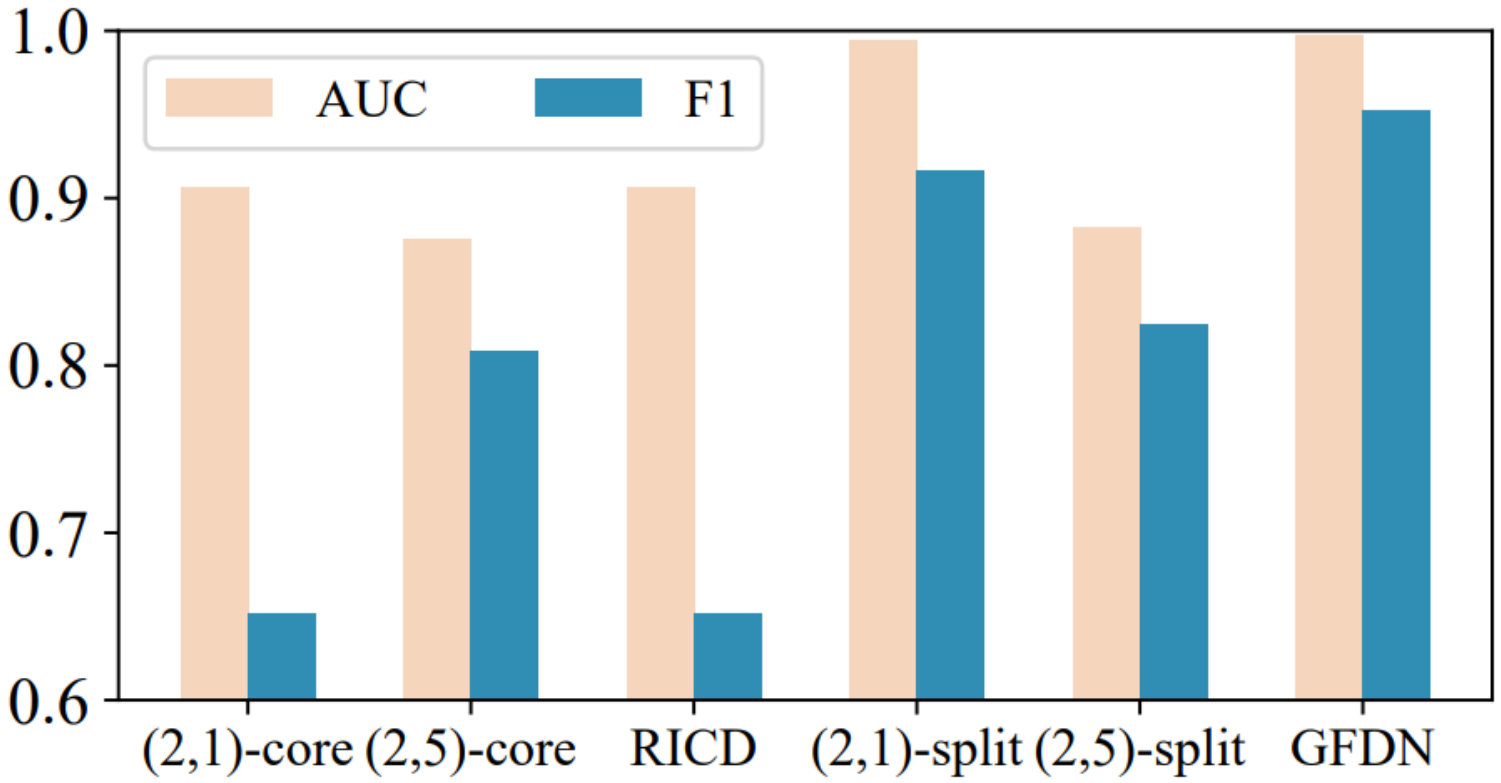
Experiments

Effectiveness Evaluation Results for "Ride Item's Coattails" Detection

	TB Data					TC Data				
	F1	Acc	AUC	Pre	Recall	F1	Acc	AUC	Pre	Recall
LPA	0.2737	0.4627	0.5517	0.1715	0.6785	0.2056	0.4284	0.5276	0.1219	0.6557
SBGNN	0.4789	0.8228	0.7947	0.4279	0.5438	0.3676	0.8074	0.7666	0.2900	0.5018
BiGI	0.5359	0.8540	0.8491	0.5097	0.5649	0.4039	0.8292	0.8044	0.3331	0.5129
SIHG	0.6449	0.8709	0.8692	0.5470	0.7853	0.5947	0.8771	0.8985	0.4735	0.7992
Tianchi	0.6446	0.8752	0.9342	0.5606	0.7581	0.5364	0.8717	0.9107	0.4527	0.6583
RICD	0.6518	0.8405	0.9063	0.4834	**1.0000**	0.4784	0.8482	0.7474	0.3906	0.6171
(α, β)-core	0.8081	0.9449	0.8757	0.8417	0.7770	0.6348	0.8907	0.8696	0.5093	0.8423
FRAUDAR	0.2580	0.1481	0.4963	0.1483	0.9927	0.2020	0.1124	0.4981	0.1124	**0.9961**
CF1	0.2407	0.7698	0.5532	0.2371	0.2445	0.1620	0.7981	0.5253	0.1523	0.1731
CF2	0.4675	0.7603	0.7376	0.3497	0.7052	0.3588	0.6837	0.7277	0.2326	0.7844
Naive	0.8109	0.9473	0.9844	0.8736	0.7565	0.6397	0.9090	0.9516	**0.7816**	0.5414
GFDN-S	0.6867	0.9202	0.9653	0.8284	0.5864	0.6122	0.8783	0.9342	0.4780	0.8514
GFDN-F	0.9212	0.9754	0.9886	0.8821	0.9639	0.6401	0.8976	0.9287	0.5302	0.8076
GFDN-L	0.9398	0.9813	0.9964	0.9050	0.9775	0.7015	0.9192	0.9654	0.6014	0.8417
GFDN-C	0.9423	0.9821	0.9967	0.9086	0.9785	0.7048	0.9226	0.9646	0.6181	0.8198
GFDN	**0.9522**	**0.9853**	**0.9974**	**0.9254**	0.9806	**0.7226**	**0.9242**	**0.9713**	0.6154	0.8752

