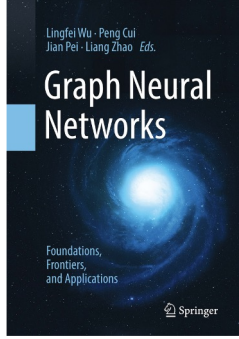


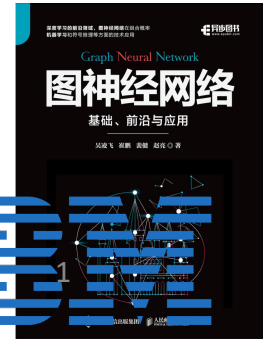
Graph Neural Networks: Foundations, Frontiers, Applications

Lingfei Wu, Peng Cui, Jian Pei,

Liang Zhao and Xiaojie Guo

AAAI 2023 Tutorial

2023-02-08



Outline

GNNs Foundations

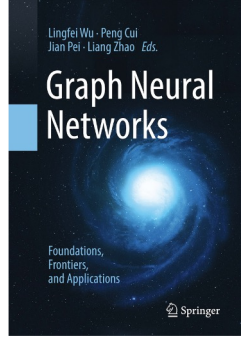
- Time History of GNNs
- GNNs: Foundations and Models
- GNNs: Theory, Scalability, Interpretability

GNNs Frontiers

- Graph Generation and Transformation
- Dynamic Graph Neural Networks
- Graph Matching
- Graph Structure Learning

GNNs Applications

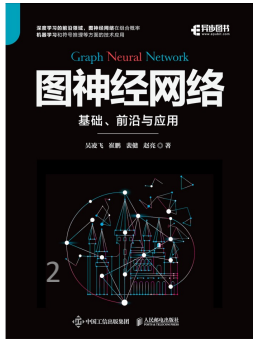
- GNNs in Recommendation
- GNNs in Natural Language Processing
- GNNs in Program Analysis
- GNNs in Protein Modeling

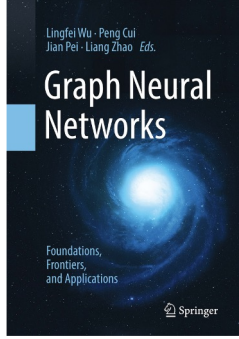


GNN book website：
https://graph-neural-networks.github.io/index.html

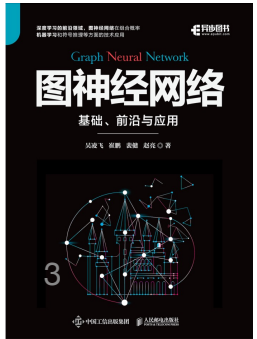
Amazon：
https://www.amazon.com/Graph-Neural-Networks-Foundations-Applications/dp/9811660530

JD.com (京东商城)：
https://item.jd.com/13536841.html





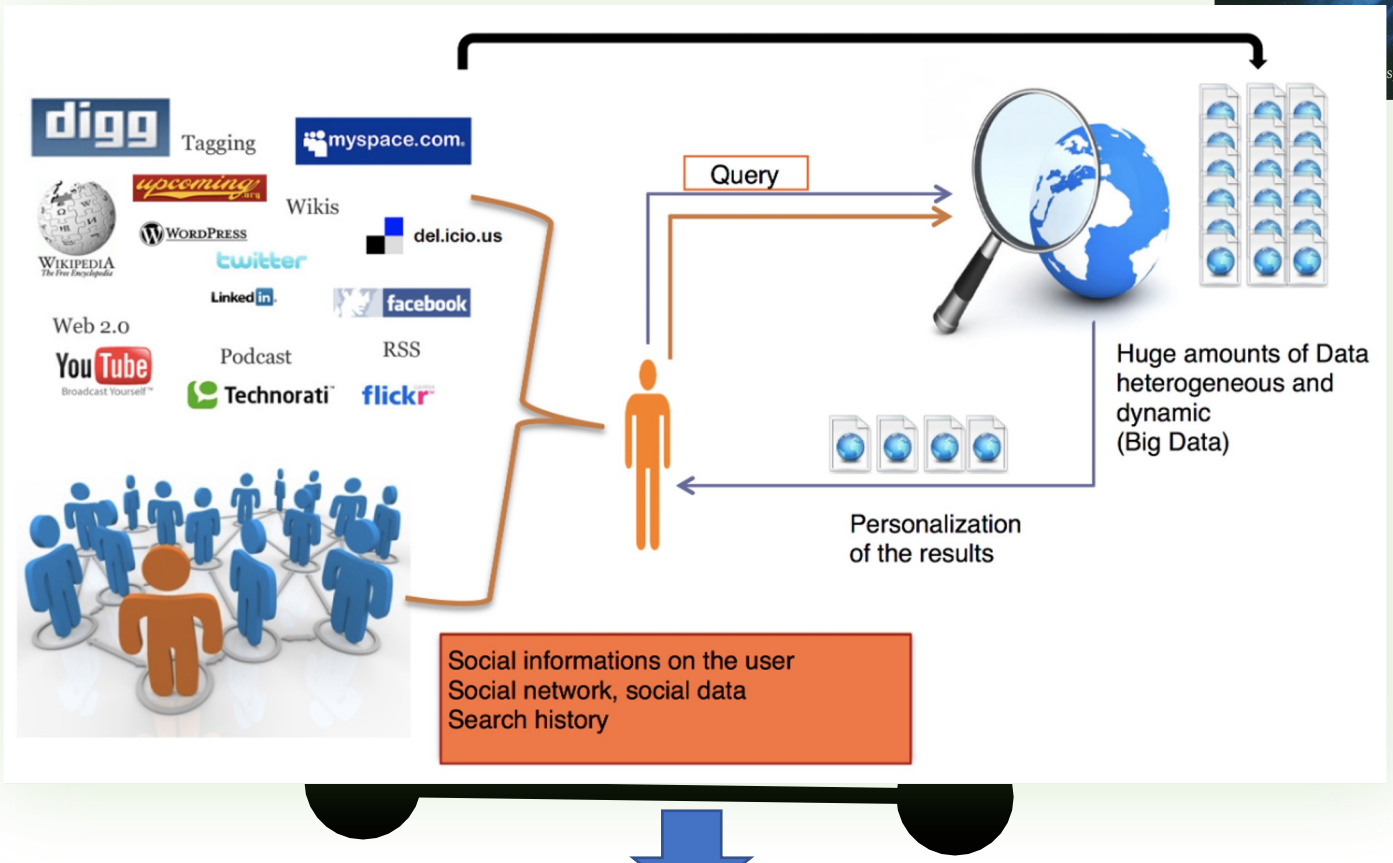
GNNs
Introduction



Why graphs?

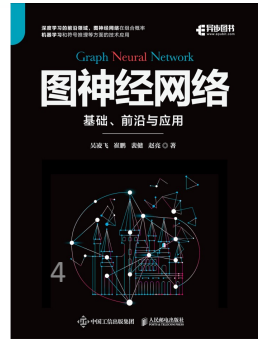
- Graphs are a general language for describing and modeling complex systems

Source from Laboratoire Hubert Curien - Université Jean Monnet

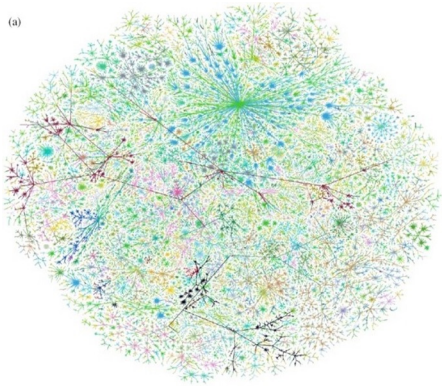


Graph Topology + Node Attributes
+ Node Types

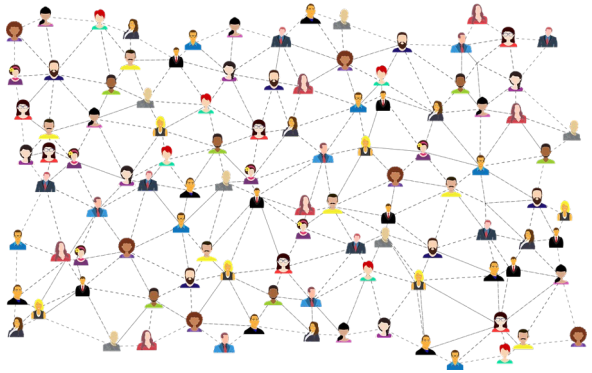
Graph!



Graph-structured data are ubiquitous

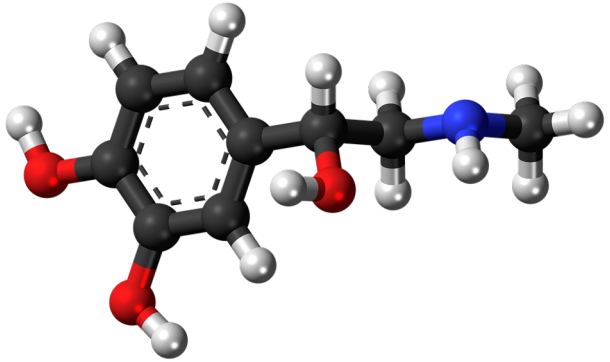


Internet

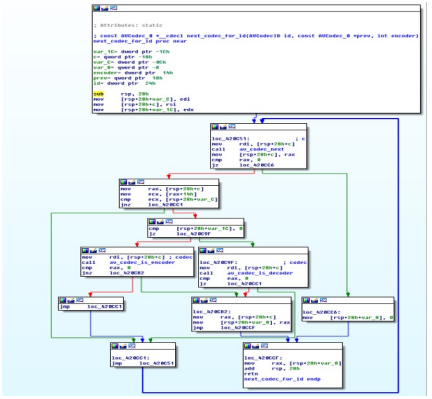


Social networks

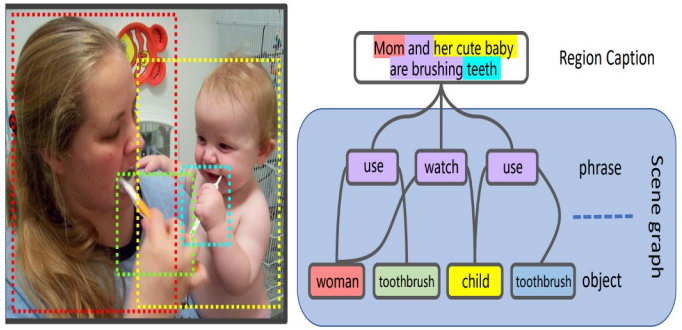
Information retrieval



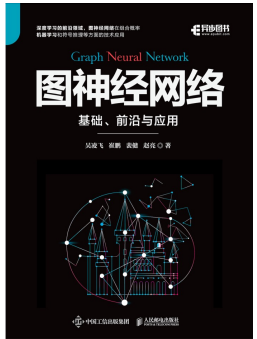
Biomedical graphs

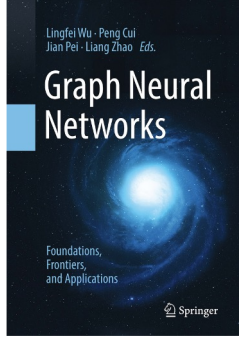


Program graphs

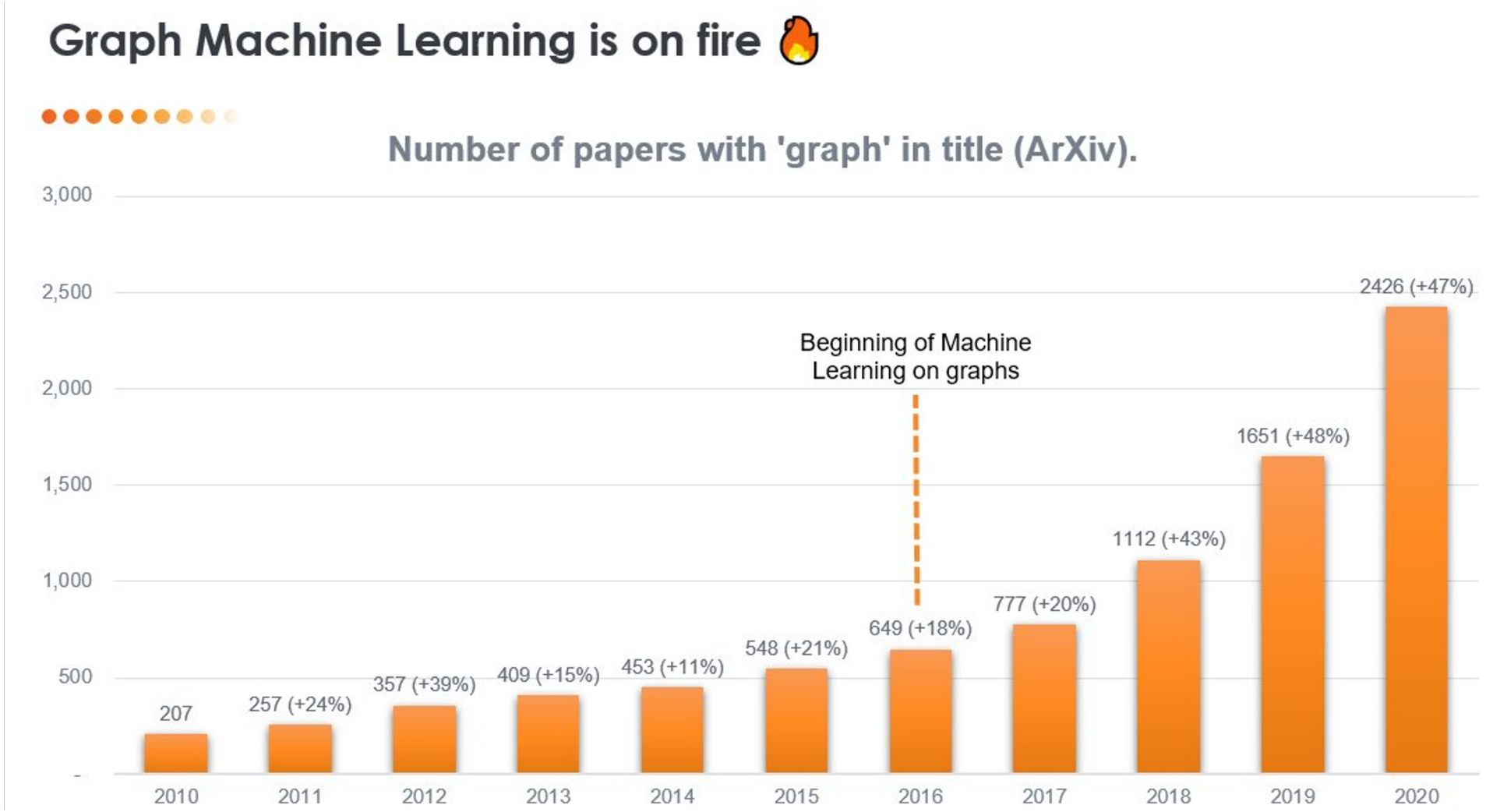


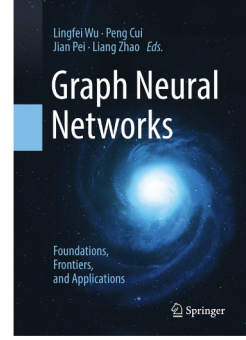
Scene graphs

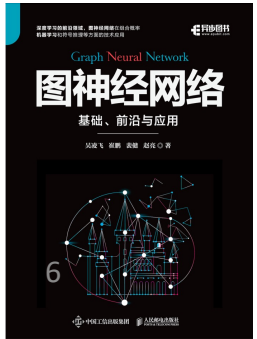




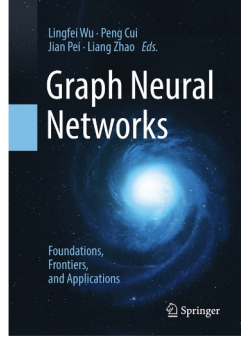
Graph Machine Learning: Recent Trending







GNNs: A Brief History



Year 2009

First GNNs Paper (Scarselli et al., 2009)

Year 2016, 2017

First GRU-based Modern GNNs paper: GGNN

First graph convolution-based paper GCN, Start a new era of GNNs history

Year 2017-2023

A series of Graph Convolution (GCN), Message Passage (MPNN, GraphSage, GIN), Attention-based (GAT), Unsupervised GNNs (Graph-Autoencoder, graph-infomax)

Many new GNNs fast developed!

Year 2019-2023

GNNs–based applications, such as search, Recommendation, drug discovery, NLP, intelligent Transport…

Many open-source Githubs like DGL, Pytorch Geometric, Graph4NLP, TorchDrug…

Year 2021

3 GNN books released simultaneously:

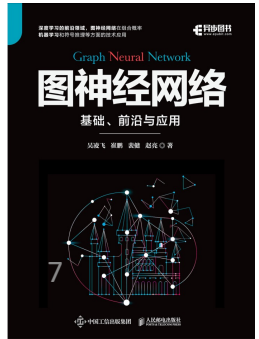
1) Prof. Liu (Tsinghua) et al.: "Introduction to Graph Neural Networks"

2) Prof. Tang (MSU) et al.: "Deep learning on graphs"

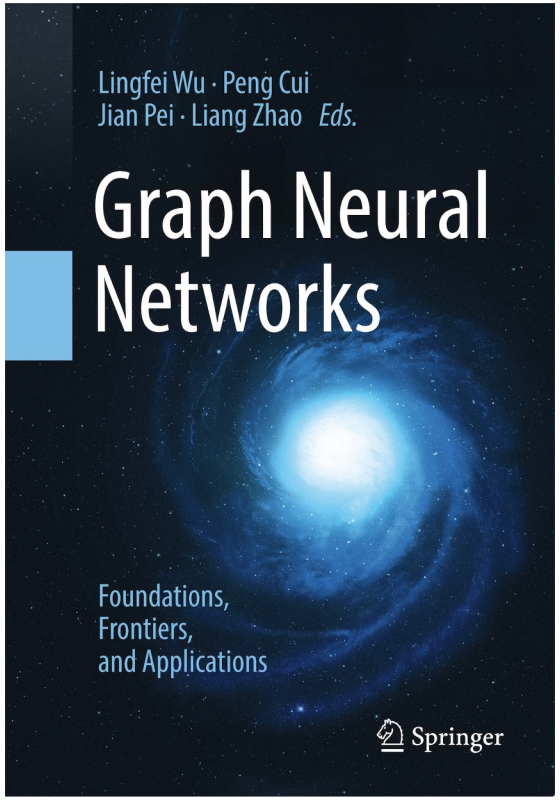
3)Prof. Hamilton (McGill): Graph representation learning

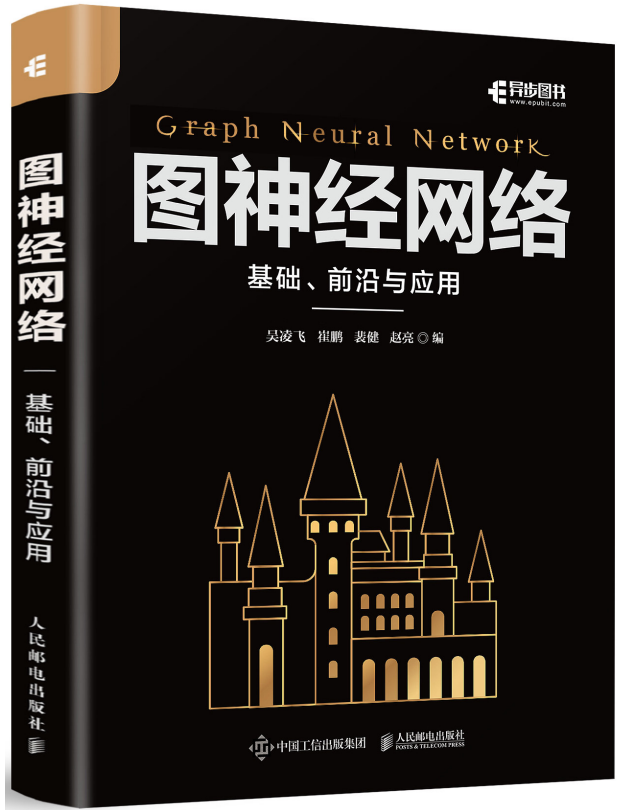
Year 2022

Most comprehensive GNN book: "Graph Neural Networks: Foundations, Frontiers, Applications" by Springer and Posts & Telecom Press



Graph Neural Networks Book 2022

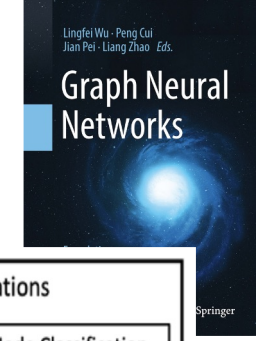


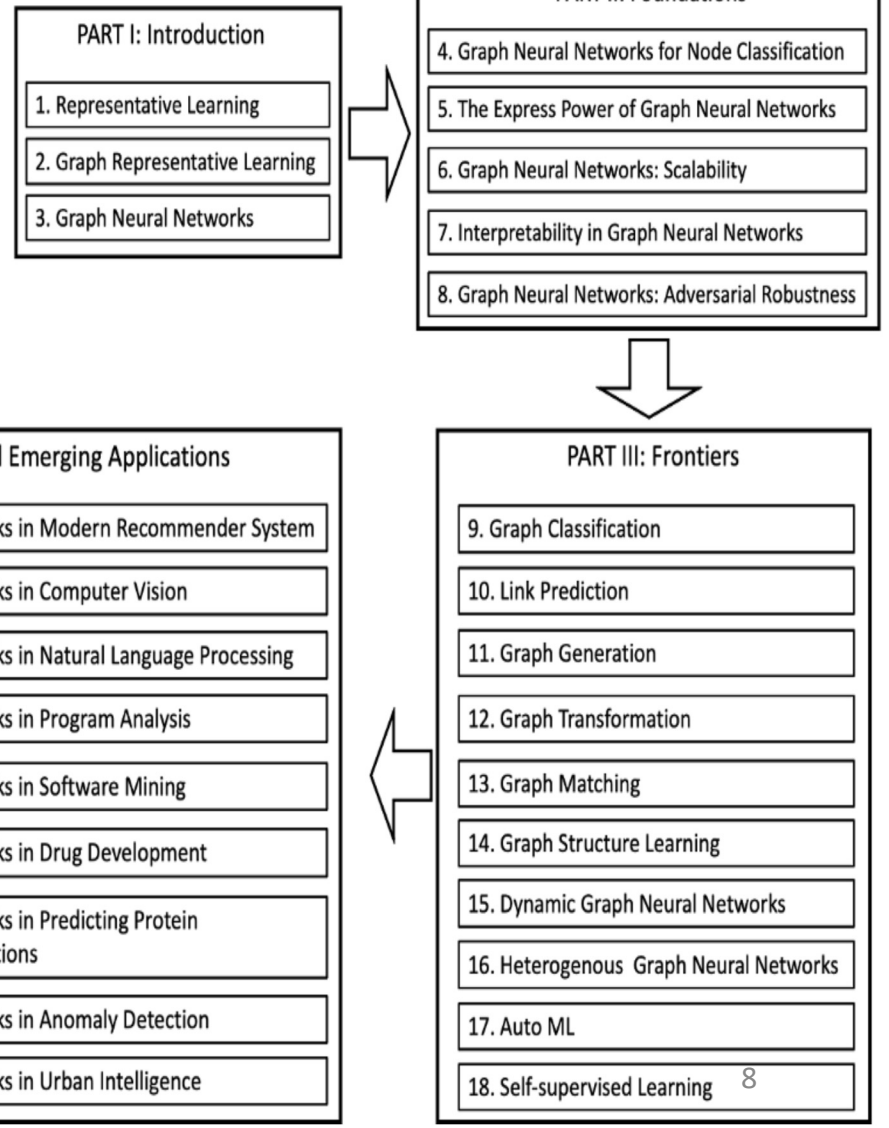


Free download:
https://graph-neural-networks.github.io/index.html

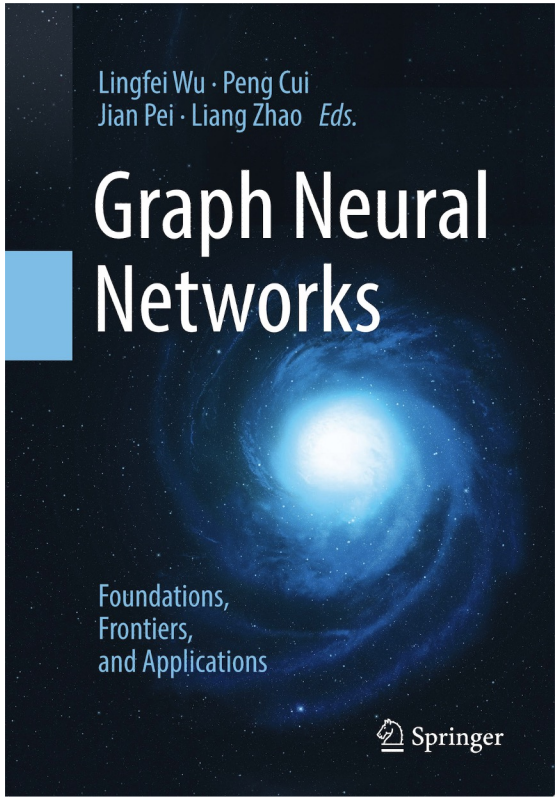
The English version of the book is available on Amazon!

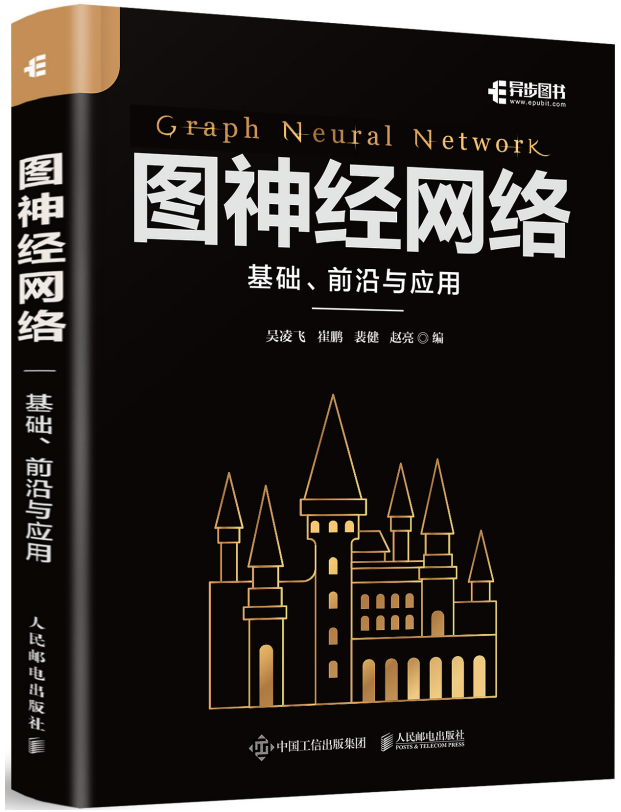
The Chinese version of the book (图神经网络中文城堡书) is available on JD.com!





Graph Neural Networks Book 2022





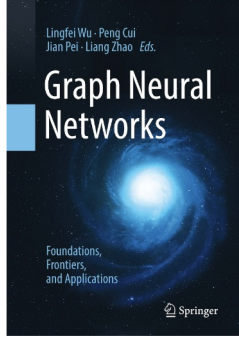
Free download:
https://graph-neural-networks.github.io/index.html

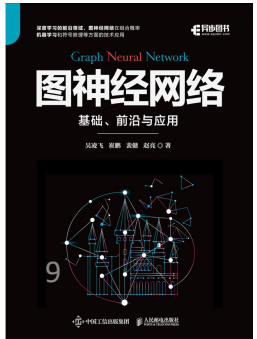
The English version of the book is available on Amazon!

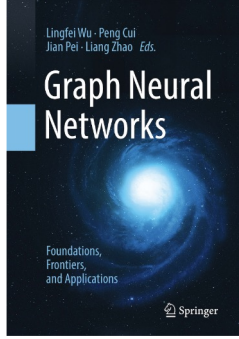
The Chinese version of the book (图神经网络中文城堡书) is available on JD.com!

Chapter contributors:

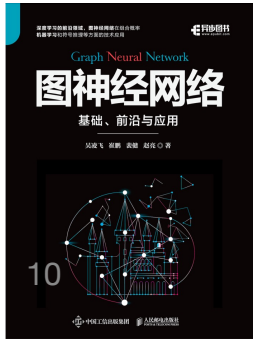
Miltiadis Allamanis、Yu Chen、Yunfei Chu、Tyler Derr、Keyu Duan、Qizhang Feng、Stephan Günnemann、Xiaojie Guo、Yu Hou、Xia Hu、Junzhou Huang、Shouling Ji、Wei Jin、Anowarul Kabir、Seyed Mehran Kazemi、Jure Leskovec、Juncheng Li、Jiacheng Li、Pan Li、Yanhua Li、Renjie Liao、Xiang Ling、Bang Liu、Ninghao Liu、Zirui Liu、Hehuan Ma、Collin McMillan、Christopher Morris、Zongshen Mu、Menghai Pan、Yu Rong、Amarda Shehu、Kai Shen、Chuan Shi、Le Song、Chang Su、Jian Tang、Siliang Tang、Fei Wang、Shen Wang、Shiyu Wang、Xiao Wang、Yu Wang、Chunming Wu、Hongxia Yang、Jiangchao Yao、Philip S. Yu、Muhan Zhang、Wenqiao Zhang、Chang Zhou、Kaixiong Zhou、Xun Zhou.

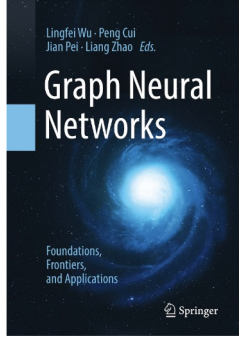




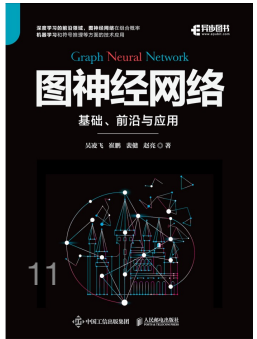


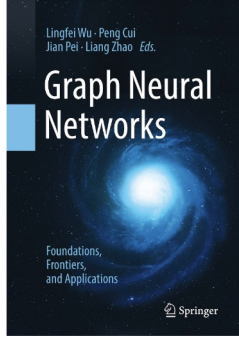
GNNs
Foundations





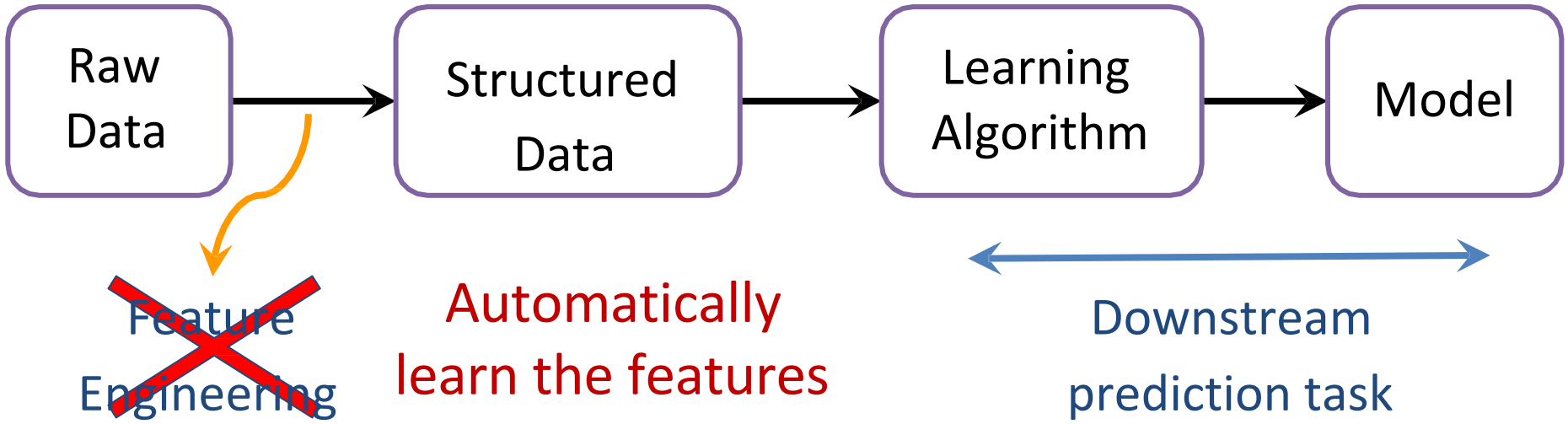
GNNs for Node Classification (Chapter 4)

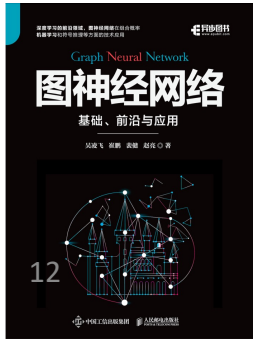


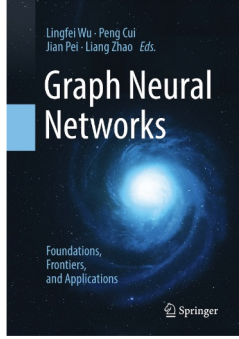


Machine Learning Lifecycle

- (Supervised) Machine Learning Lifecycle: feature learning is the key

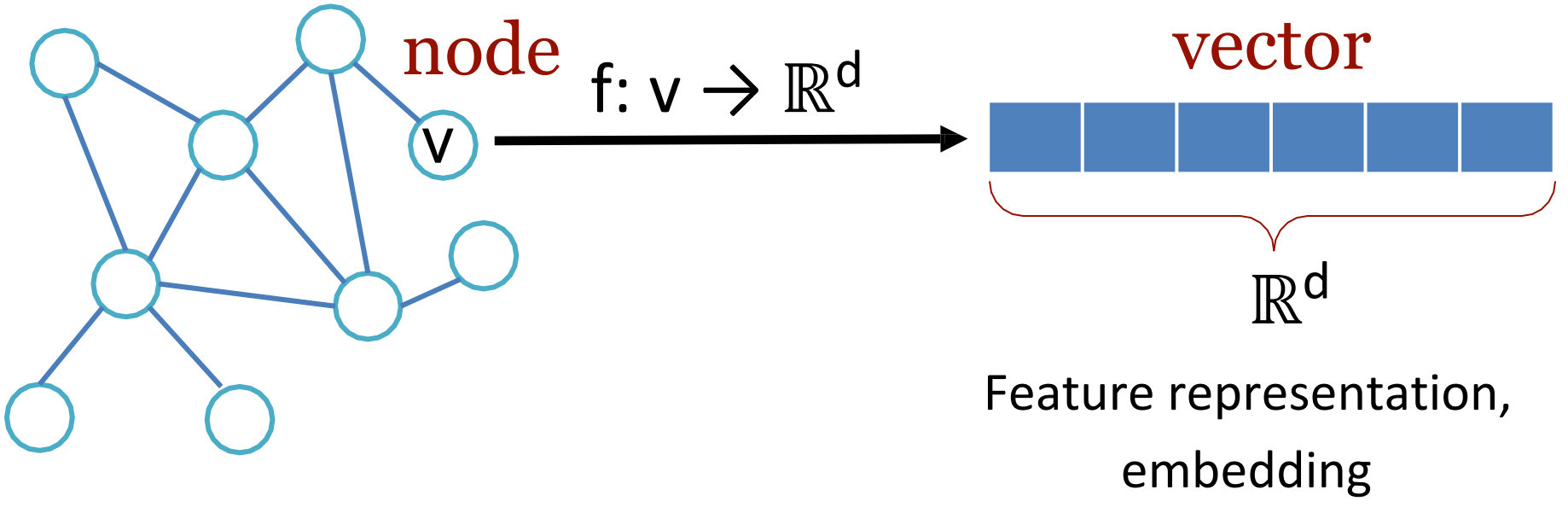


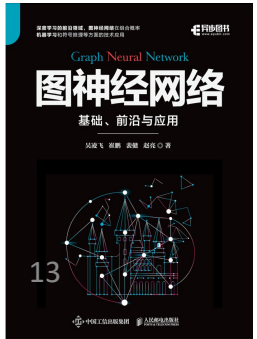


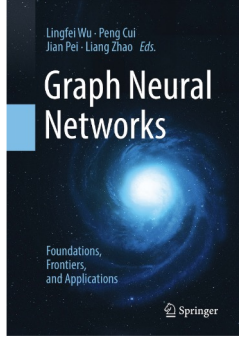


Feature Learning in Graphs

- Our Goal: Design efficient task-independent/ task-dependent feature learning for machine learning in graphs!







Graph Neural Networks: Foundations

- Learning node embeddings:

A graph filter adjacency matrix

$$\mathbf{h}_i^{(l)} = f_{\text{filter}}(A, \mathbf{H}^{(l-1)})$$

Updated node embeddings Input node embeddings

$f_{\text{filter}}(\cdot,\cdot)$

- Spectral-based
- Spatial-based
- Attention-based
- Recurrent-based

- Learning graph-level embeddings:

$$A', \mathbf{H}' = f_{\text{pool}}(A, \mathbf{H})$$

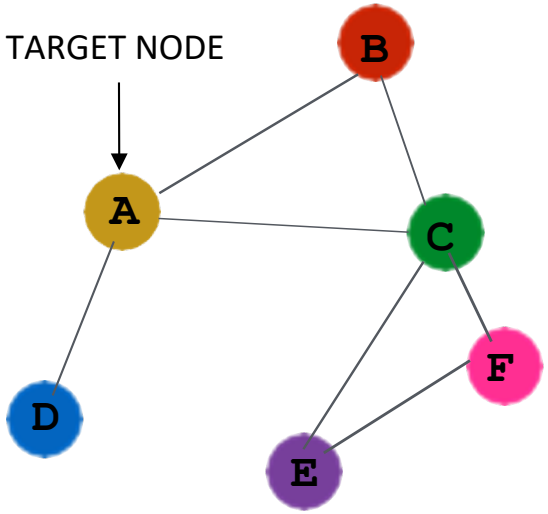
A small graph w/ fewer nodes New node embeddings Input graph Input node embeddings

$f_{\text{pool}}(\cdot,\cdot)$

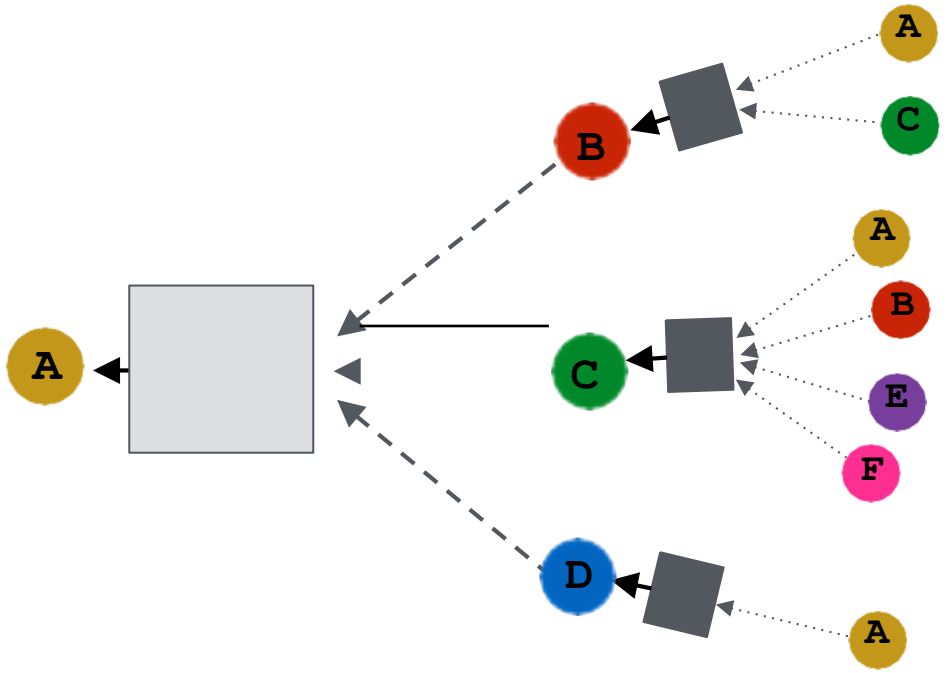
- Flat Graph Pooling (i.e. Max, Ave, Min)
- Hierarchical Graph Pooling (i.e.

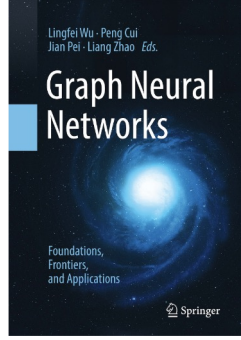
Graph Neural Networks: Basic Model

- Key idea: Generate node embeddings based on local neighborhoods.



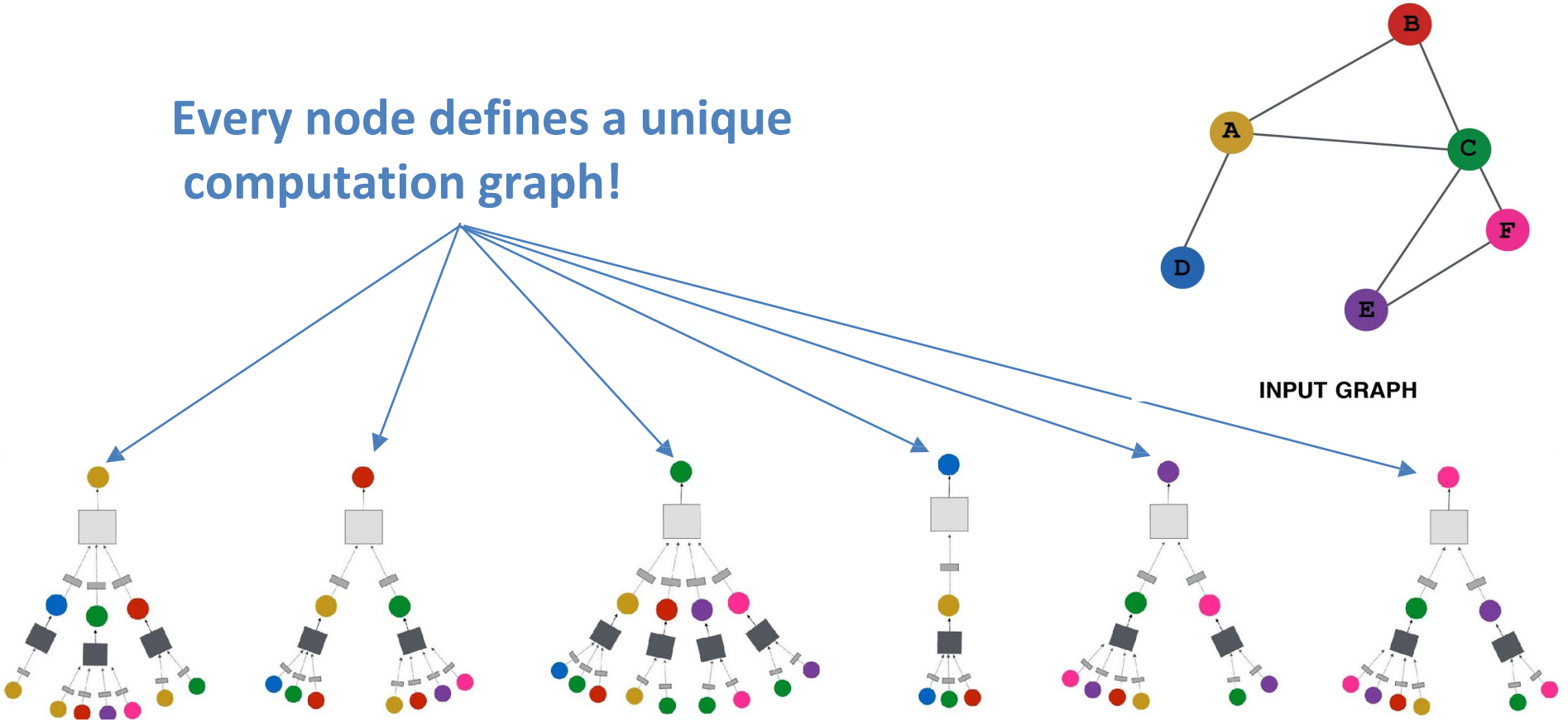
INPUT GRAPH

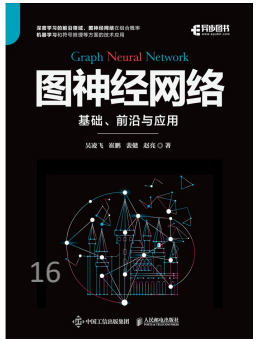




Neighborhood Aggregation

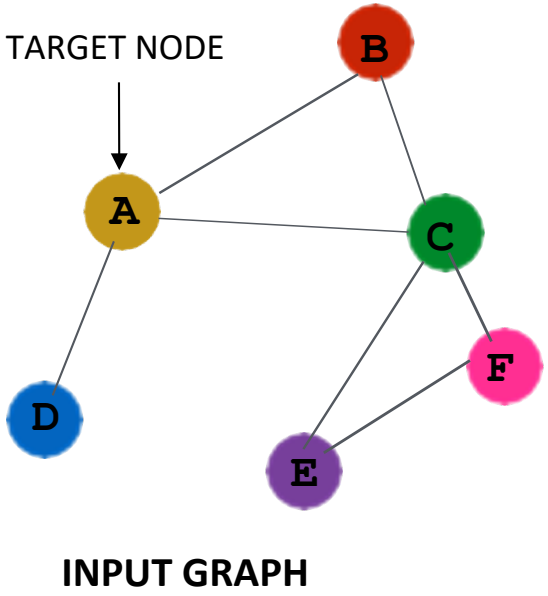
- Intuition: Network neighborhood defines a computation graph

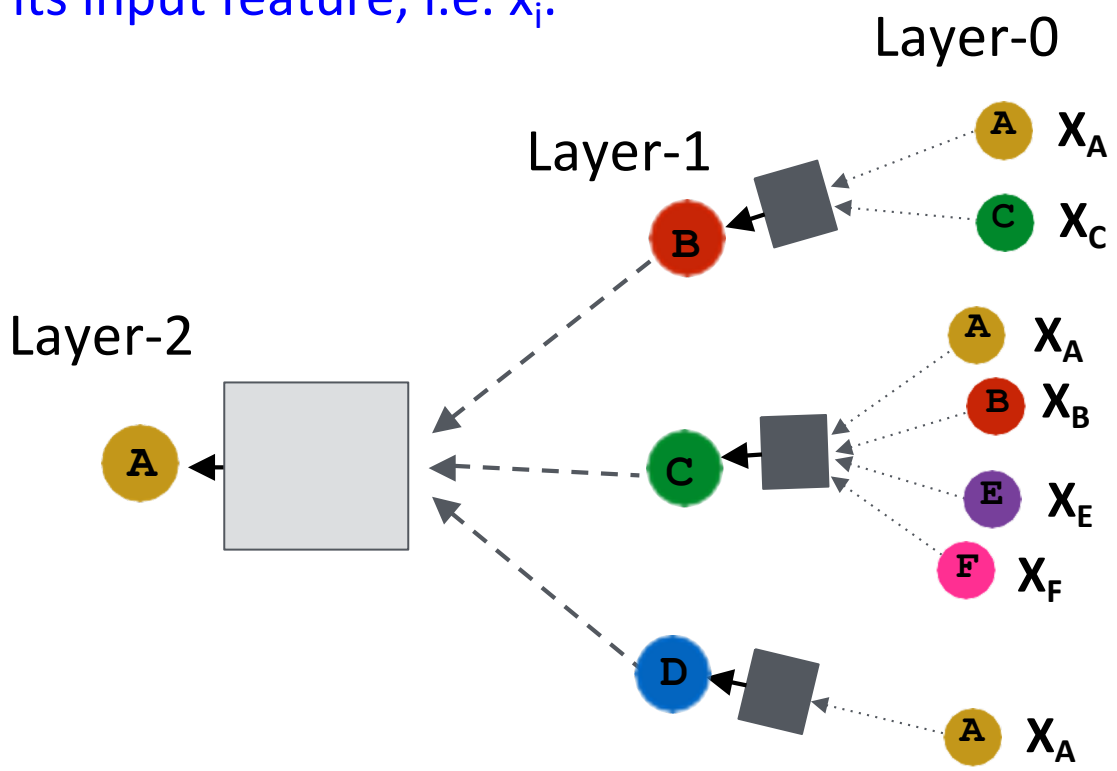




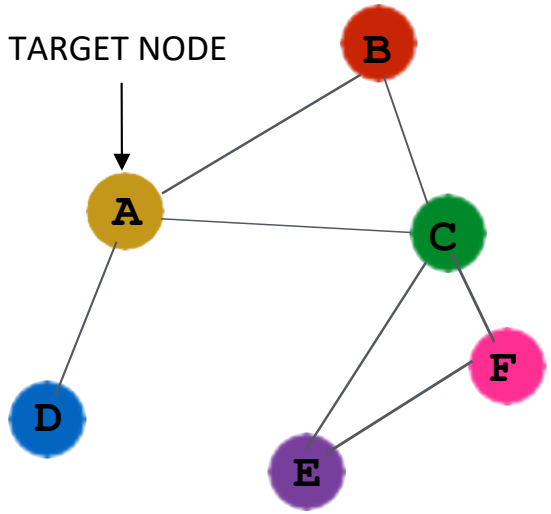
Neighborhood Aggregation

- Nodes have embeddings at each layer.
- Model can be arbitrary depth.
- "layer-0" embedding of node i is its input feature, i.e. x_i.