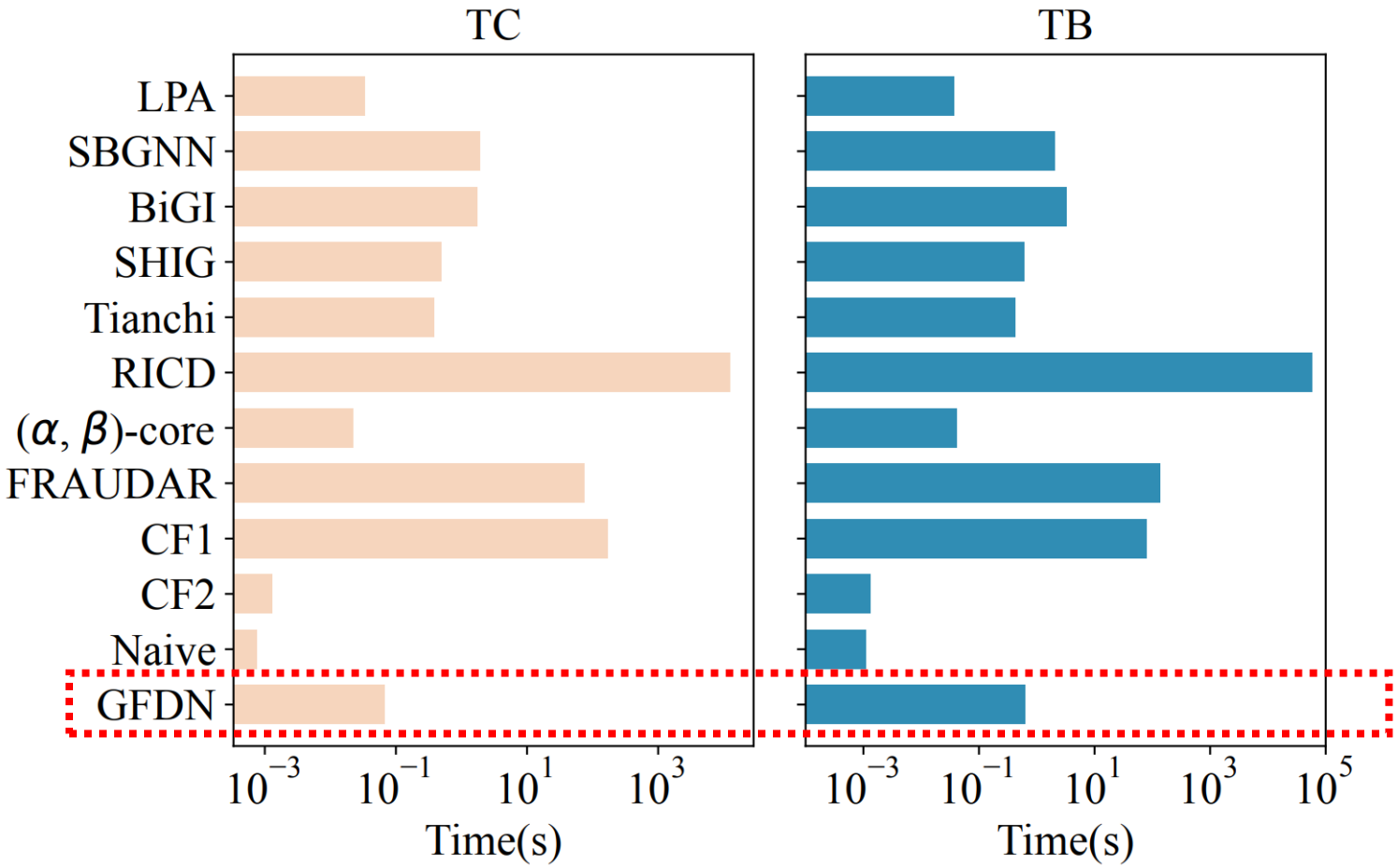
Experiments

Comparison with Pattern-based Algorithms



Experiments

Query Time Evaluation of “Ride Item’s Coattails” Detection



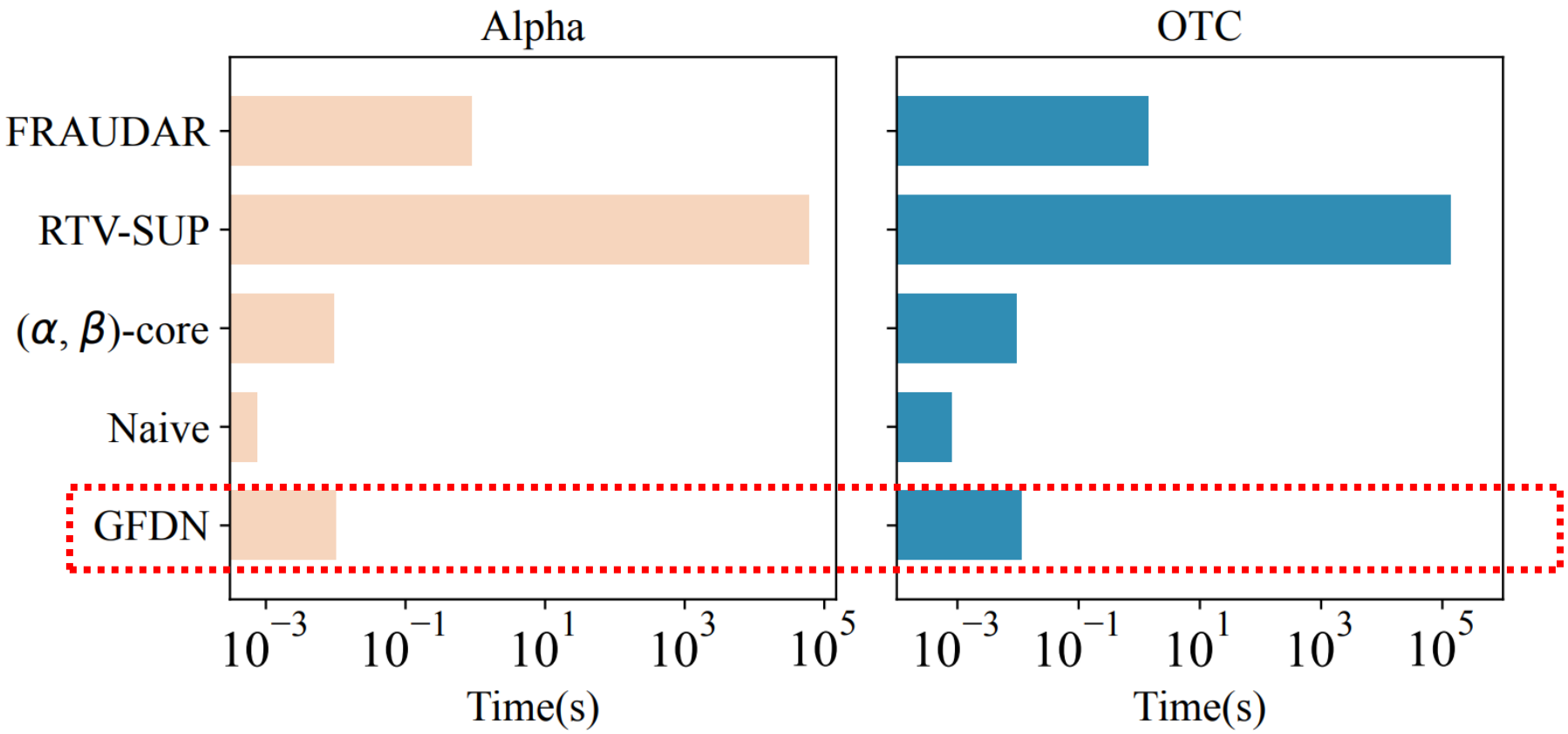
Experiments

Effectiveness Evaluation Results for STARS Detection

	Alpha					OTC				
	F1	Acc	AUC	Pre	Recall	F1	Acc	AUC	Pre	Recall
FRAUDAR	0.3800	0.2626	0.5236	0.2346	**1.0000**	0.3780	0.2547	0.5183	0.2330	**1.0000**
RTV-SUP	0.8652	**0.9452**	0.8859	**0.9747**	0.7778	0.7010	0.8082	0.8736	0.5417	0.9931
(α, β)-core	0.7857	0.8767	0.9204	0.6471	**1.0000**	0.7784	0.8711	0.9167	0.6372	**1.0000**
Naive	0.8089	0.9018	0.9789	0.7222	0.9192	0.7937	0.8978	0.9508	0.7310	0.8681
GFDN	**0.8919**	**0.9452**	**0.9913**	0.8049	**1.0000**	**0.9231**	**0.9623**	**0.9746**	**0.8571**	**1.0000**

Experiments

Efficiency Evaluation Results for STARS Detection



Thank you!

Q&A

hanchen.wang@uts.edu.au

https://hanchen-wang.com/