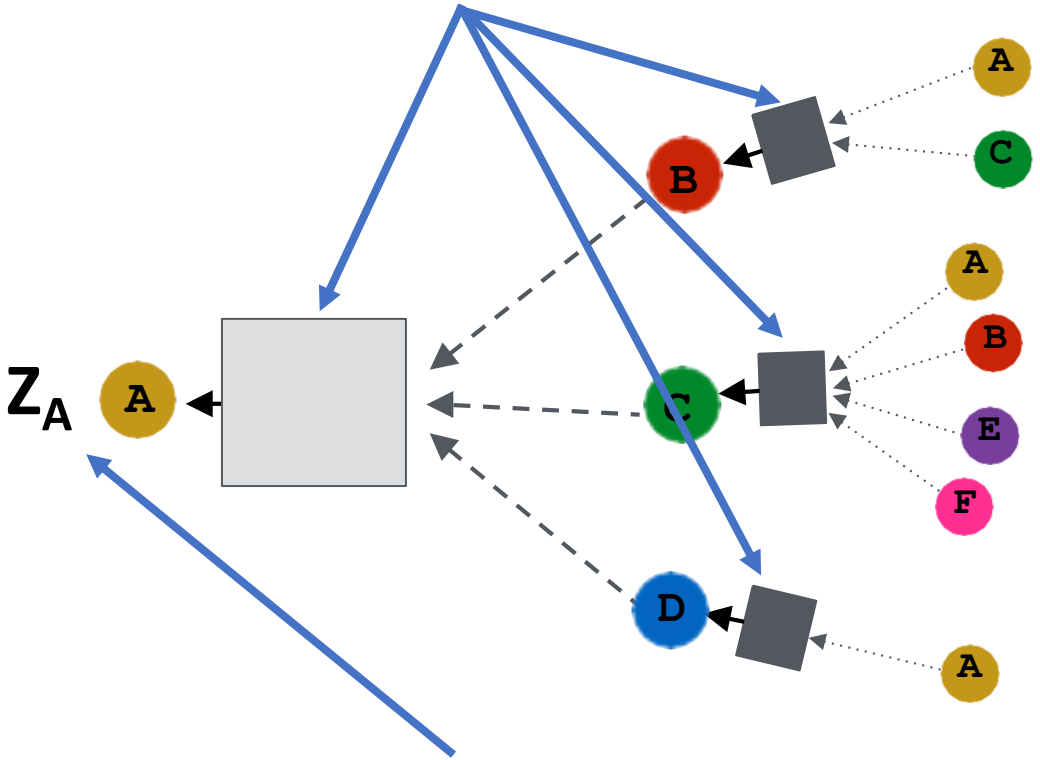




Overview of GNN Model

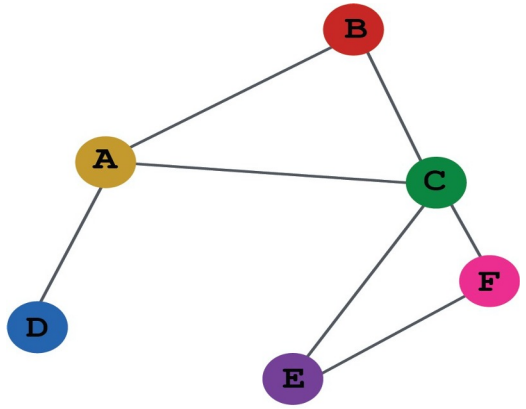


1) Define a neighborhood aggregation function



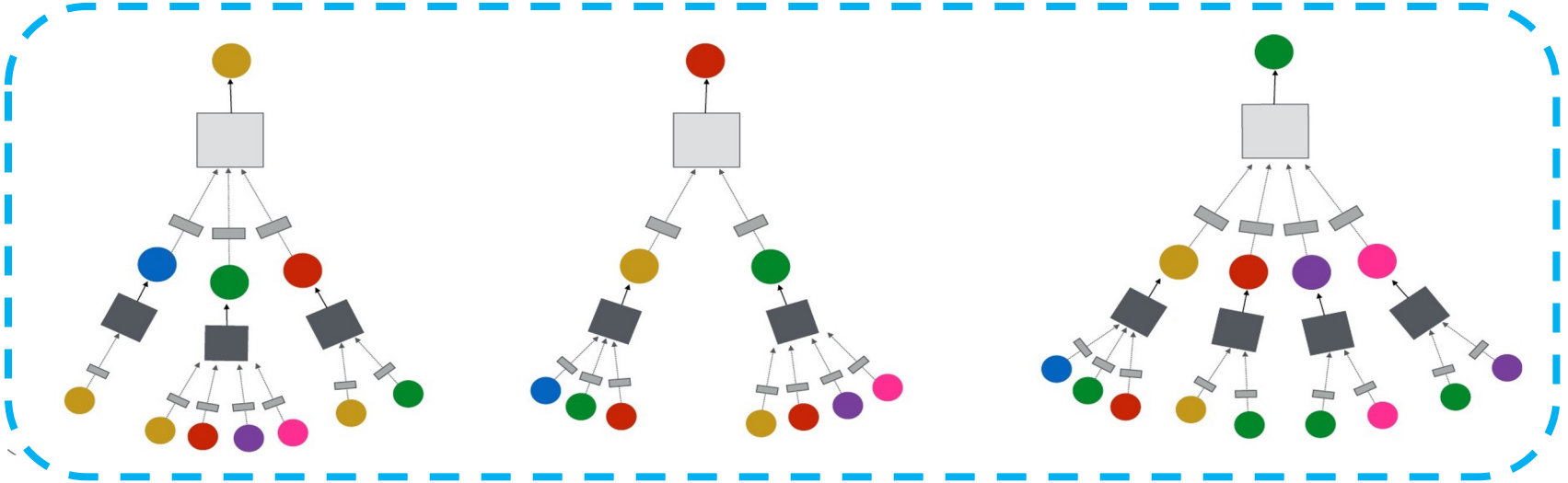
2) Define a loss function on the embeddings, $\mathbf{L(z_v)}$

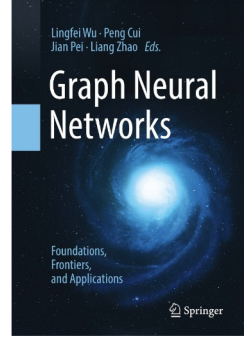
Overview of GNN Model

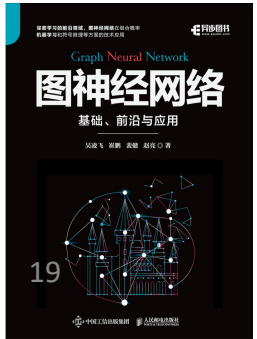


INPUT GRAPH

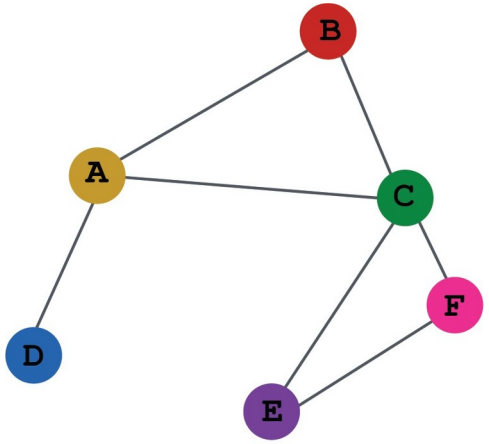
3) Train on a set of nodes, i.e., a batch of computation graphs







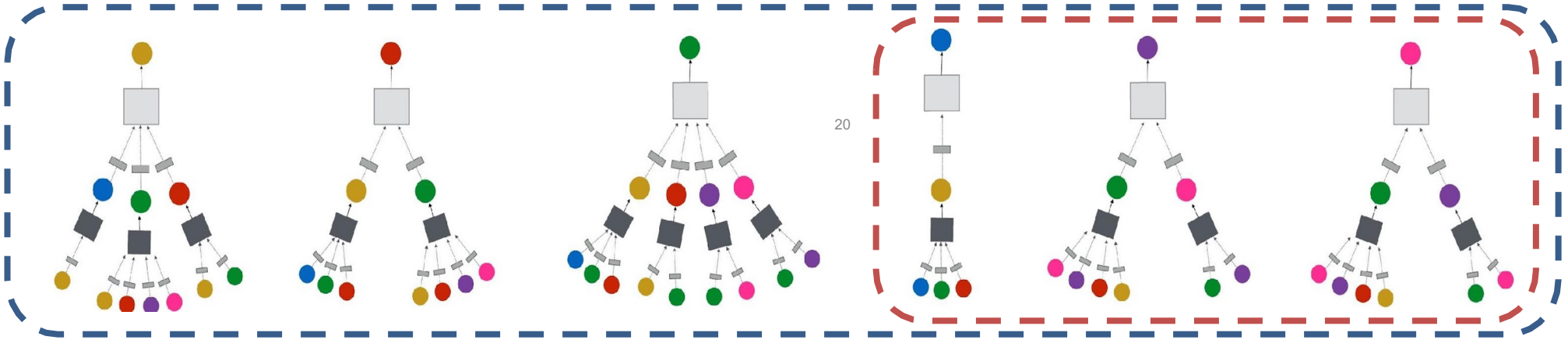
Overview of GNN Model

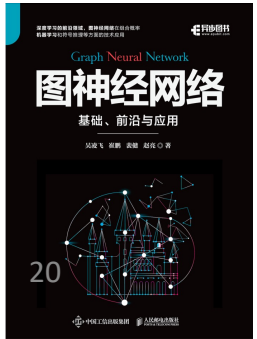


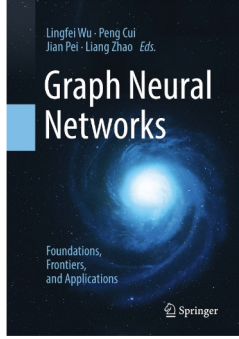
INPUT GRAPH

4) Generate embeddings for nodes as needed

Even for nodes we never trained on!



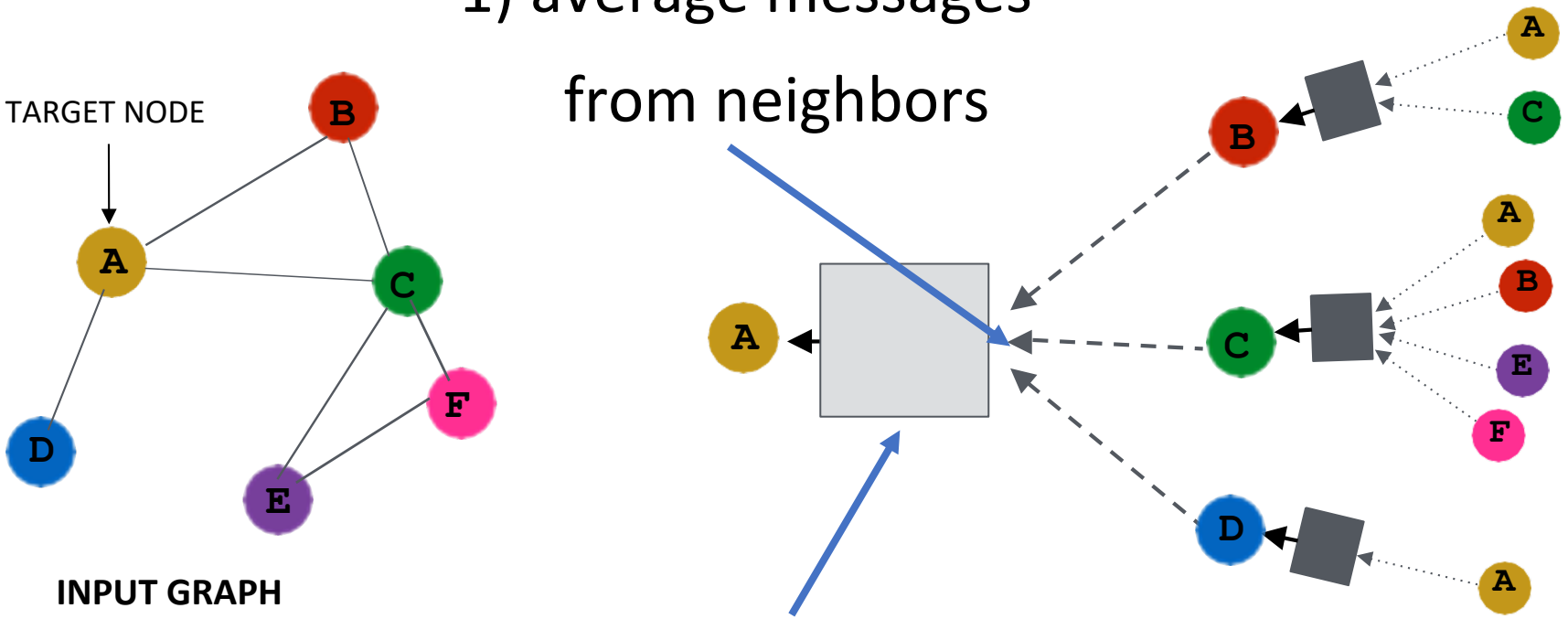




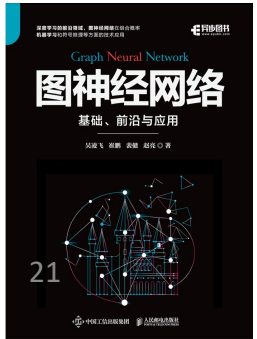
GNN Model: A Case Study

- Basic approach: Average neighbor information and apply a neural network

1) average messages from neighbors

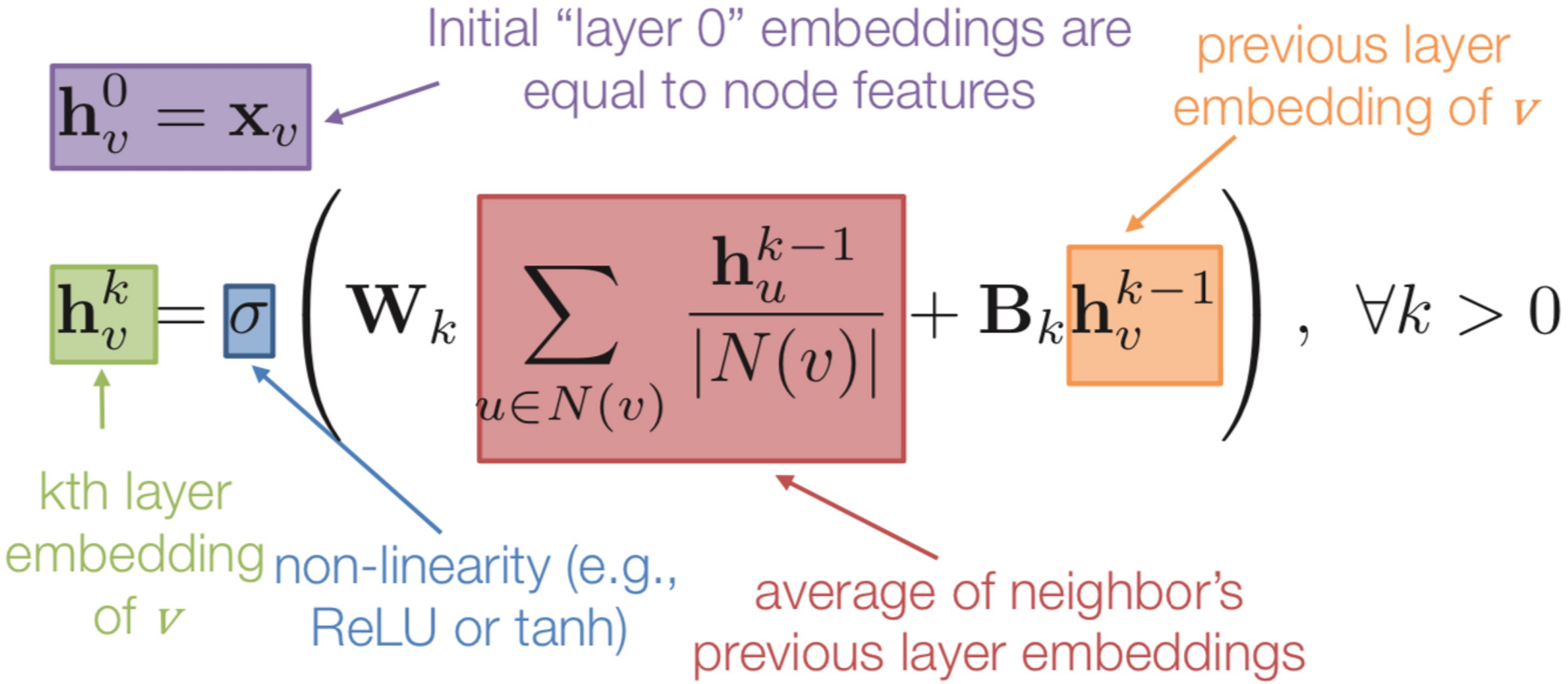


2) apply neural network



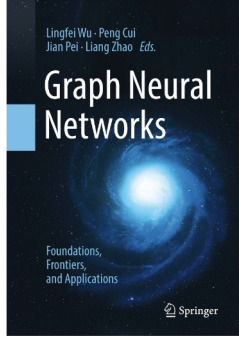
GNN Model: A Case Study

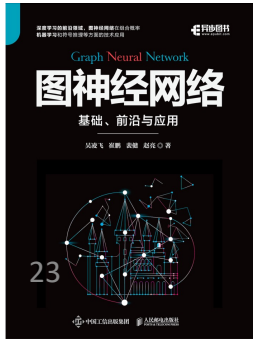
- Basic approach: Average neighbor information and apply a neural network.

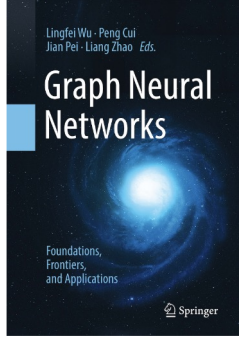


GNN Model: Quick Summary

- Key idea: generate node embeddings by aggregating neighborhood information.
  - Allows for parameter sharing in the encoder
  - Allows for inductive learning

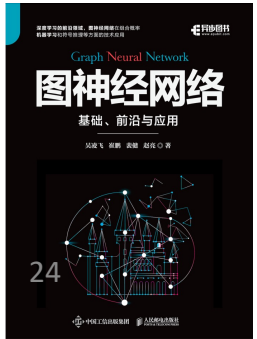


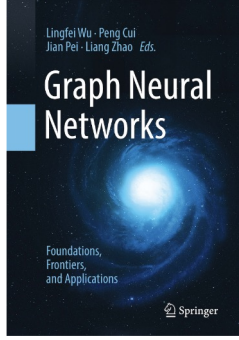




Graph Neural Networks: Classic Models

- Supervised Graph Neural Networks
  - Spectral-based Graph Filters
    - GCN (Kipf & Welling, ICLR 2017), Chebyshev-GNN (Defferrard et al. NIPS 2016)
  - Spatial-based Graph Filters
    - MPNN (Gilmer et al. ICML 2017), GraphSage (Hamilton et al. NIPS 2017)
    - GIN (Xu et al. ICLR 2019)
  - Attention-based Graph Filters
    - GAT (Velickovic et al. ICLR 2018)
  - Recurrent-based Graph Filters
    - GGNN (Li et al. ICLR 2016)
- Unsupervised Graph Neural Networks
  - Variational graph auto-encoders (Kipf and Welling, 2016)
  - Deep graph infomax (Velickovic et al, 2019)





Graph Convolution Networks (GCN)

Key idea: spectral convolution on graphs

Eigen-decomposition is expensive

$$f_{\text{filter}} * \mathbf{x}_i = \mathbf{U} f(\Lambda) \mathbf{U}^T \mathbf{x}_i$$

Chebyshev polynomials accelerates but still not powerful

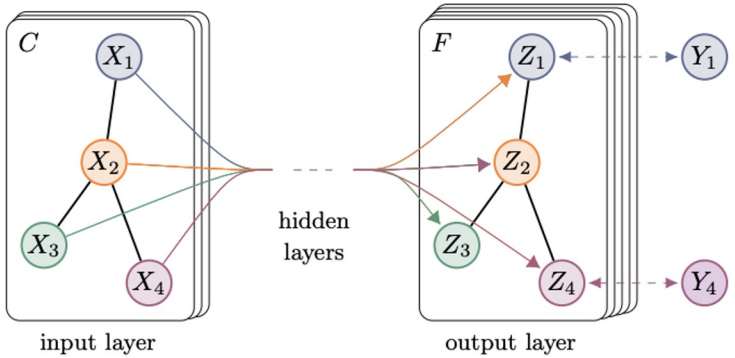
$$f'_{\text{filter}} * \mathbf{x}_i \approx \sum_{p=0}^{P} \theta'_p \mathbf{T}_p(\tilde{\mathbf{L}}) \mathbf{x}_i$$

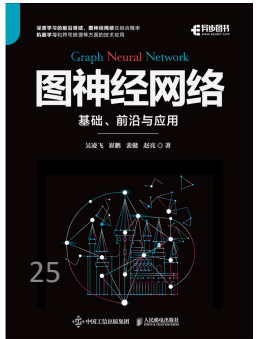
First-order approximation fast and powerful

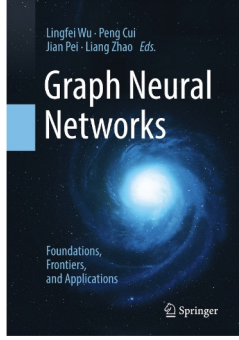
$$f_{\text{filter}} * \mathbf{h}_i^{(l)} \approx \theta(I_n + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}) \mathbf{h}_i^{(l)}$$

Renormalization trick stabilizes the numerical computation

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



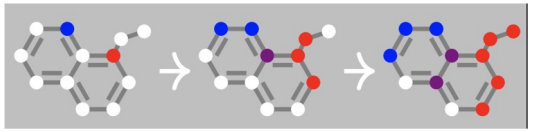




Message Passing Neural Network (MPNN)

Key idea: graph convolutions as a message passing process

MPNN: $$\mathbf{h}_i^{(l)} = f_{\text{filter}}(A, \mathbf{H}^{(l-1)}) = f_U(\mathbf{h}_i^{(l-1)}, \sum_{v_j \in N(v_i)} f_M(\mathbf{h}_i^{(l-1)}, \mathbf{h}_j^{(l-1)}, \mathbf{e}_{i,j}))$$

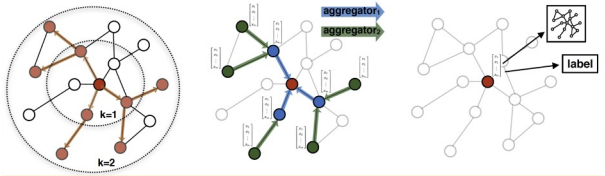


expensive if the number of nodes are large

Update and aggregation functions

Node and edge embeddings

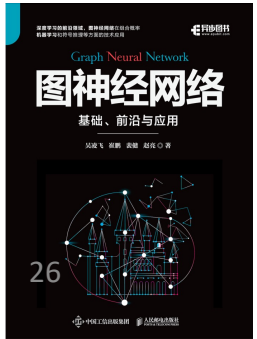
GraphSage: $$f_{\text{filter}}(A, \mathbf{H}^{(l-1)}) = \sigma(\mathbf{W}^{(l)} \cdot f_M(\mathbf{h}_i^{(l-1)}, \{\mathbf{h}_j^{(l-1)}, \forall v_j \in N(v_i)\}))$$

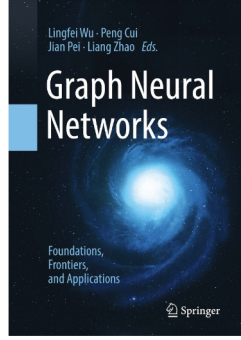


sampling to obtain a fixed number of neighbors

Aggregation functions

Node embeddings





Graph Attention Network (GAT)

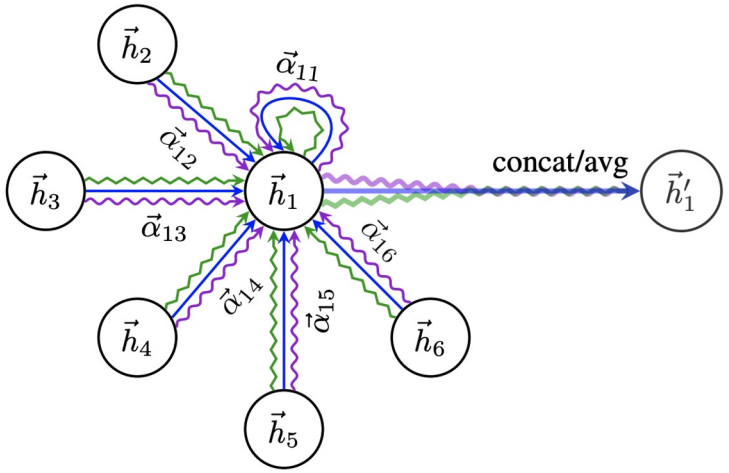
Key idea: dynamically learn the weights (attention scores) on the edges when performing message passing

Weighted sum of node embeddings

$$\mathbf{h}_i^{(l)} = f_{\text{filter}}(A, \mathbf{H}^{(l-1)}) = \sigma\Big(\sum_{v_j \in N(v_i)} \alpha_{ij} \mathbf{W}^{(l)} \mathbf{h}_j^{(l-1)}\Big)$$

Learned local weights with self-attention

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(\mathbf{u}^{(l)T}[\mathbf{W}^{(l)}\mathbf{h}_i^{(l-1)} || \mathbf{W}^{(l)}\mathbf{h}_j^{(l-1)}]))}{\sum_{v_k \in N(v_i)} \exp(\text{LeakyReLU}(\mathbf{u}^{(l)T}[\mathbf{W}^{(l)}\mathbf{h}_i^{(l-1)} || \mathbf{W}^{(l)}\mathbf{h}_k^{(l-1)}]))}$$

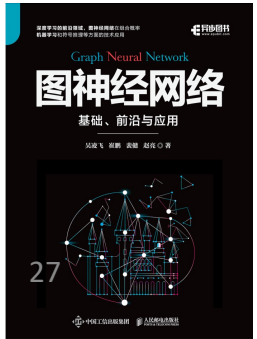


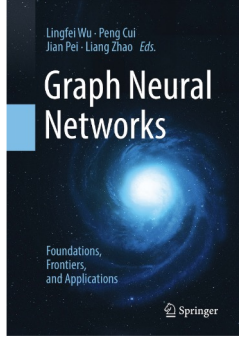
Intermediate node embeddings

$$f_{\text{filter}}(A, \mathbf{H}^{(l-1)}) = ||_{k=1}^{K} \sigma\Big(\sum_{v_j \in N(v_i)} \alpha_{ij}^k \mathbf{W}_k^{(l)} \mathbf{h}_j^{(l-1)}\Big)$$

Final node embeddings

$$f_{\text{filter}}(A, \mathbf{H}^{(L-1)}) = \sigma\Big(\frac{1}{K}\sum_{k=1}^{K}\sum_{v_j \in N(v_i)} \alpha_{ij}^k \mathbf{W}_k^{(L)} \mathbf{h}_j^{(L-1)}\Big)$$





Gated Graph Neural Networks (GGNN)

Key idea: the use of Gated Recurrent Units while taking into account edge type and directions

Zero-padding input node embeddings

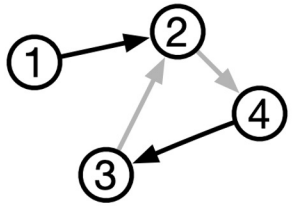
Incoming & outcoming edges for node v_i

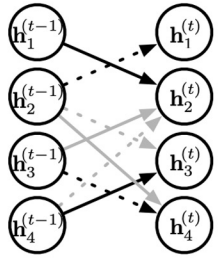
$$\mathbf{h}_i^{(0)} = [\mathbf{x}_i^T, \mathbf{0}]^T$$

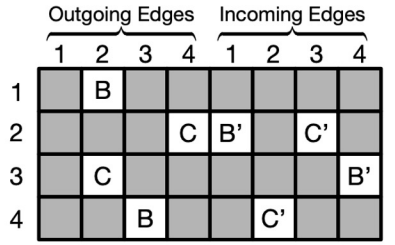
$$\mathbf{a}_i^{(l)} = A_{i:}^T [\mathbf{h}_1^{(l-1)} \ldots \mathbf{h}_n^{(l-1)}]^T$$

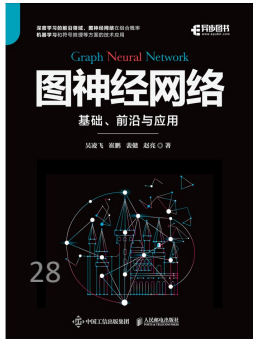
$$\mathbf{h}_i^{(l)} = \mathrm{GRU}(\mathbf{a}_i^{(l)}, \mathbf{h}_i^{(l-1)})$$

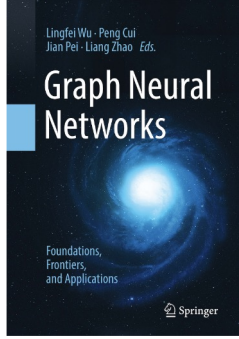
GRU for fusing node embeddings



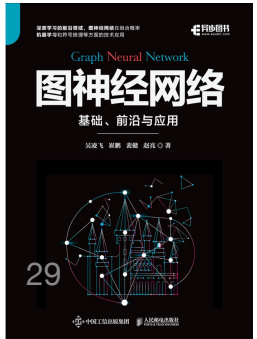




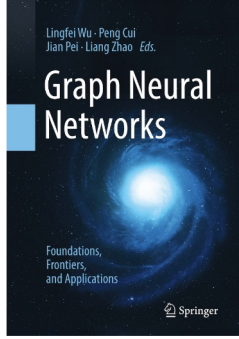




GNNs: Expressive Power (Chapter 5)



Graph Representation Learning and Problem Formulation

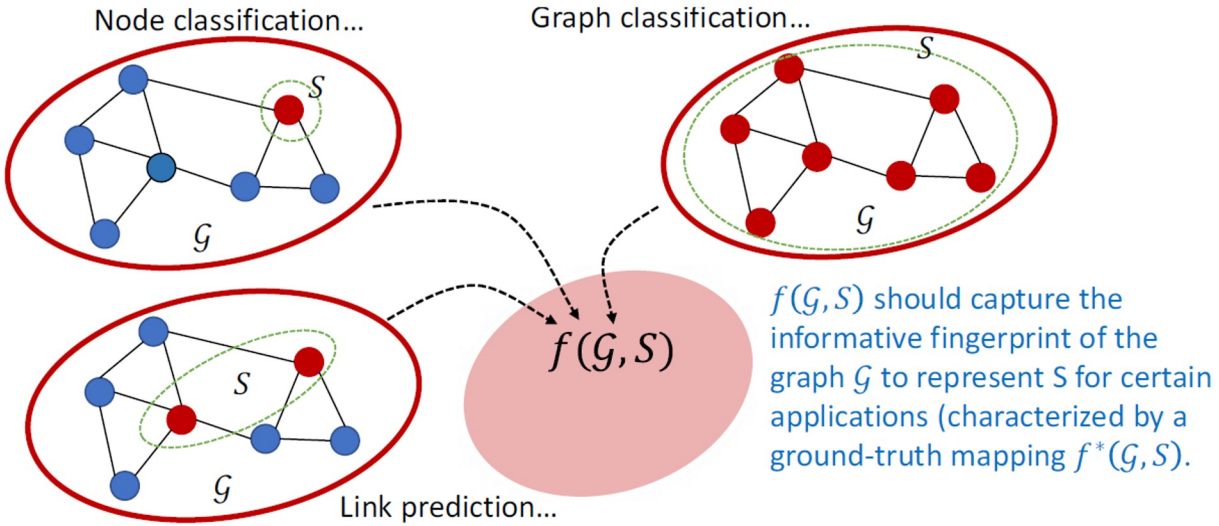


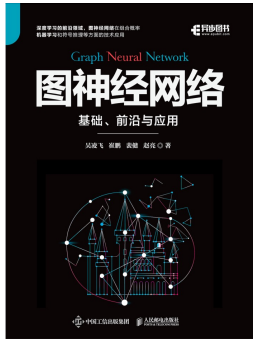
Feature Space $\mathscr{X} := \Gamma \times \mathscr{S}$.

Space of graph-structured data

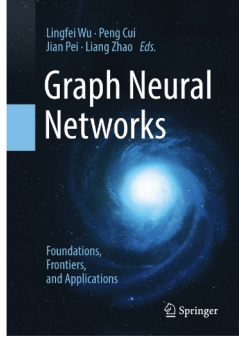
All node set of interest

Point in Feature Space $f^*(\mathscr{G}, S) = y$





Graph Representation Learning and Problem Formulation



Isomorphism:

$$(\mathcal{G}^{(1)}, S^{(1)}), (\mathcal{G}^{(2)}, S^{(2)})$$

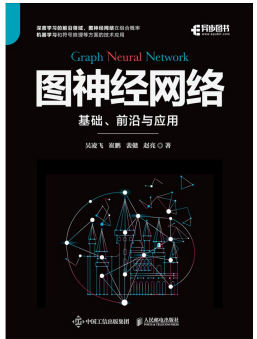
$$\mathcal{G}^{(1)} = (A^{(1)}, X^{(1)})$$

$$\mathcal{G}^{(2)} = (A^{(2)}, X^{(2)})$$

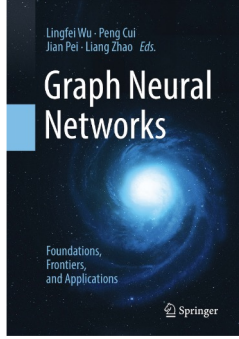
$$\xrightarrow[\text{condition}]{\pi : \mathcal{V}[\mathcal{G}^{(1)}] \to \mathcal{V}[\mathcal{G}^{(2)}],\ A^{(1)}_{uv} = A^{(2)}_{\pi(u)\pi(v)},\ X^{(1)}_{u} = X^{(2)}_{\pi(u)}}$$

$$(\mathcal{G}^{(1)}, S^{(1)}) \cong (\mathcal{G}^{(2)}, S^{(2)})$$

Permutation invariance: $(\mathcal{G}^{(1)}, S^{(1)}) \cong (\mathcal{G}^{(2)}, S^{(2)}), f(\mathcal{G}^{(1)}, S^{(1)}) = f(\mathcal{G}^{(2)}, S^{(2)})$



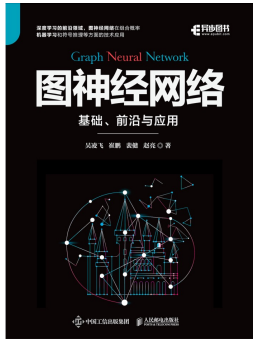
Graph Representation Learning and Problem Formulation

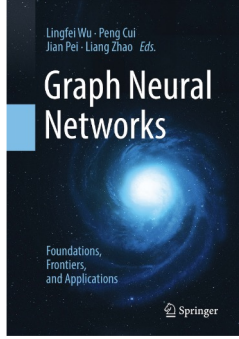


Expressive Power: how a model can distinguish non-isomorphic GRL examples

Definition 5.5. (Expressive power) Consider a feature space $\mathscr{X}$ of a graph representation learning problem and a model f defined on $\mathscr{X}$. Define another space $\mathscr{X}(f)$ as a subspace of the quotient space $\mathscr{X}/\cong$ such that for two GRL examples $(\mathscr{G}^{(1)}, S^{(1)})$, $(\mathscr{G}^{(2)}, S^{(2)}) \in \mathscr{X}(f)$, $f(\mathscr{G}^{(1)}, S^{(1)}) \neq f(\mathscr{G}^{(2)}, S^{(2)})$. Then, the size of $\mathscr{X}(f)$ characterizes the expressive power of f. For two models, $f^{(1)}$ and $f^{(2)}$, if $\mathscr{X}(f^{(1)}) \supset \mathscr{X}(f^{(2)})$, we say that $f^{(1)}$ is more expressive than $f^{(2)}$.

How to build the most expressive permutation invariant GNNs for graph representation learning problems?





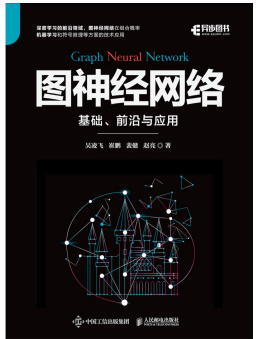
The Expressive Power of GNNs

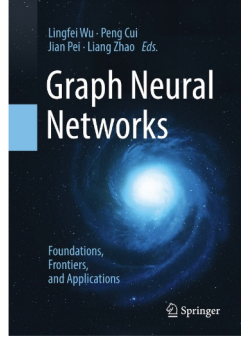
Question 1: **Function approximation**

What class of functions can GNNs approximate?

Question 2: Distinguishing graph structures

Whether can GNNs distinguish two different graph structures or not ?





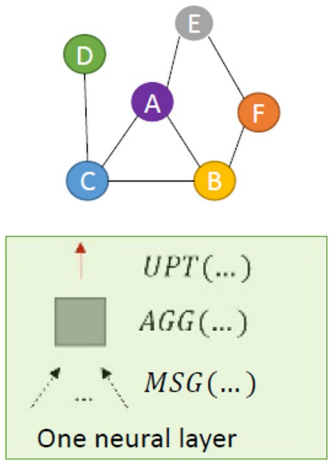
Message Passing Graph Neural Networks

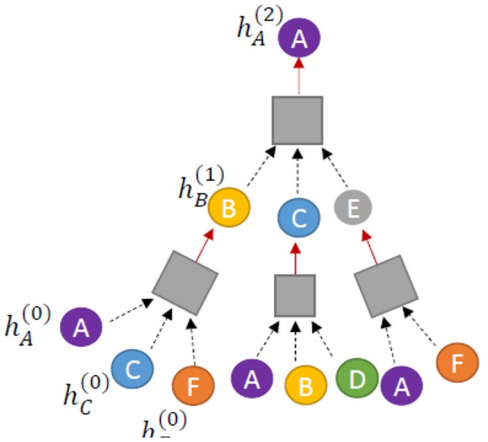
1. Initialize node vector representations as node attributes: $\mathbf{h}_v^{(0)} \leftarrow X_v, \forall v \in \mathcal{V}$.
2. Update each node representation based on message passing over the graph structure. In l-th layer, $l = 1, 2, \ldots, L$, perform the following steps:

$$\text{Message:} \quad \mathbf{m}_{vu}^{(l)} \leftarrow \text{MSG}(\mathbf{h}_v^{(l-1)}, \mathbf{h}_u^{(l-1)}), \forall (u,v) \in \mathcal{E},$$

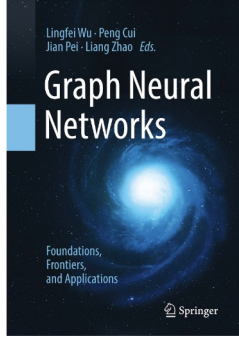
$$\text{Aggregation:} \quad \mathbf{a}_v^{(l)} \leftarrow \text{AGG}(\{\mathbf{m}_{vu}^{(l)} | u \in \mathcal{N}_v\}), \forall v \in \mathcal{V},$$

$$\text{Update:} \quad \mathbf{h}_v^{(l)} \leftarrow \text{UPT}(\mathbf{h}_v^{(l-1)}, \mathbf{a}_v^{(l)}), \forall v \in \mathcal{V}.$$





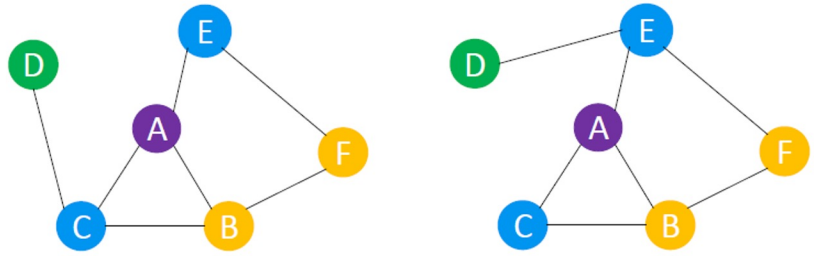
Theorem 5.1. *(Invariance of MP-GNN) $f_{MP-GNN}(\cdot,\cdot)$ satisfies permutation invariance (Def. 5.4) as long as the AGG and READOUT operations are invariant pooling operations (Def. 5.7).*



The Expressive Power of MP-GNN

MP-GNN is at most as powerful as the 1-WL test in distinguishing different graph-structured features (upper bounds).

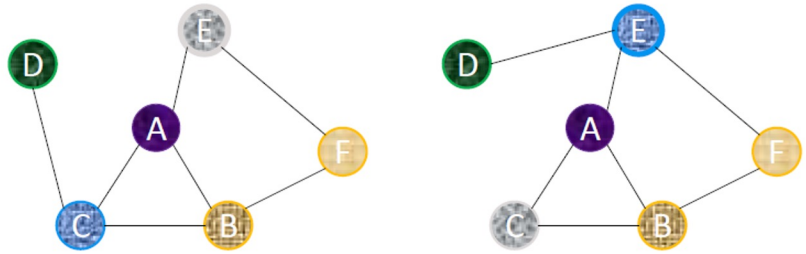
Step 1: Each node is initialized with some color according to its attribute (if no attributes, use the same color).



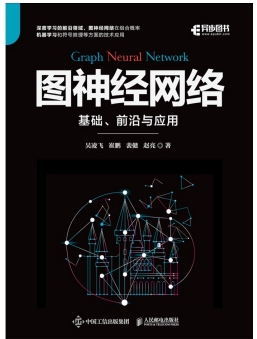
The mapping "attributes → colors" is injective.

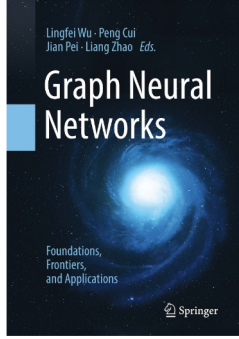
Step 2: Each node will collect the colors from their neighbors:

Node A: (p,{bby})
Left node E: (b,{py});
Right node E: (b,{pyg}) ...



The mapping "(self-color, set of colors from neighbors) → a new color" is injective





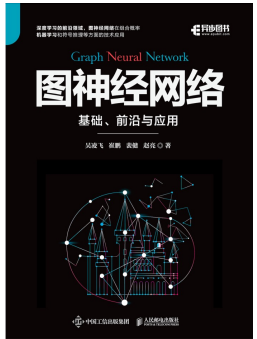
The Expressive Power of MP-GNN

What kinds of GNNs that are, in principle, as powerful as the 1-WL test?

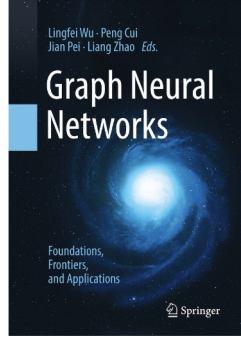
Theorem 5.3. *(Theorem 3 in (Xu et al, 2019d)) After sufficient iterations, MP-GNN may map any GRL examples* $(\mathcal{G}^{(1)}, S^{(1)})$ *and* $(\mathcal{G}^{(2)}, S^{(2)})$*, that the 1-WL test decides as non-isomorphic, to different representations if the following two conditions hold:*

a) The composition of MSE, AGG and UPT (Eqs.equation 11.45-equation 5.3) constructs an injective mapping from $(\mathbf{h}_v^{(k-1)}, \{\mathbf{h}_u^{(k-1)} | u \in \mathcal{N}_v\})$ *to* $\mathbf{h}_v^{(k)}$.

b) The READOUT (Eq.equation 5.4) is injective.



MP-GNN with the Power of the 1-WL Test



a) The composition of MSE, AGG and UPT (Eqs.equation 11.45-equation 5.3) constructs an injective mapping from $(\mathbf{h}_v^{(k-1)}, \{\mathbf{h}_u^{(k-1)} | u \in \mathcal{N}_v\})$ *to* $\mathbf{h}_v^{(k)}$.

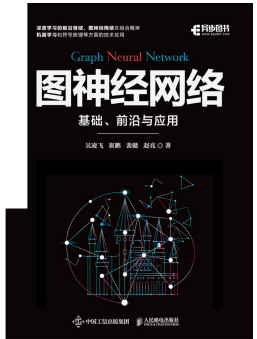
AGG operation is suggested to adopt the sum pooling operation

(Injective multiset functions: record the number of repetitive elements)

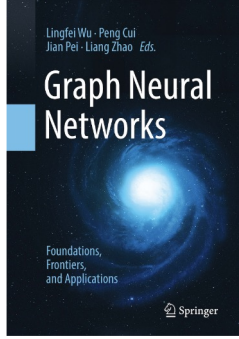
$$\text{Expressive Message:} \quad \mathbf{m}_{vu}^{(k)} \leftarrow MLP_1^{(k-1)}(\mathbf{h}_v^{(k-1)} \oplus \mathbf{h}_u^{(k-1)}), \ \forall (u,v) \in \mathcal{E},$$

$$\text{Expressive Aggregation:} \quad \mathbf{a}_v^{(k)} \leftarrow \sum_{u \in \mathcal{N}_v} \mathbf{m}_{vu}^{(k)}, \ \forall v \in \mathcal{V},$$

$$\text{Expressive Update:} \quad \mathbf{h}_v^{(k)} \leftarrow MLP_2^{(k-1)}(\mathbf{h}_v^{(k-1)} \oplus \mathbf{a}_v^{(k)}), \ \forall v \in \mathcal{V}.$$



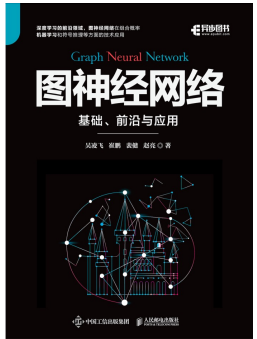
MP-GNN with the Power of the 1-WL Test

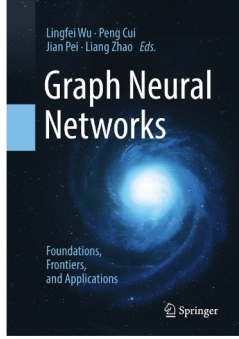


b) The READOUT (Eq.equation 5.4) is injective.

Expressive Inference: $\hat{y}_S = MLP(\sum_{v \in S} \mathbf{h}_v^{(L)})$.

- Combine the proof for injectiveness of the sum aggregation with the universal approximation property of MLP.

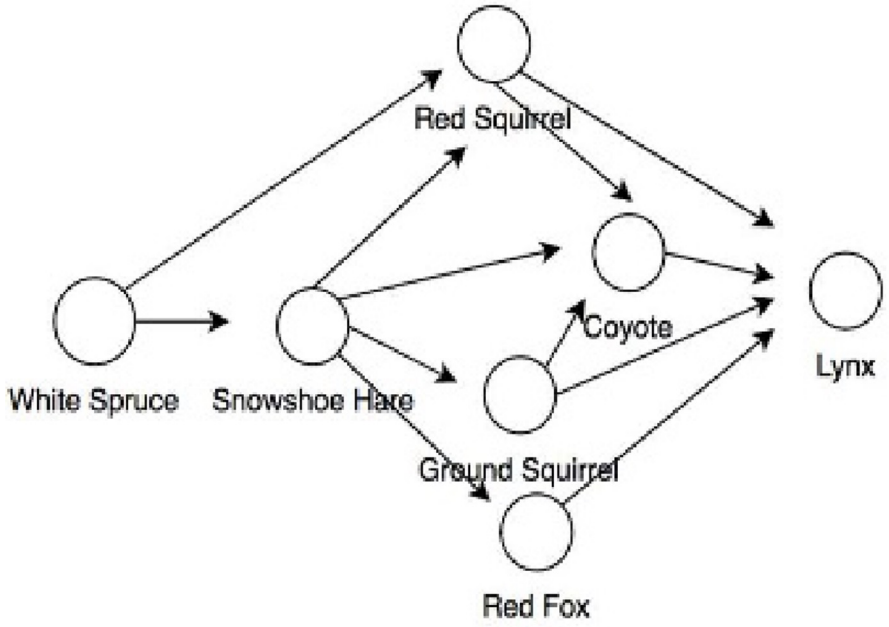


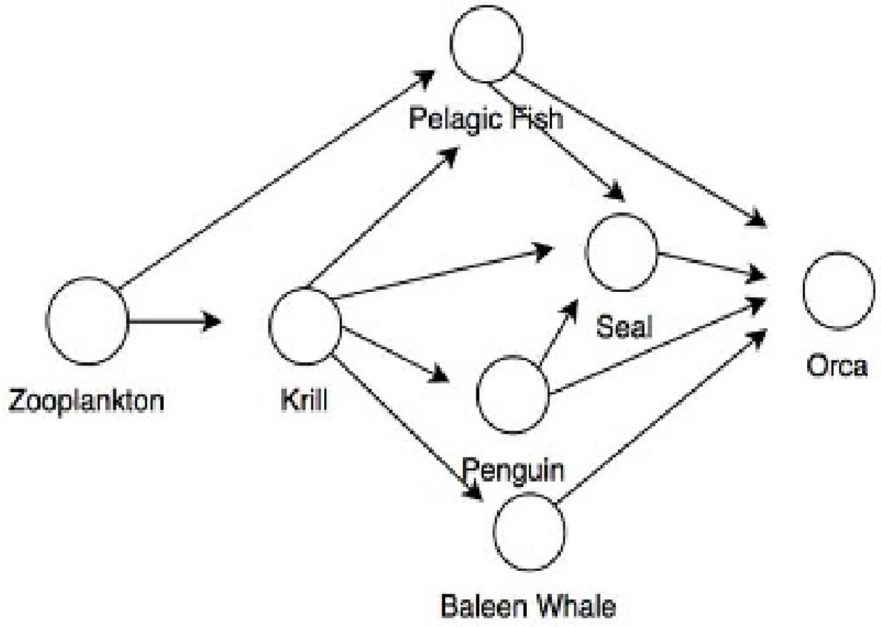


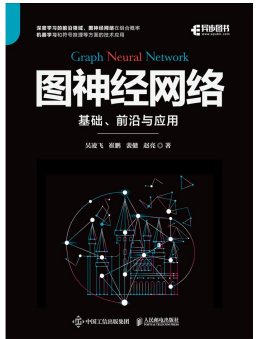
GNNs that are more Powerful than 1-WL Test

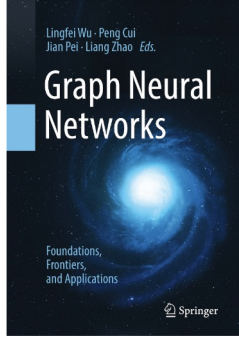
Information lost when using MP-GNN

- The information about the distance between multiple nodes is lost.





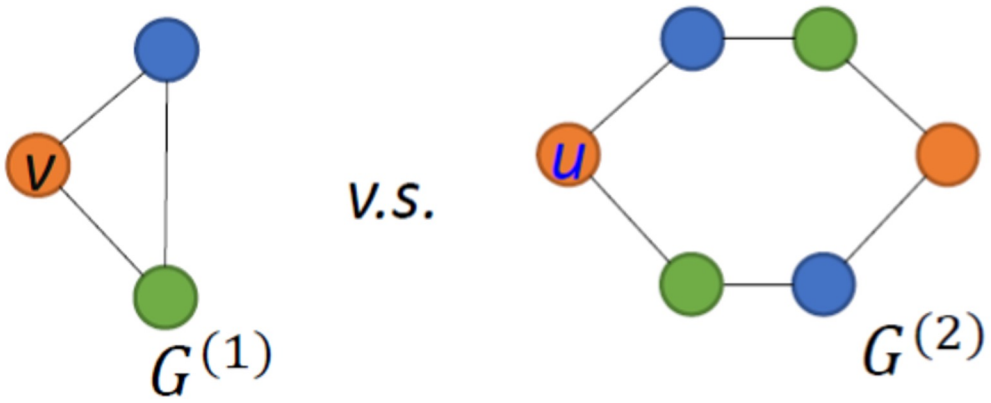




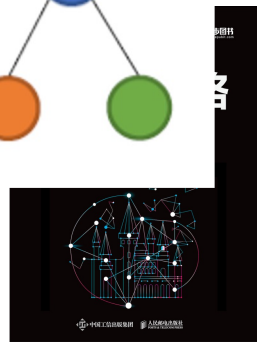
GNNs that are more Powerful than 1-WL Test

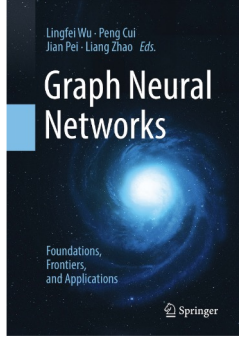
Information lost when using MP-GNN

- The information about cycles is lost.



Corresponding subtrees:



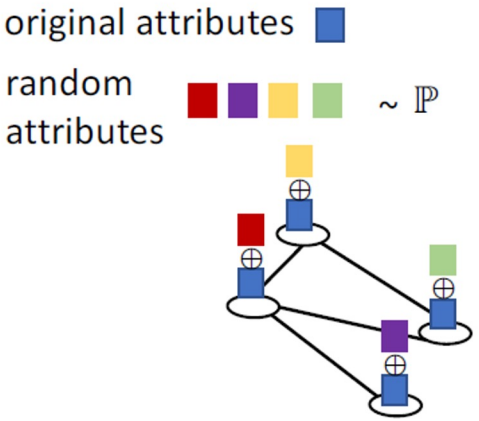


GNNs that are more Powerful than 1-WL Test

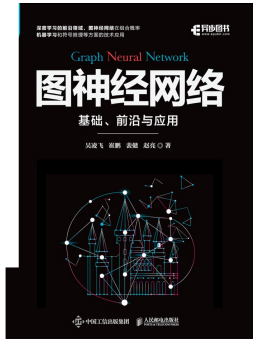
$$g_I(\mathcal{G}, S) = (\mathcal{G}_I, S), \quad \text{where } \mathcal{G}_I = (A, X \oplus \mathbf{I}),$$

Problem: it is not permutation invariant

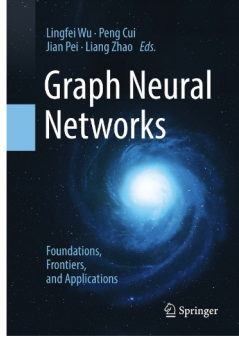
Injecting Random Attributes



Types of random attributes	Positional information	Model & reference
Random permutations	No	RP-GNN (Murphy et al, 2019)
(Almost uniform) Discrete r.v.	No	rGIN (Sato et al, 2020)
Distances to random anchor sets	Yes	PGNN (You et al, 2019)
Graph-convoluted Gaussian r.v.	Yes	CGNN (Srinivasan & Ribeiro, 2020)
Random signed Laplacian eigenmap	Yes	LE-GNN (Dwivedi et al, 2020)



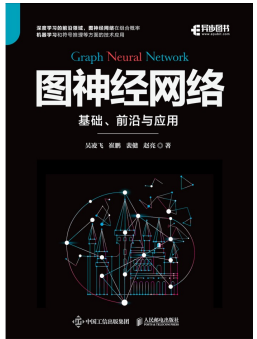
GNNs that are more Powerful than 1-WL Test

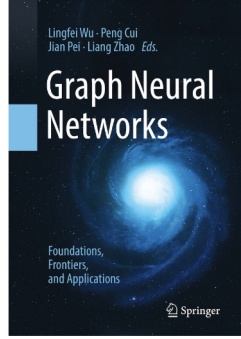


Injecting Deterministic Distance Attributes.

Two important pieces of intuition:

- Effective distance information is typically correlated with the task
- Distance from S to other nodes in G may be also useful side information.





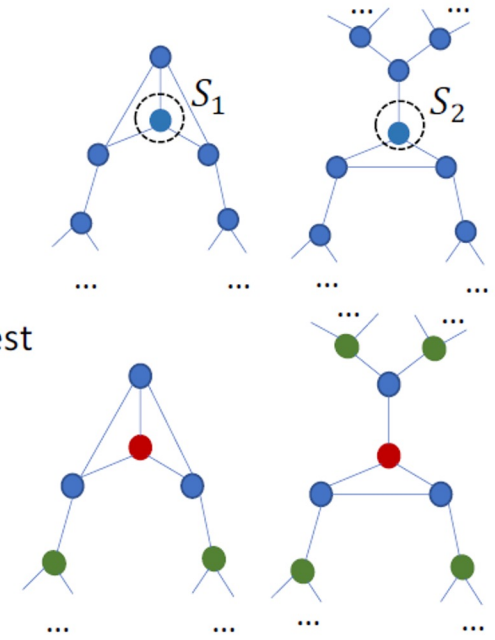
GNNs that are more Powerful than 1-WL Test

Distance Encoding (Li et al (2020e) DE-GNN)

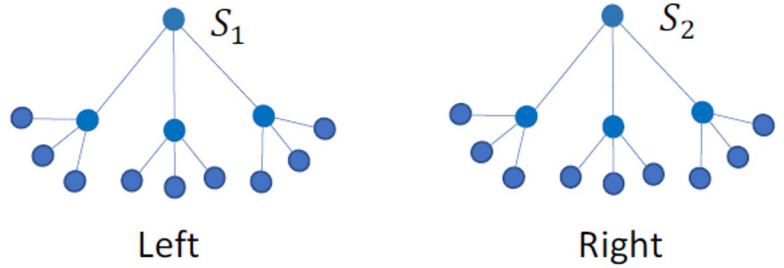
Node classification
for structural-role prediction

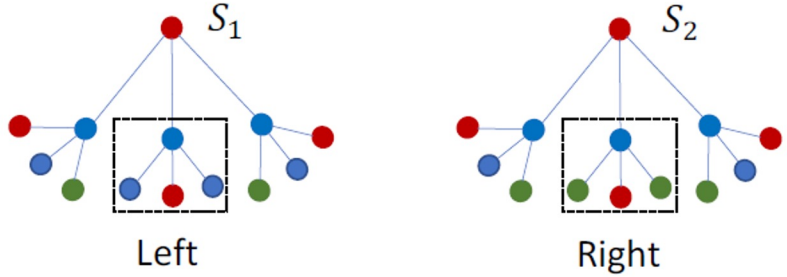
+ distance encoding (use shortest path distance as an example)

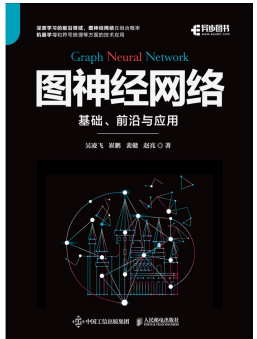
- DE = 0
- DE = 1
- DE = 2

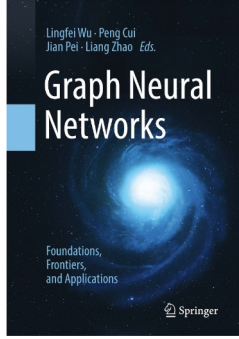


Subtrees rooted at the nodes of interest



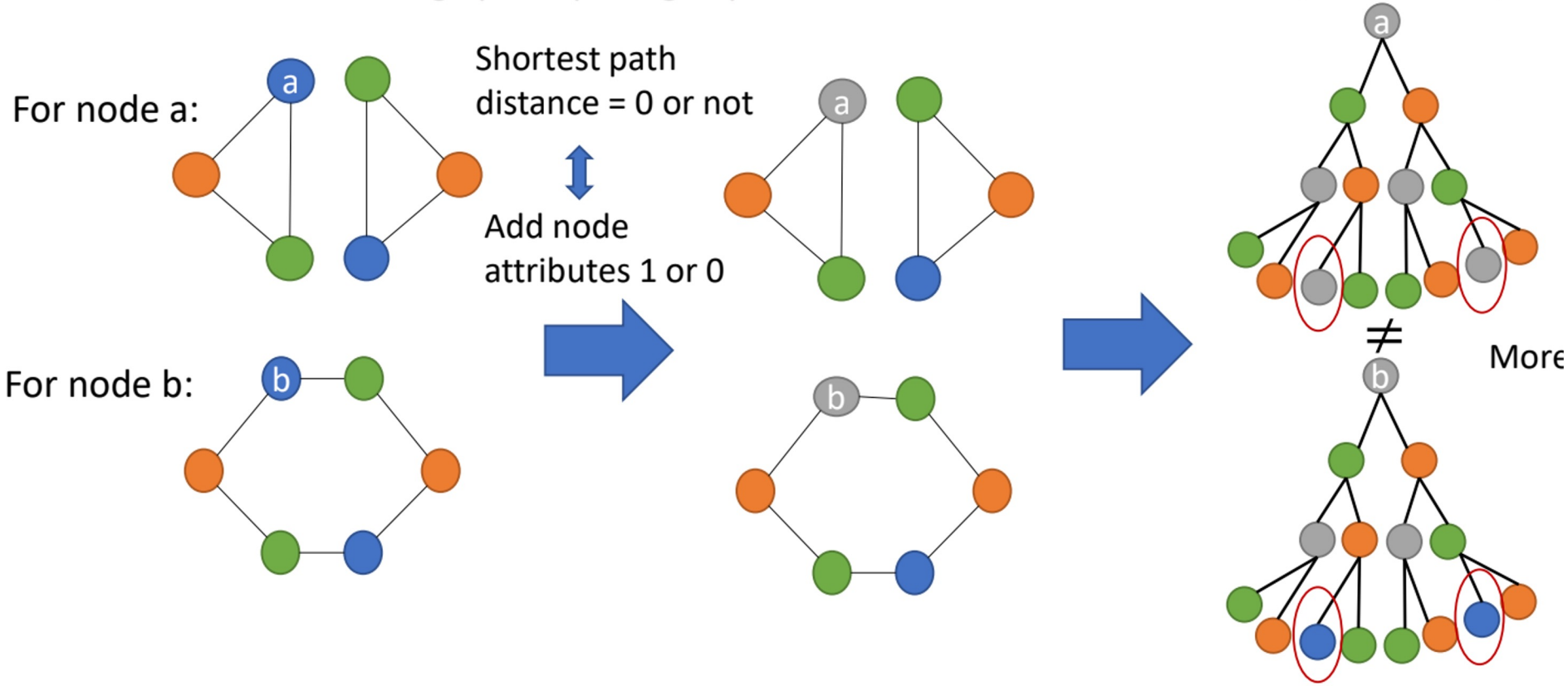


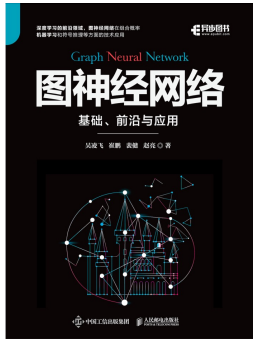


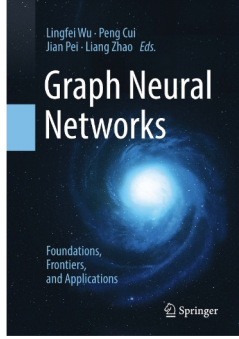


GNNs that are more Powerful than 1-WL Test

Distance Encoding (Identity-aware GNN (You at al, 2021)



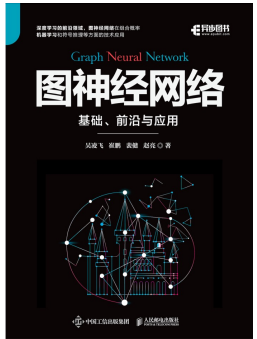


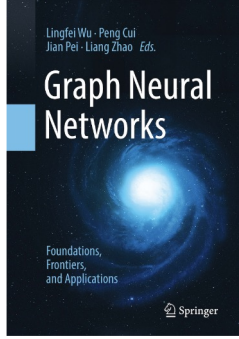


GNNs that are more Powerful than 1-WL Test

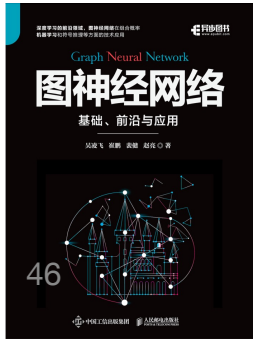
Higher-order Graph Neural Networks.

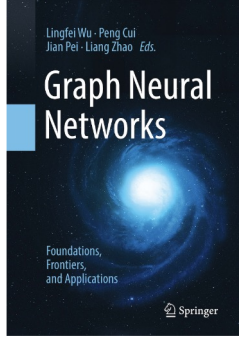
- k-WL-induced GNNs (Morris et al, 2019)
- Invariant and equivariant GNNs (Maron et al, 2018, 2019b)
- k-FWL-induced GNNs (Maron et al, 2019a; Chen et al, 2019f)





GNNs: Interpretability (Chapter 7)



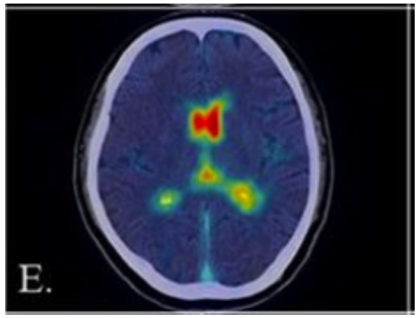


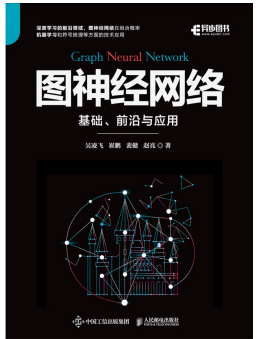
Interpretability of AI models

Interpretability is the degree to which an observer can understand the cause of a decision.

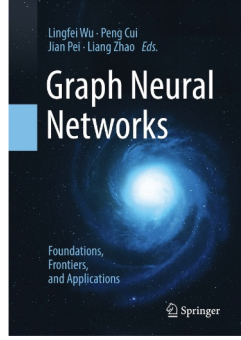
An **interpretation** is the mapping of an abstract concept into a domain that the human can understand.

- **Model-Oriented Reasons**
  - Credibility
  - Fairness
  - Adversarial-Attack Robustness
  - Backdoor-Attack Robustness
- **User-Oriented Reasons**
  - Improving User Experiences (e.g., cancer diagnosis, recommender system, etc.)
  - Facilitating Decision Making (e.g., the cause of the traffic jam prediction)

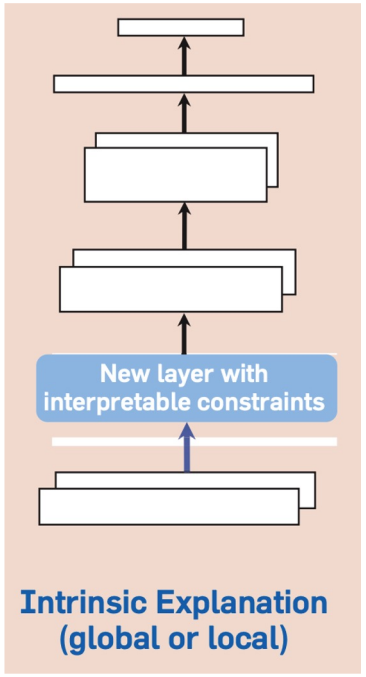




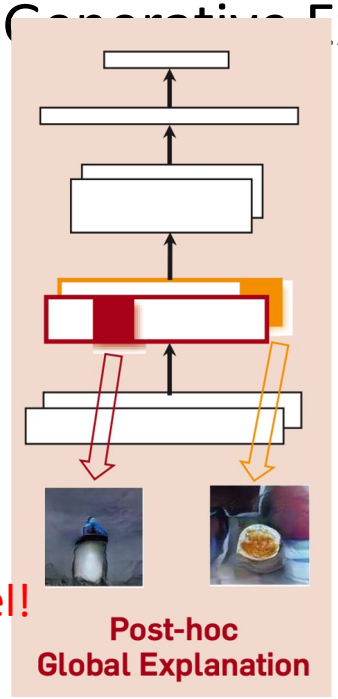
Types of Interpretations

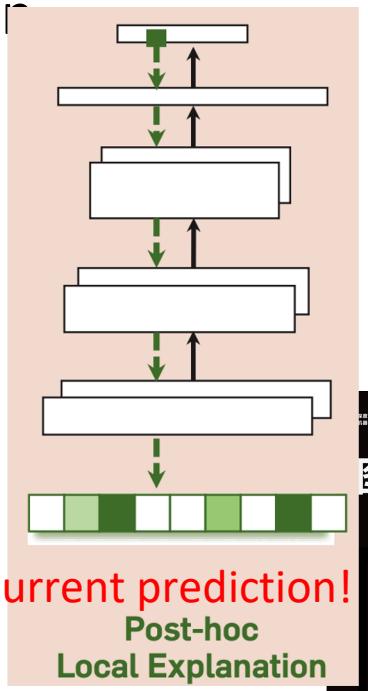


- Intrinsic Explanation
  - Distillation.
  - Attention models.
  - Disentangled representation learning.

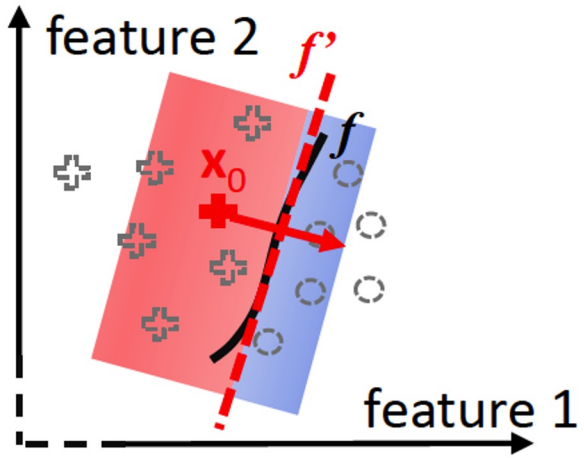


- Post-Hoc Interpretation
  - Approximation-Based Explanation
  - Relevance-Propagation Based Explanation
  - Perturbation-Based Approaches
  - Generative Explanation

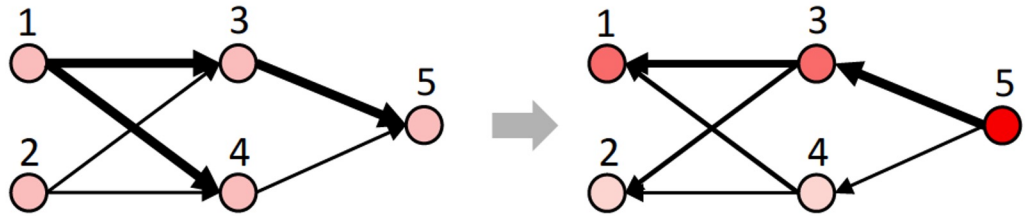




Types of post-hoc interpretations

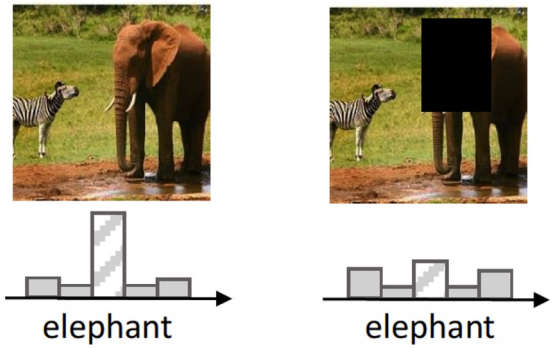


Approximation-Based Explanation

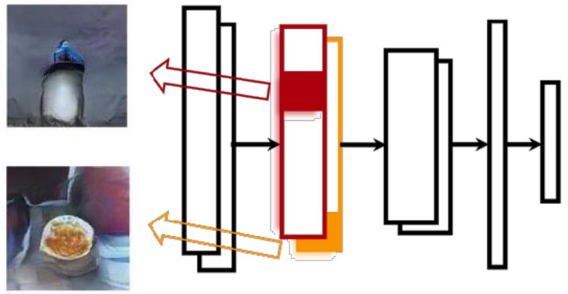


feed-forward connection weights relevance propagation

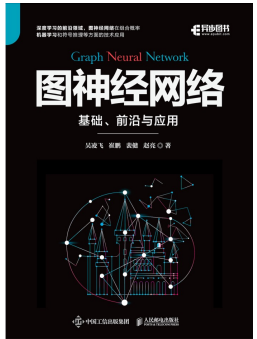
Relevance-Propagation Based Explanation



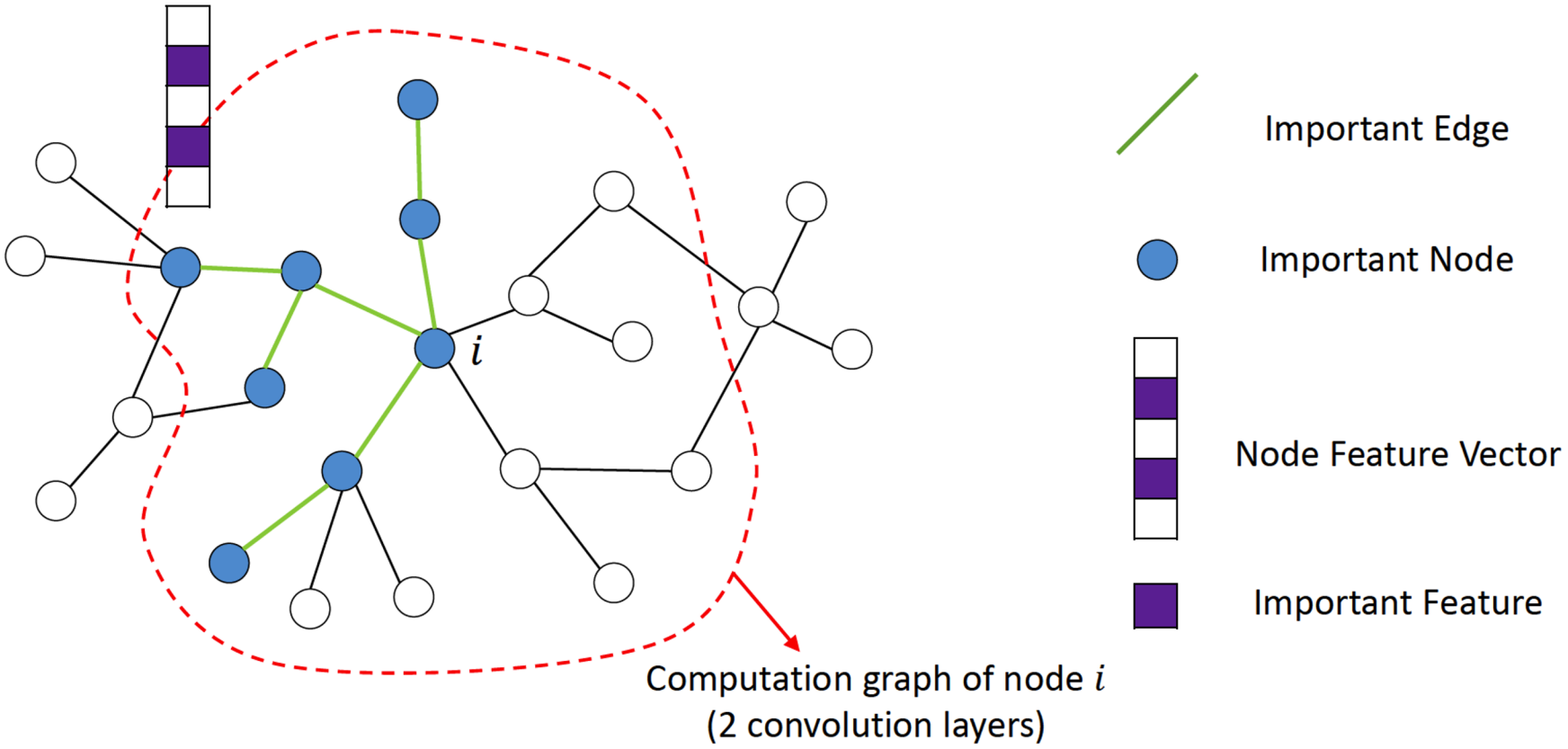
Perturbation-Based Approaches

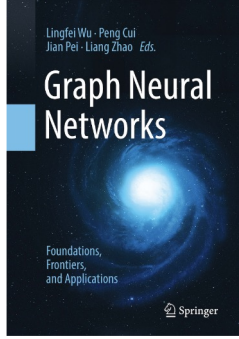


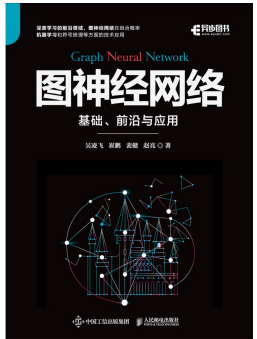
Generative Explanation



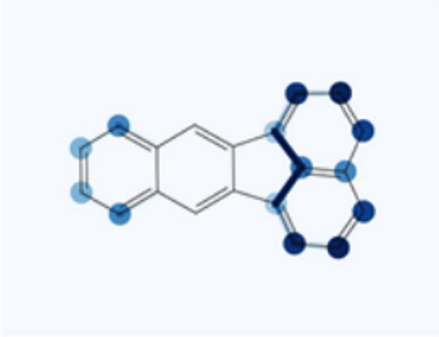
Formats of Explanation on Graphs



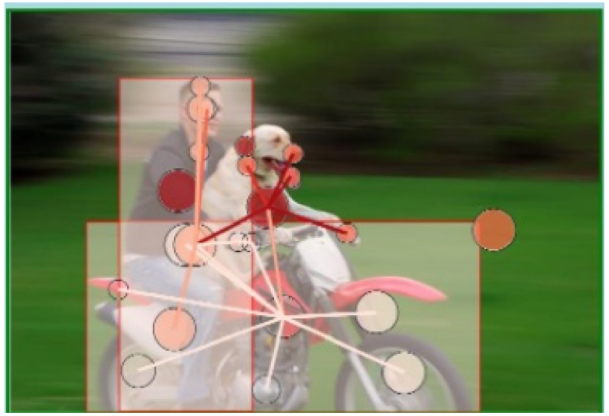




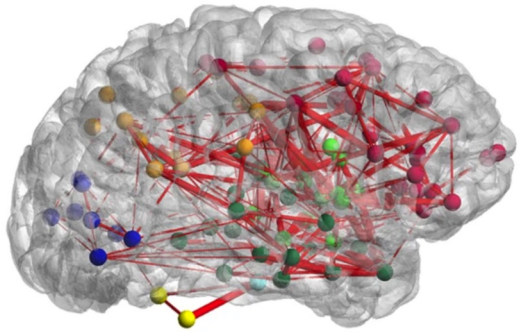
Intepretable GNN on graphs



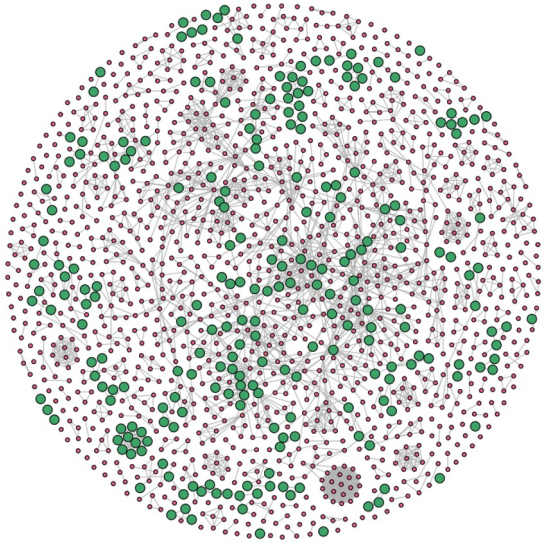
Which part determines the toxicity?



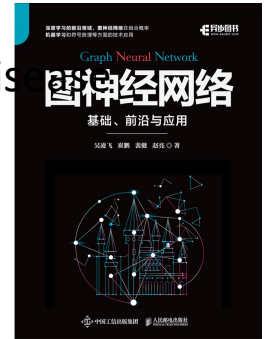
Why this image is about driving?



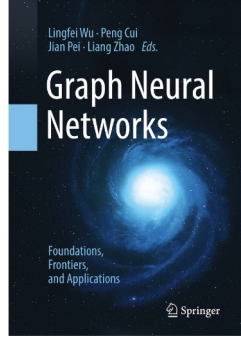
The rationale of mental disease prediction?



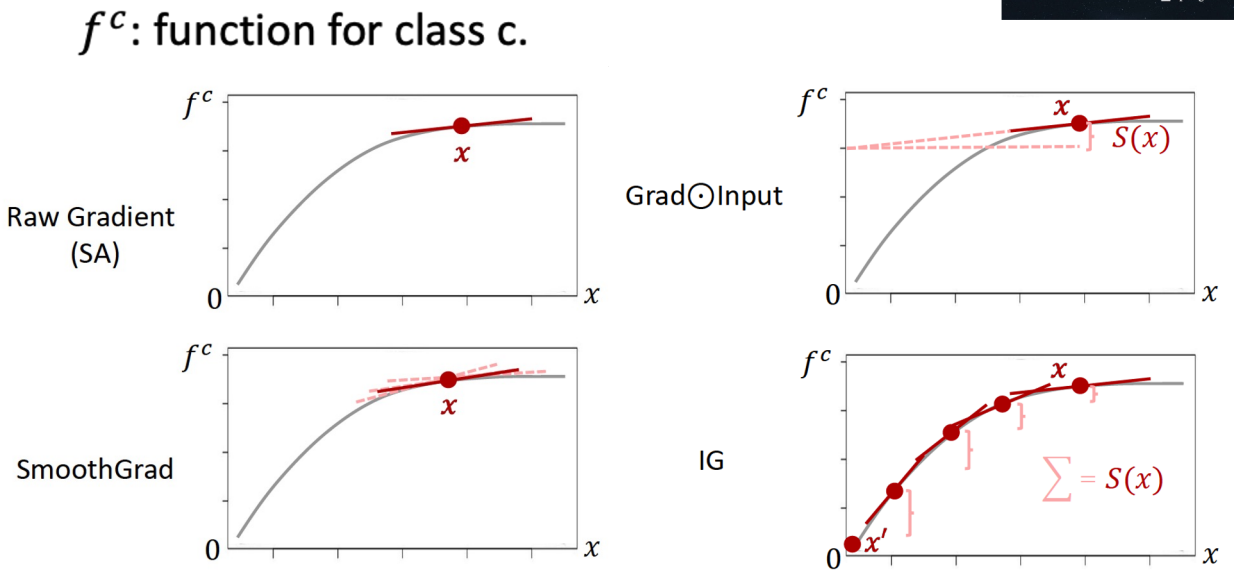
Source of infection in dis
contact network?



Approximation-Based Explanation



- White-Box Approximation
  - Input Gradient [Baldassarre et al. 2019]
  - Grad*Input [Sanchez-Lengeling et al. 2020]
  - Integrated Gradients (IG)
  - SmoothGrad [Sanchez-Lengeling et al. 2020]
  - Class Activation Mapping (CAM) [Pope et al. 2019]
  - Grad-CAM
  - [Pope et al. 2019]
- Black-Box Approximation Methods
  - GraphLime [Huang et al. 2022]
  - RelEx [Zhang et al. 2020]
  - PGM-Explainer [Vu et al. 2020]



•[Baldassarre 2019] Baldassarre F, Azizpour H (2019) Explainability techniques for graph convolutional networks. arXiv preprint arXiv:190513686

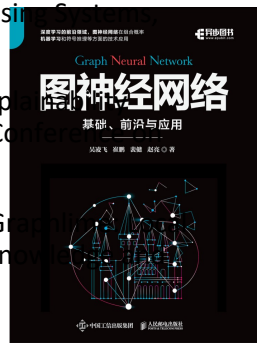
•[Sanchez-Lengeling et al. 2020] Sanchez-Lengeling B, Wei J, Lee B, Reif E, Wang P, Qian W, McCloskey K, Colwell L, Wiltschko A (2020) Evaluating attribution for graph neural networks. In: Larochelle H, Ranzato M, Hadsell R, Balcan MF, Lin H (eds) Advances in Neural Information Processing Systems, Curran Associates, Inc., vol 33, pp 5898–5910

•[Pope et al. 2019] Pope PE, Kolouri S, Rostami M, Martin CE, Hoffmann H (2019) Explainability methods for graph convolutional neural networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp 10,772–10,781

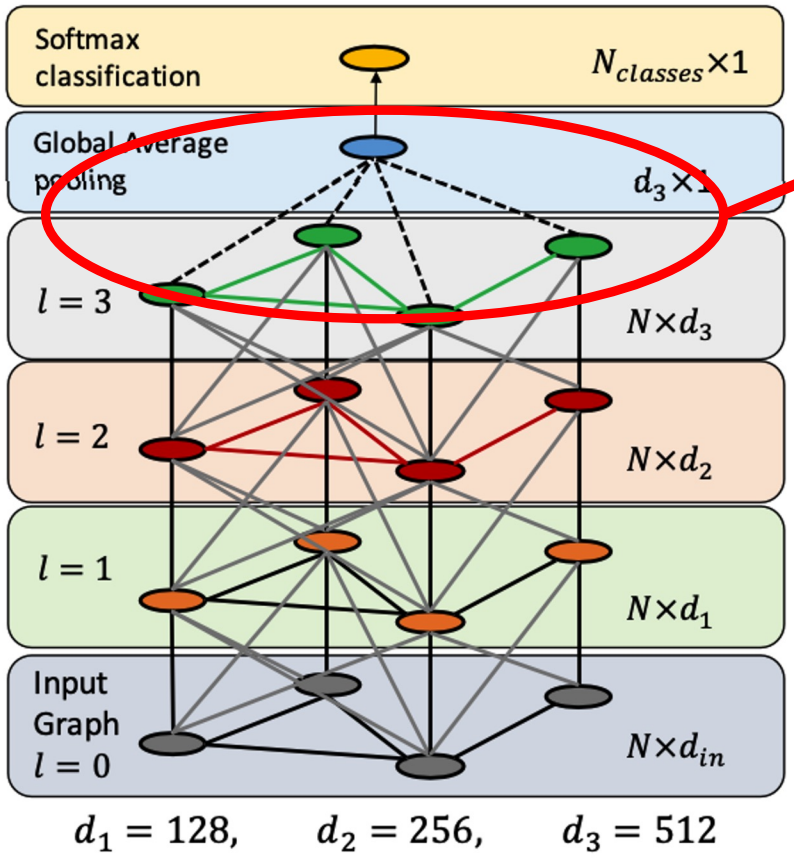
•[Huang et al. 2022] Huang, Q., Yamada, M., Tian, Y., Singh, D., & Chang, Y. (2022). GraphLIME: Local interpretable model explanations for graph neural networks. IEEE Transactions on Knowledge and Data Engineering.

•[Zhang et al. 2020] Zhang M, Chen Y (2020) Inductive matrix completion based on graph neural networks. In: International Conference on Learning Representations.

•[Vu et al. 2020] Vu, Minh, and My T. Thai. "Pgm-explainer: Probabilistic graphical model explanations for graph neural networks." Advances in neural information processing systems 33 (2020): 12225-12235.



CAM and GradCAM for GNNs



[Pope et al. 2019]

CAM

$$\mathbf{h}=\frac{1}{n}\sum_{i=1}^{n}H_{i,:}^{L}$$

$$f^c(\mathscr{G})=\sum_k w_k^c\mathbf{h}_k$$

Extension 1: Generalization to any form of f^c

GradCAM

$$w_k^c=\frac{1}{n}\sum_{i=1}^{n}\frac{\partial f^c(\mathscr{G})}{\partial H_{i,k}^{L}}$$

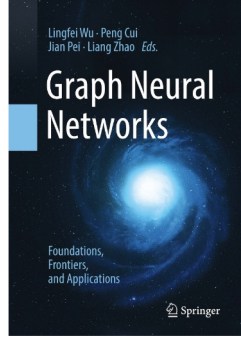
$$\mathscr{S}(i)=\frac{1}{n}\sum_k w_k^c H_{i,k}^{L}.$$

$$\mathscr{S}(i)=ReLU\left(\sum_k w_k^c H_{i,k}^{L}\right)$$

Extension 2: Add ReLU to get rid of negative features.

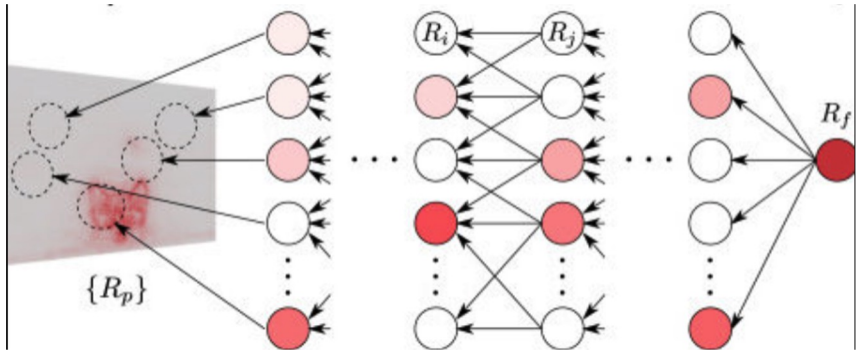
•[Pope et al. 2019] Pope PE, Kolouri S, Rostami M, Martin CE, Hoffmann H (2019) Explainability methods for graph convolutional neural networks. In: Proceedings IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp 10,772–10,781

Relevance-Propagation Based Explanation



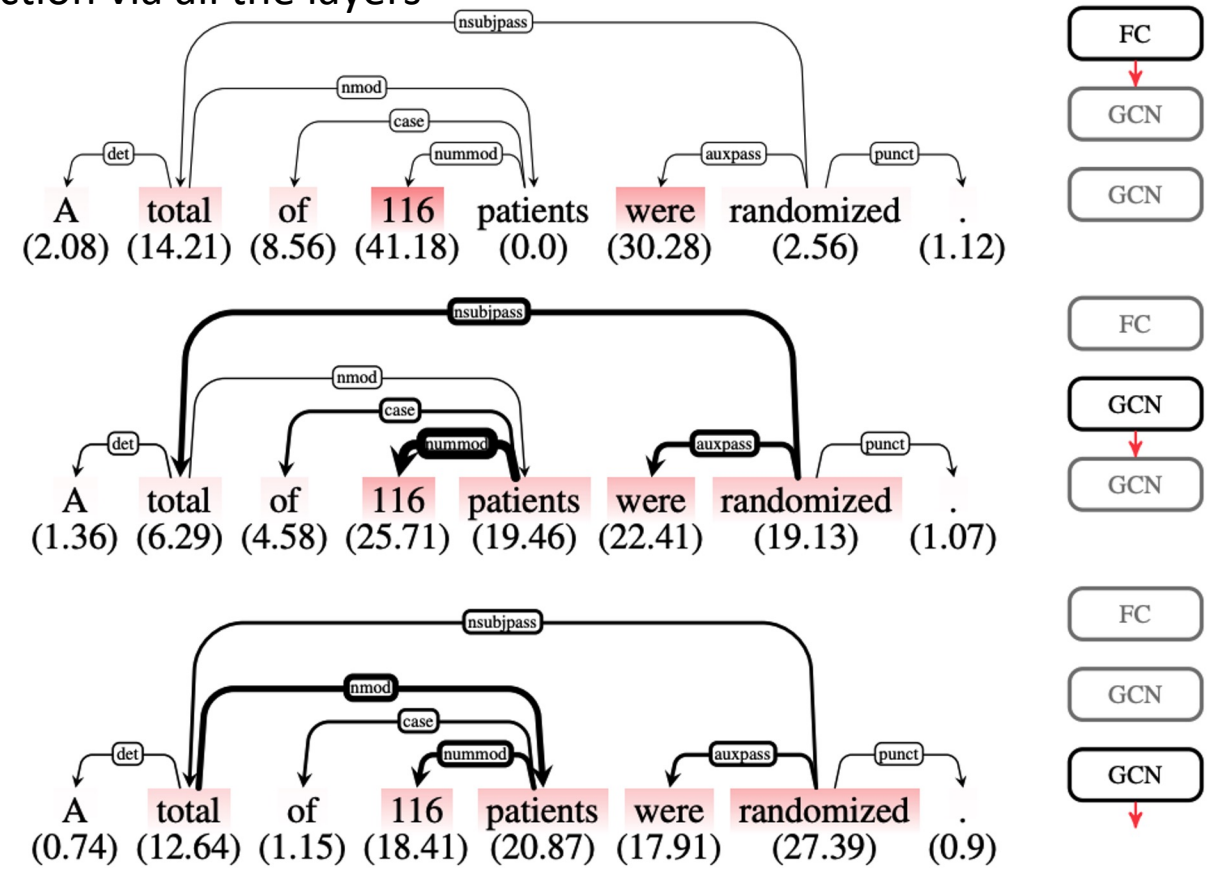
- Layer-wise Relevance Propagation (LRP) [Bach et al. 2015, Baldassarre and Azizpour. 2019; Schwarzenberg et al. 2019]

Quantify the Contribution of each input neuron to the prediction via all the layers

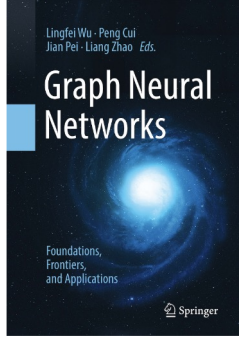


$$R_i^l = \sum_j \frac{z_{i,j}^+}{\sum_k z_{k,j}^+ + b_j^+ + \varepsilon} R_j^{l+1}$$

$$z_{i,j} = x_i^l w_{i,j}$$



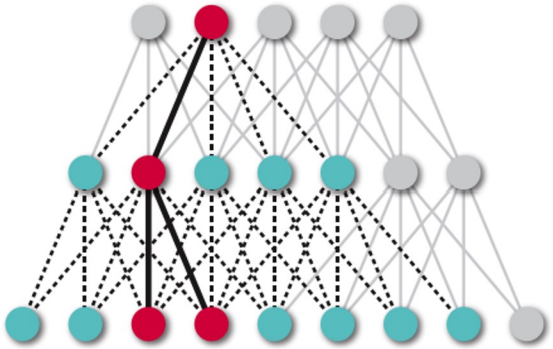
Relevance-Propagation Based Explanation



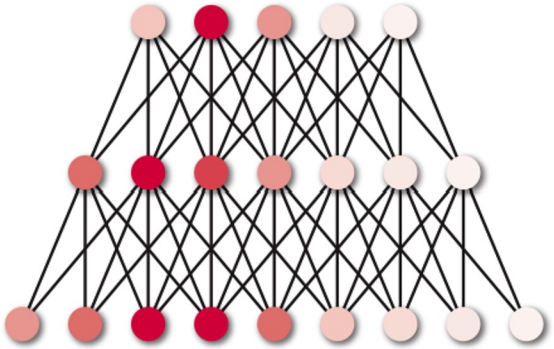
- Excitation BackPropagation (EBP) [Zhang et al, 2018]

$$P(a_i) = \sum_{a_j \in \mathcal{P}_i} P(a_i|a_j)P(a_j)$$

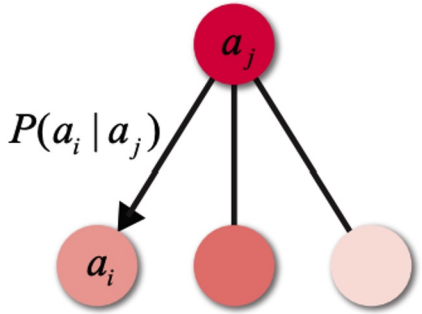
- Only non-negative weights are considered and negative weights are set to zero.



(a) Deterministic WTA

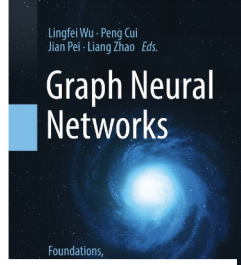


(b) Probabilistic WTA



(c) Winner Sampling

[Zhang et al, 2018d] Zhang J, Bargal SA, Lin Z, Brandt J, Shen X, Sclaroff S (2018d) Top-down neural attention by excitation backprop. International Journal of Computer Vision 126(10):1084–1102



Approximation-Based Explanation

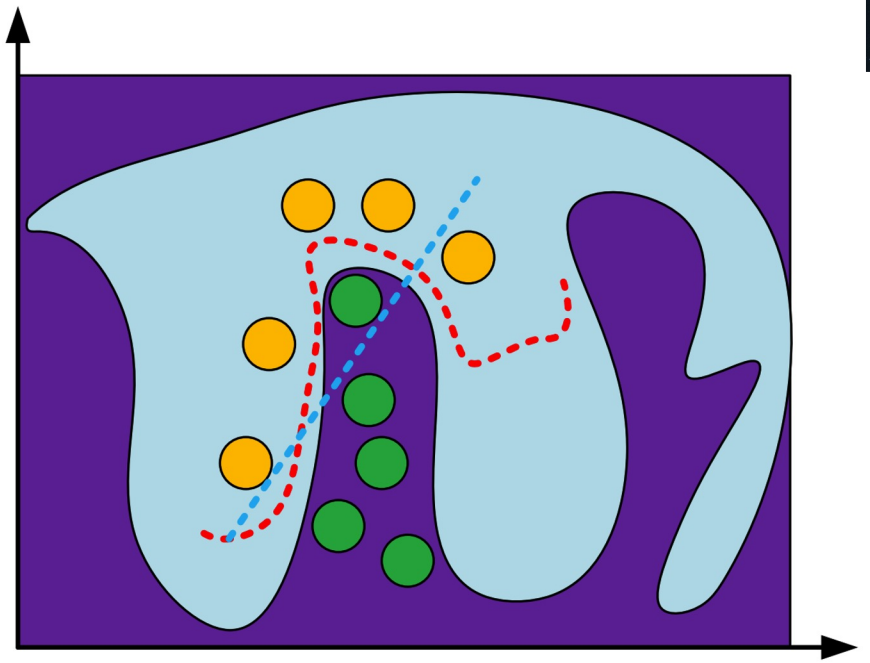
- GraphLime: Extend LIME to GNN [Huang et al. 2022]
  - Consider the N-hop neighbors

$$\boldsymbol{X}_n = [\boldsymbol{x}_1, \boldsymbol{x}_2, ...\boldsymbol{x}_m]$$

  - Use non-linear kernel-based model parameterized by β to approximate the original model

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^d} \frac{1}{2}\|\bar{\boldsymbol{L}} - \sum_{k=1}^{d} \beta_k \bar{\boldsymbol{K}}^{(k)}\|_F^2 + \rho\|\boldsymbol{\beta}\|_1$$

$$\text{s.t.} \quad \beta_1, \ldots, \beta_d \geq 0,$$



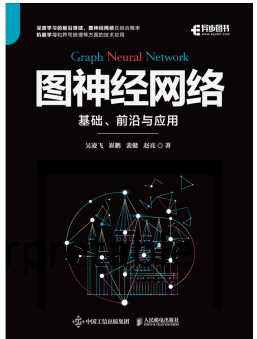
top K non-zero β_k are indicating representative K features

$$\bar{\boldsymbol{L}} = \ddot{\boldsymbol{H}}\boldsymbol{L}\ddot{\boldsymbol{H}}/\|\boldsymbol{H}\boldsymbol{L}\boldsymbol{H}\|_F$$

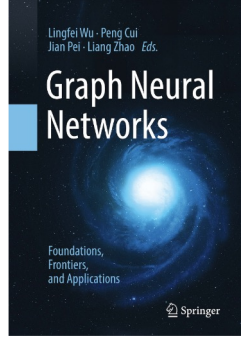
$$K(\boldsymbol{x}_i^{(k)}, \boldsymbol{x}_j^{(k)}) = \exp\left(-\frac{(\boldsymbol{x}_i^{(k)} - \boldsymbol{x}_j^{(k)})^2}{2\sigma_x^2}\right),$$

$$\bar{\boldsymbol{K}}^{(k)} = \boldsymbol{H}\boldsymbol{K}^{(k)}\boldsymbol{H}/\|\boldsymbol{H}\boldsymbol{K}^{(k)}\boldsymbol{H}\|_F$$

$$L(y_i, y_j) = \exp\left(-\frac{\|y_i - y_j\|_2^2}{2\sigma_y^2}\right),$$



•[Huang et al. 2022] Huang, Q., Yamada, M., Tian, Y., Singh, D., & Chang, Y. (2022). Graphlime: Local interpretable model explanations for graph neural networks. IEEE Transactions on Knowledge and Data Engineering.



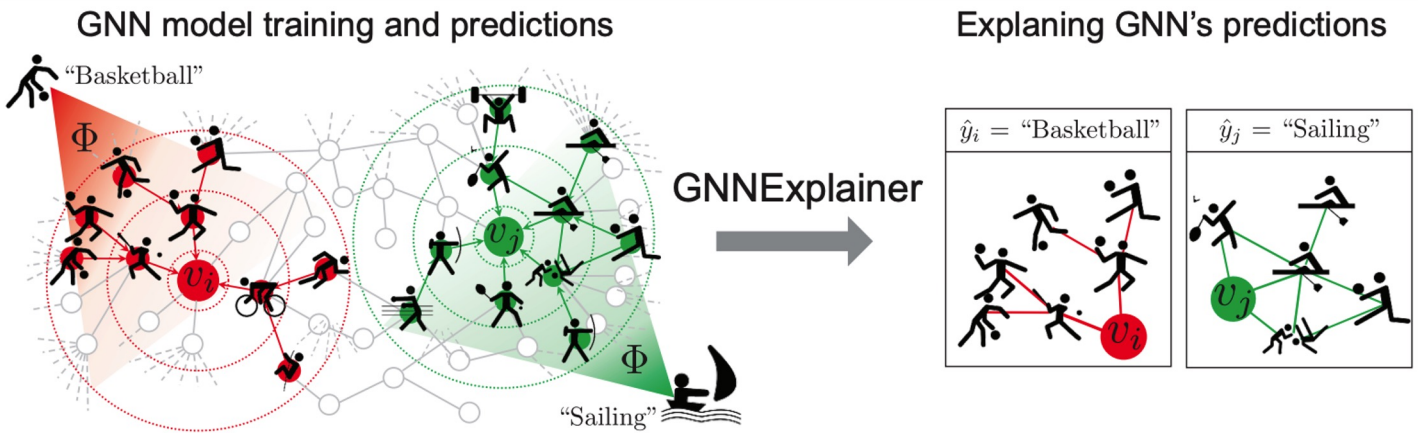
Perturbation-Based Approaches

- GNNExplainer [Ying et al. 2019]: Find the portion of graphs that has largest mutual information with the prediction

$$\max_{G_S} MI\left(Y, (G_S, X_S)\right) = H(Y) - H(Y|G = G_S, X = X_S).$$

$$H(Y|G{=}G_S, X{=}X_S) = -\mathbb{E}_{Y|G_S,X_S}\left[\log P_\Phi(Y|G{=}G_S, X{=}X_S)\right].$$

$$\min_M -\sum_{c=1}^{C} \mathbb{1}[y=c] \log P_\Phi(Y=y|G=A_c \odot \sigma(M), X=X_c),$$

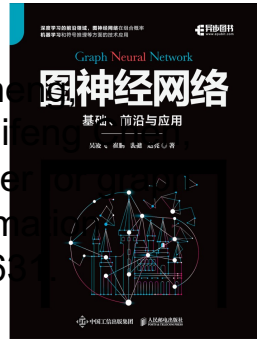


[Ying et al. 2019] Ying R, Bourgeois D, You J, Zitnik M, Leskovec J (2019) Gnnexplainer: Generating explanations for graph neural networks. Advances in neural information processing systems 32:9240

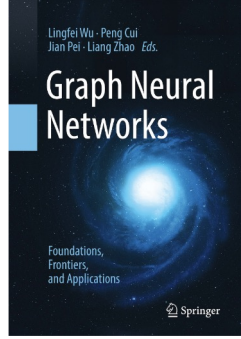
- PGExplainer [Luo et al, 2020]

$$M_{i,j} = \mathrm{MLP}_\Psi\left([\mathbf{z}_i; \mathbf{z}_j]\right),$$

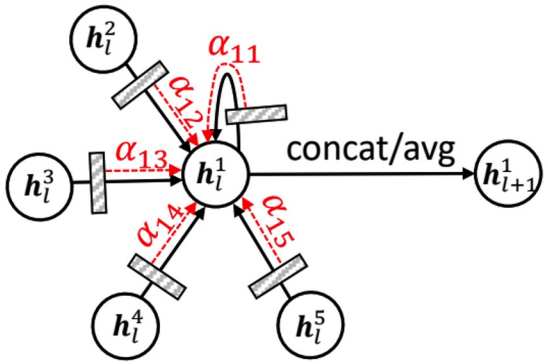
[Luo et al, 2020] Luo, Dongsheng, Wei Ch... Dongkuan Xu, Wenchao Yu, Bo Zong, Hai... and Xiang Zhang. "Parameterized explaine... neural network." Advances in neural inform... processing systems 33 (2020): 19620-196...

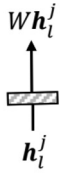


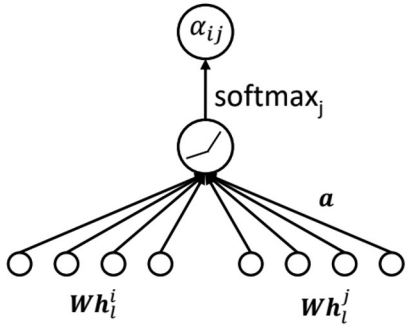
Interpretable Modeling on Graph Neural Networks



- GNN-Based Attention Models [Velčković et al. 2018]







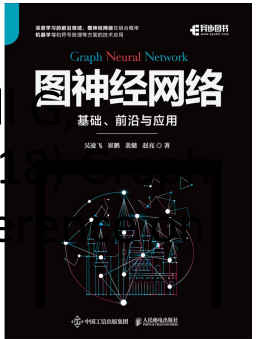
multi-head extension

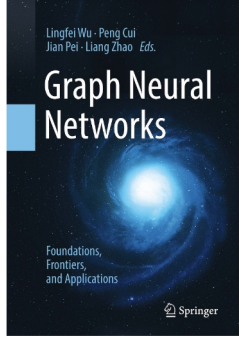
$$\mathbf{h}_{l+1}^i = \sigma\Big(\sum_{j\in\mathcal{V}_i\cup\{i\}} \alpha_{i,j} W\mathbf{h}_l^j\Big), \quad \Rightarrow \quad \mathbf{h}_{l+1}^i = \Big\|_{k=1}^{K} \sigma\Big(\sum_{j\in\mathcal{V}_i\cup\{i\}} \alpha_{i,j}^k W^k\mathbf{h}_l^j\Big),$$

$$\alpha_{i,j} = \mathrm{softmax}(e_{i,j}) = \frac{\exp(e_{i,j})}{\sum_{k\in\mathcal{V}_i\cup\{i\}}\exp(e_{i,k})},$$

$$e_{i,j} = \mathrm{LeakyReLU}(\mathbf{a}^\top[W\mathbf{h}_l^i \| W\mathbf{h}_l^j]),$$

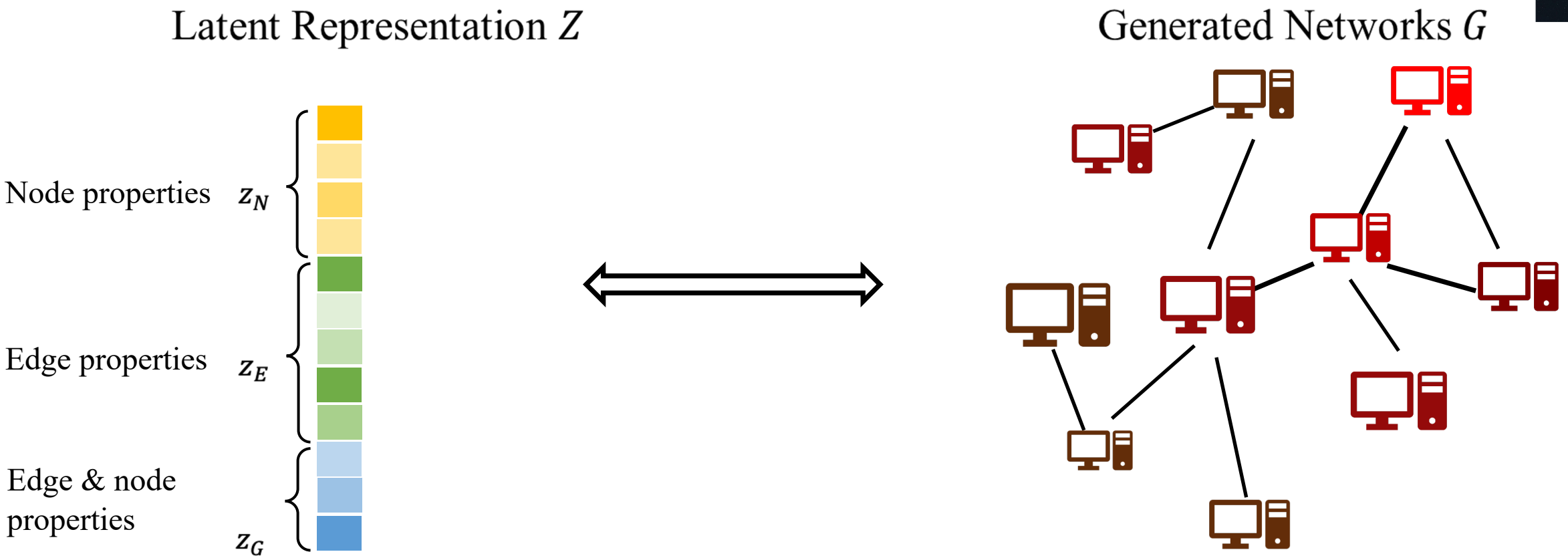
[Velčković et al. 2018] Velčković P, Cucurull G, Casanova A, Romero A, Lio P, Bengio Y (2018) Graph attention networks. In: International Conference on Learning Representations





Disentangled Representation Learning on Graphs

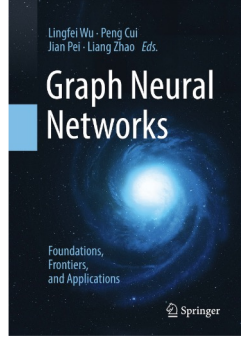
[Guo et al. 2020]



- **Goal:** Learning a graph generative model where each property of the generated graphs can be controlled by each latent variable.

[Guo et al. 2020] Xiaojie Guo, Liang Zhao, Zhao Qin, Lingfei Wu, Amarda Shehu, and Yanfang Ye. 2020. Interpretable Deep Graph Generation with Node-edge Codisentanglement. In Proceedings of the 26th KDD 2020,Aug 23-27, 2020.

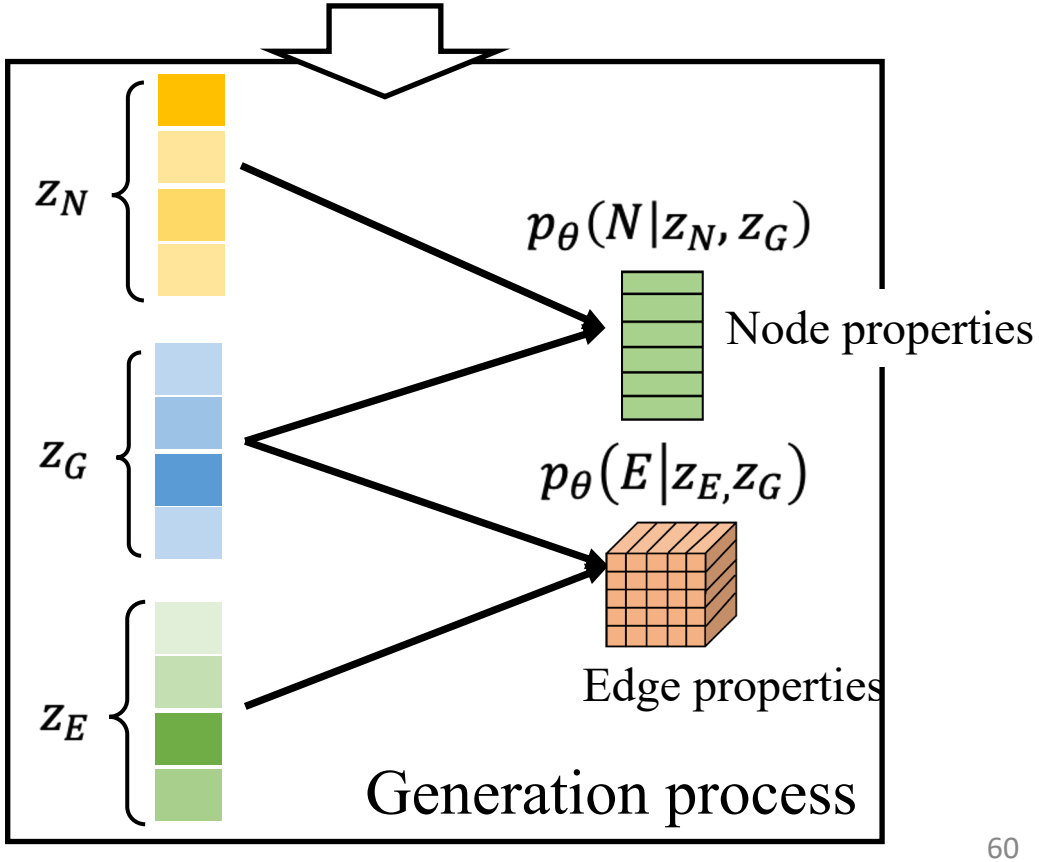
Node-edge Disentangled Variational Autoencoder



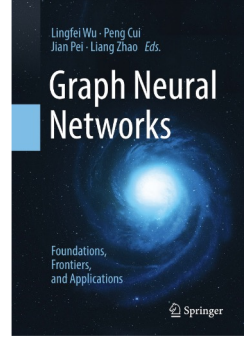
$$p_\theta(E, N, z_N, z_G, z_E)$$

$$p(z_N, z_G, z_E)\, p_\theta(E, N | z_N, z_G, z_E)$$

$$p(z_E)p(z_G)p(z_N)p_\theta(N | z_N, z_G)p_\theta(E | z_E, z_G)$$



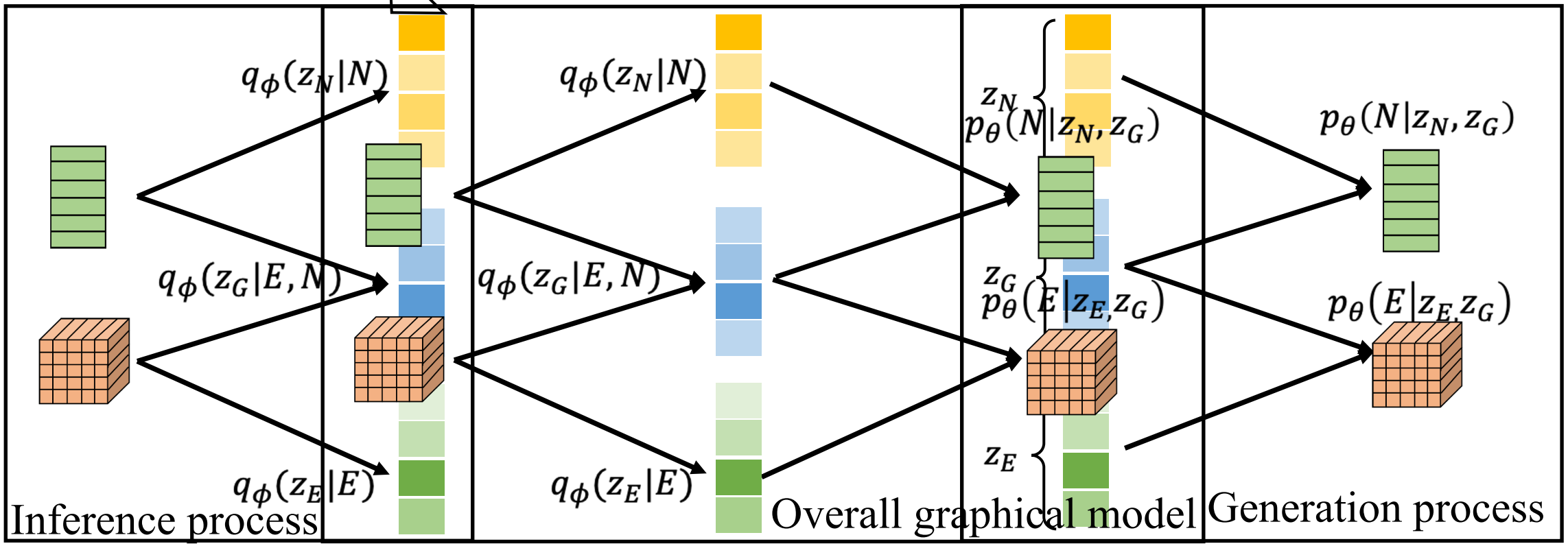
Node-edge Disentangled Variational Autoencoder



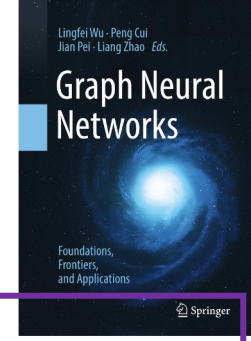
$$p(z_N) \approx q_\phi(z_N|N)$$
$$p(z_E) \approx q_\phi(z_E|E)$$
$$p(z_G) \approx q_\phi(z_G|E,N)$$

approximate

$$p_\theta(E,N,z_N,z_G,z_E)$$
$$p(z_N,z_G,z_E)\ p_\theta(E,N|z_N,z_G,z_E)$$
$$p(z_E)p(z_G)p(z_N)\ p_\theta(N|z_N,z_G)p_\theta(E|z_E,z_G)$$



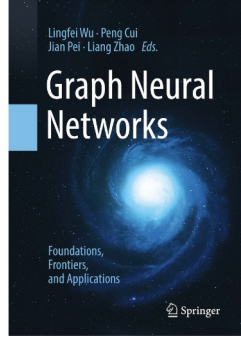
Node-edge Disentanglement VAE (NED-VAE)



[Guo et al. 2020]

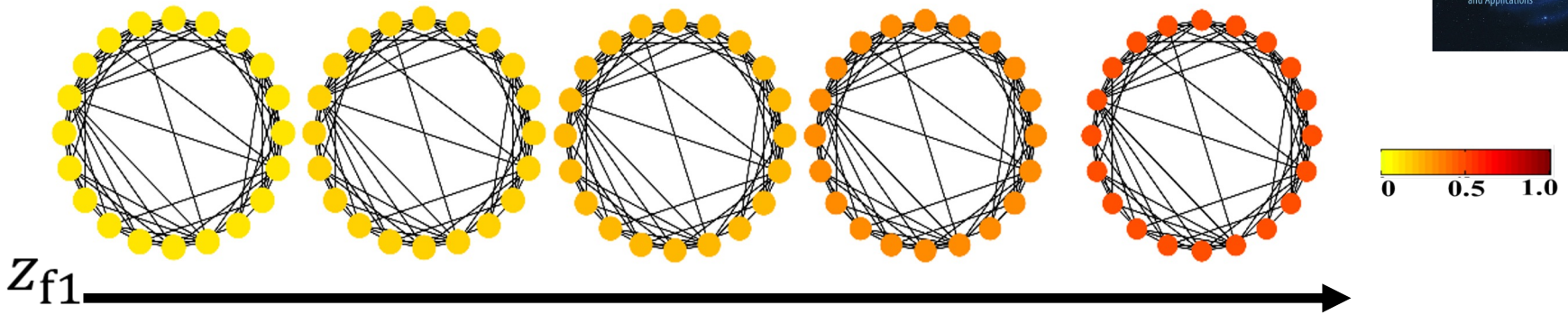
$$\max_{\theta,\phi} E_{q_\phi(Z|G)}[\log p_\theta(F|z_N, z_G)\, p_\theta(E|z_E, z_G)] - \beta D_{KL}(q_\phi(z_N|N) \| p(z_N)) - \beta D_{KL}(q_\phi(z_E|E) \| p(z_E)) - \beta D_{KL}(q_\phi(z_G|F,E) \| p(z_G))$$

Experiment: Qualitative evaluation (WS graphs)



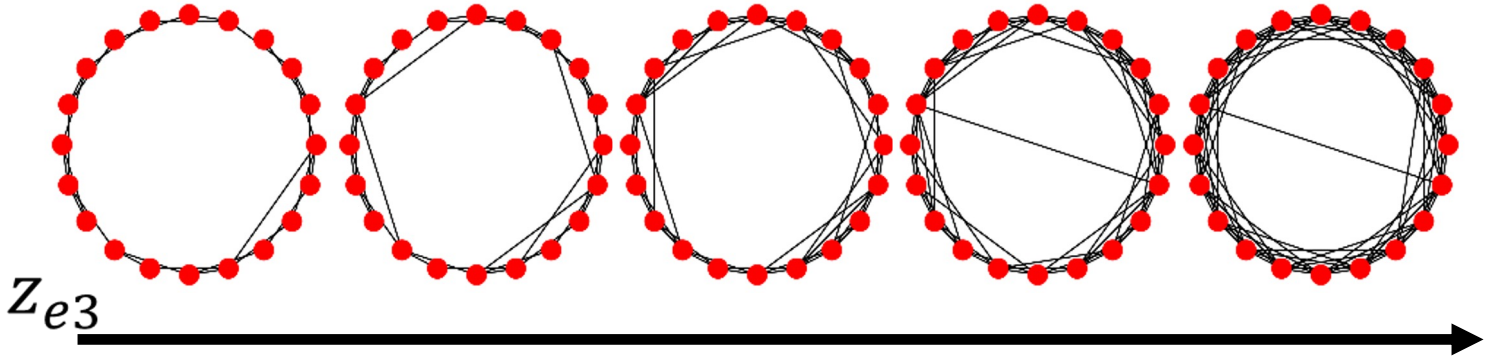
Node related factor

(node value)



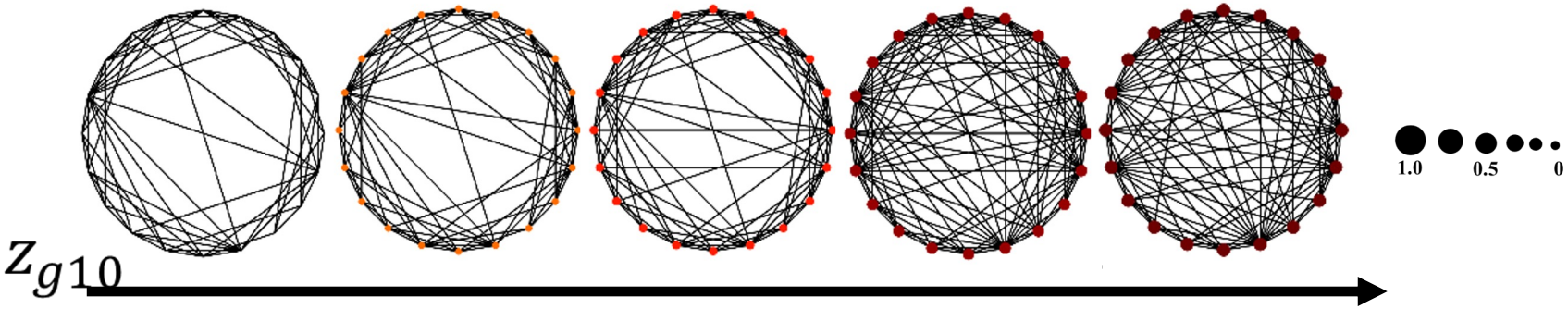
Edge related factor

(number of rings)

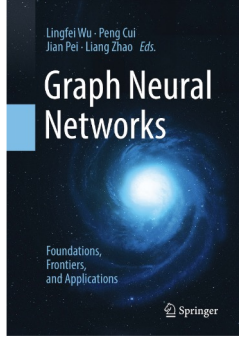


Node-edge-joint related factor

(density and node value)

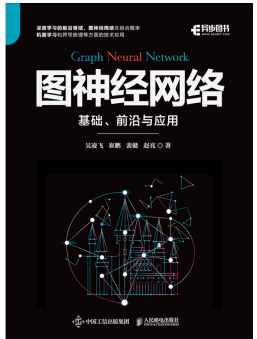


Evaluation of Graph Neural Networks

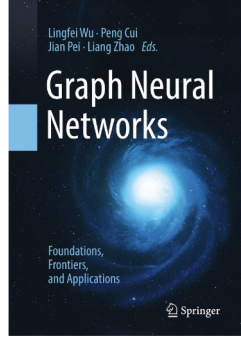


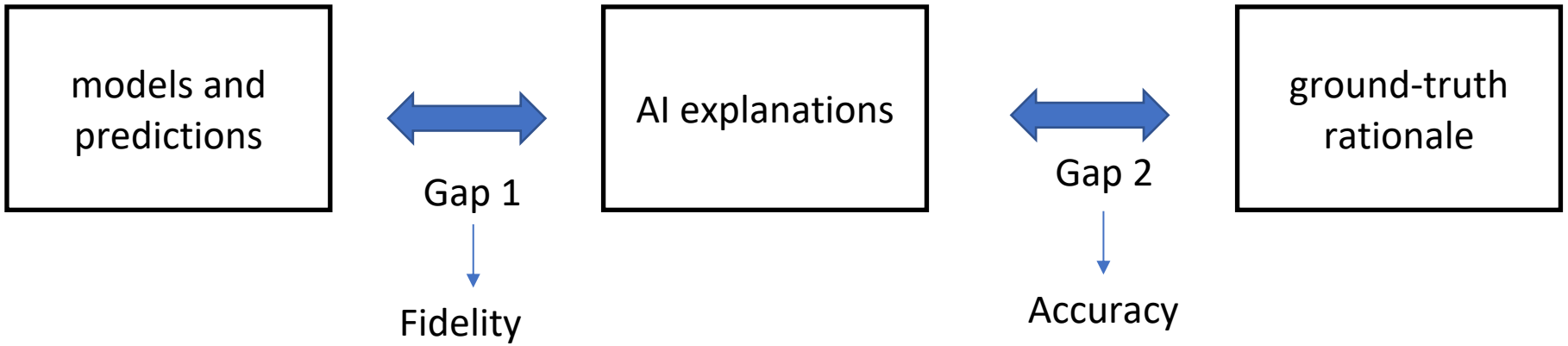
Explanations: Benchmark Datasets

- Synthetic Datasets
  - BA-Shapes (Ying et al, 2019)
  - BA-Community (Ying et al, 2019)
  - Tree-Cycle (Ying et al, 2019)
  - Tree-Grid (Ying et al, 2019)
  - Noisy BA-Community, Noisy Tree-Cycle, Noisy Tree-Grid (Lin et al, 2020a)
  - BA-2Motifs (Luo et al, 2020)
- Real-World Datasets
  - MUTAG (Debnath et al, 1991)
  - REDDIT-BINARY (Yanardag and Vishwanathan, 2015)
  - Delaney Solubility (Delaney, 2004)
  - Bitcoin-Alpha, Bitcoin-OTC (Kumar et al, 2016)
  - MNIST SuperPixel-Graph (Dwivedi et al, 2020)



Evaluation Metrics

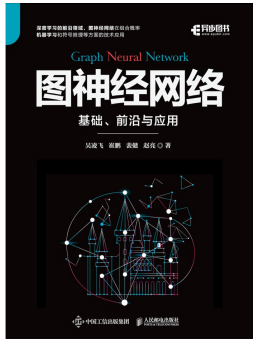




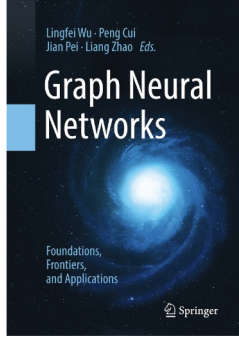
$$fidelity = \frac{1}{N}\sum_{i=1}^{N}\left(f^{y_i}\left(\mathcal{G}_i\right) - f^{y_i}\left(\mathcal{G}_i \setminus \mathcal{G}_i'\right)\right)$$

(Acc, AUC, F1, etc.)

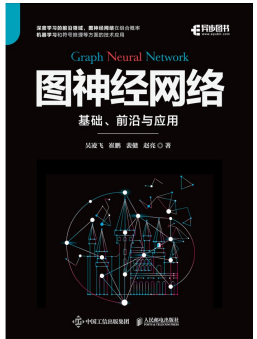
- Fidelity
- Accuracy
- Contrastivity: uses distance measure to quantify the differences between two explanations for different classes
- Sparsity: the ratio of explanation graph size to input graph size
- Stability: how stable the explanations are before and after adding noise to the input



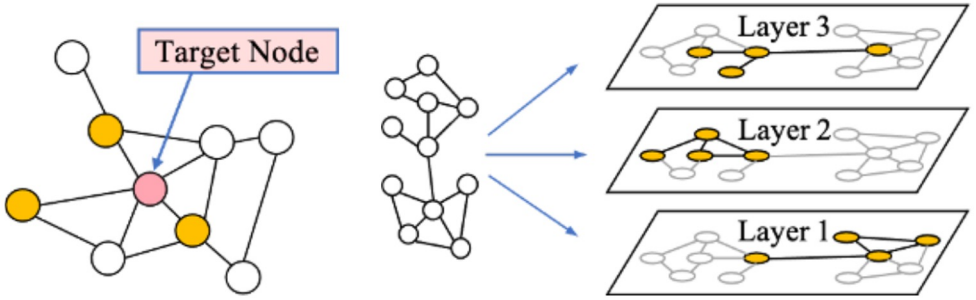
Future directions

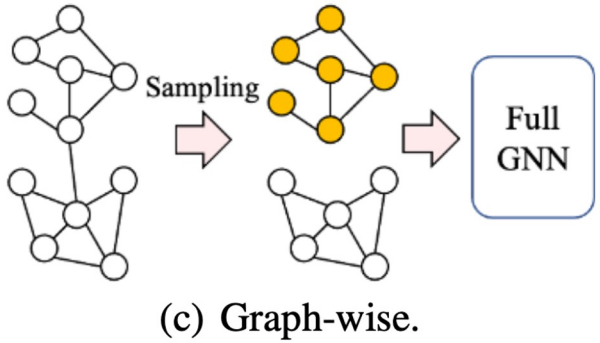


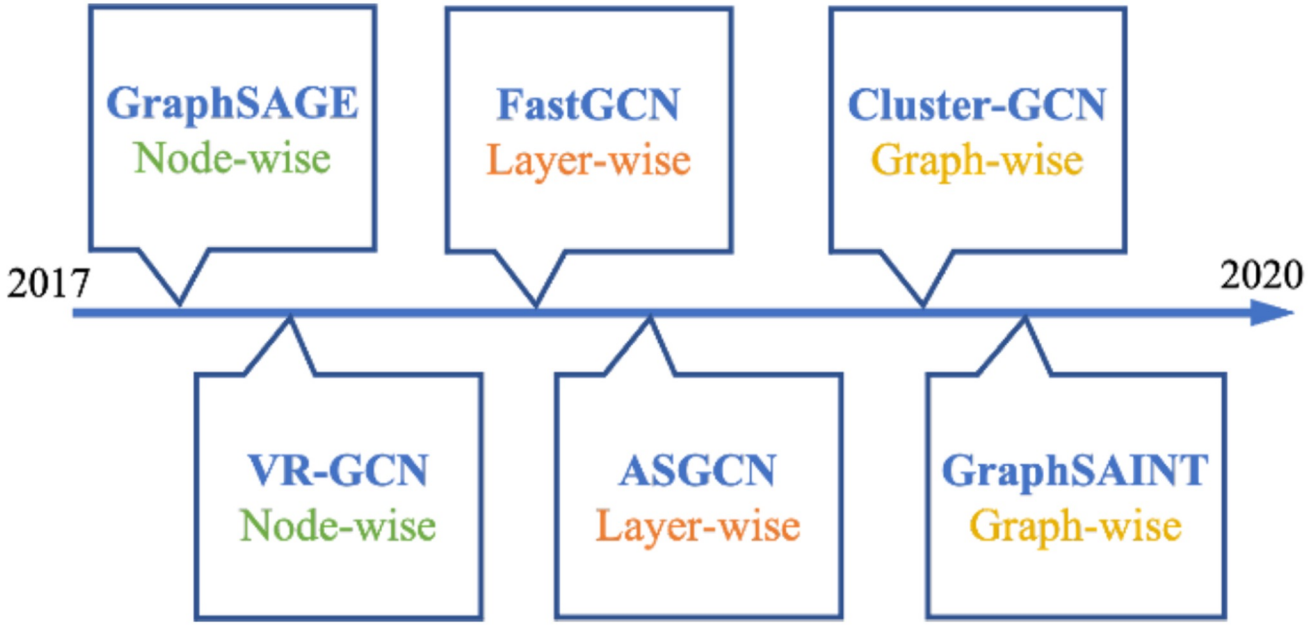
- more efficient explanation algorithms
- utilize interpretation towards identifying GNN model defects and improving model properties
- how to improve the explanation performance
- how to bring causality into GNN learning
- how to incorporate human-computer interaction (HCI) to show explanation in a more user-friendly format



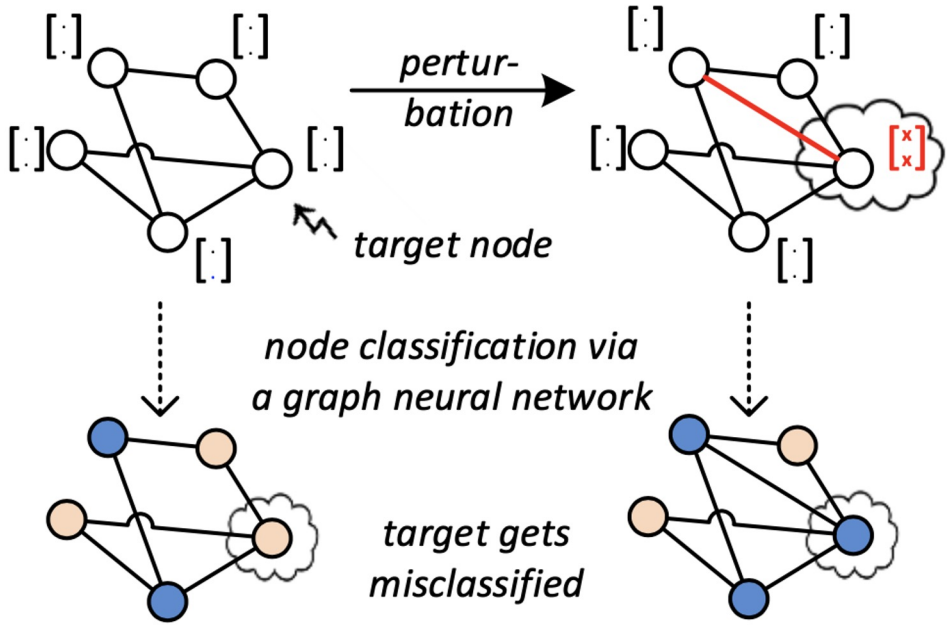
GNNs: Scalability (Chapter 6)

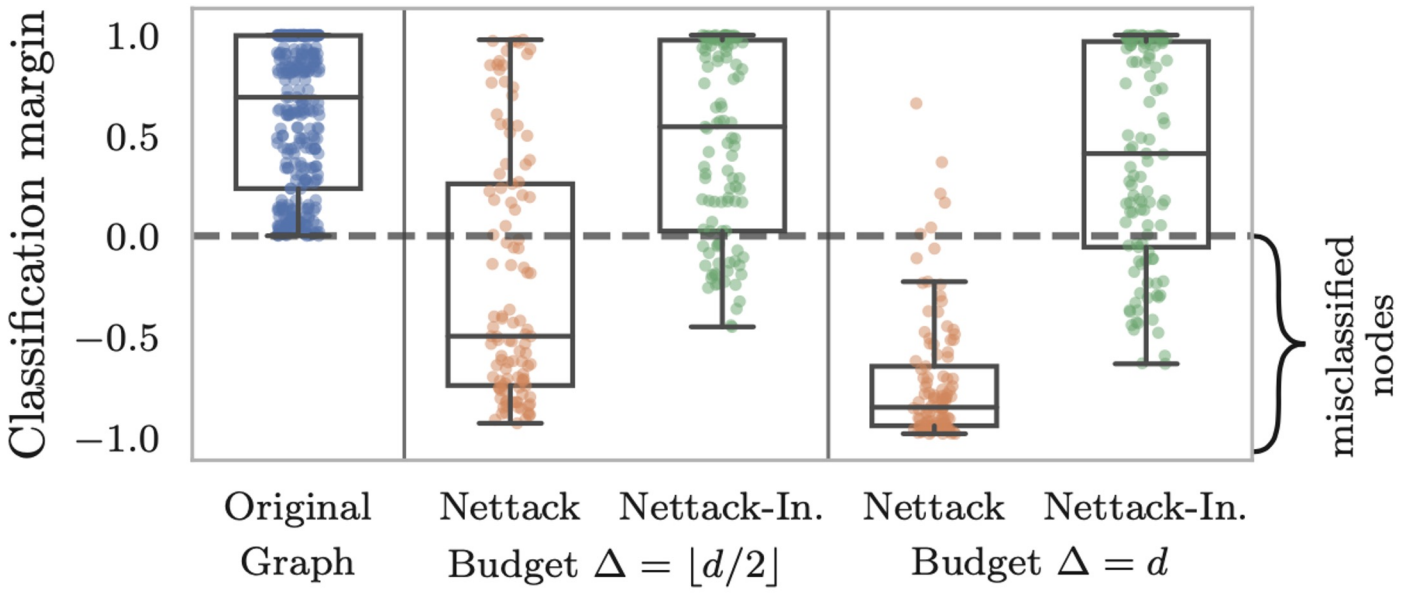






GNNs: Adversarial Robustness (Chapter 8)





Outline

GNNs Foundations

- Time History of GNNs
- GNNs: Foundations and Models
- GNNs: Theory, Scalability, Interpretability

GNNs Frontiers

- Graph Generation and Transformation
- Dynamic Graph Neural Networks
- Graph Matching
- Graph Structure Learning

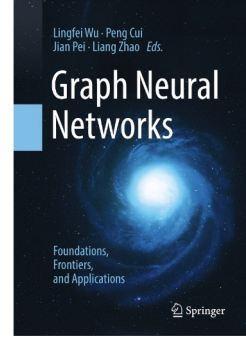
GNNs Applications

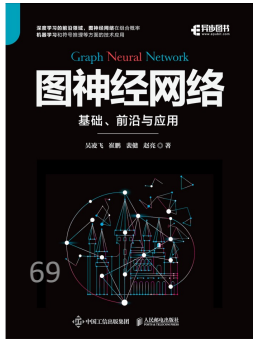
- GNNs in Recommendation
- GNNs in Natural Language Processing
- GNNs in Program Analysis
- GNNs in Protein Modeling

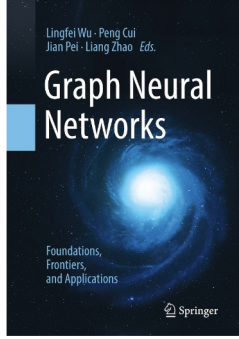
GNN book website：
https://graph-neural-networks.github.io/index.html

Amazon：
https://www.amazon.com/Graph-Neural-Networks-Foundations-Applications/dp/9811660530

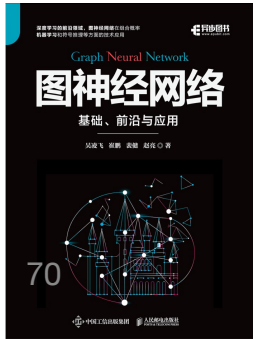
JD.com (京东商城)：
https://item.jd.com/13536841.html

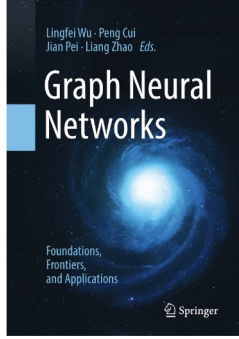




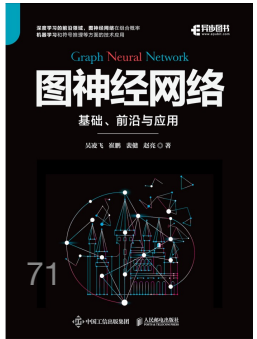


GNNs
Frontiers

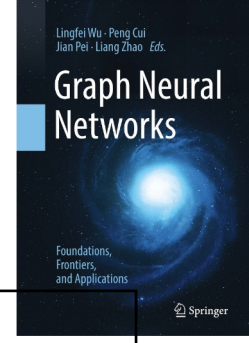




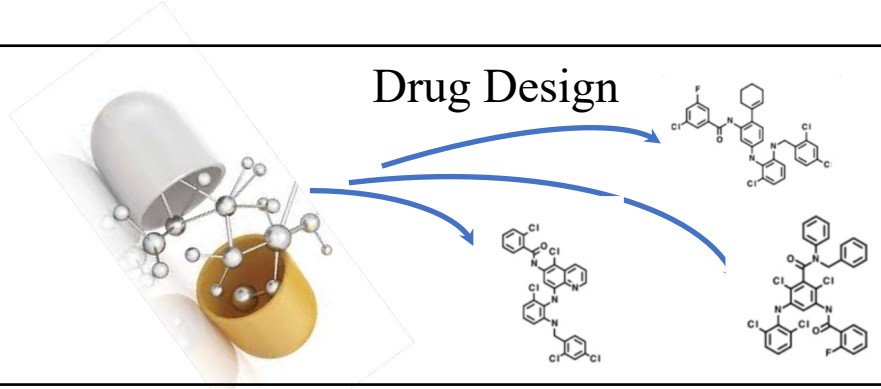
GNNs: Graph Generation & Transformation (Chapter 11 &12)



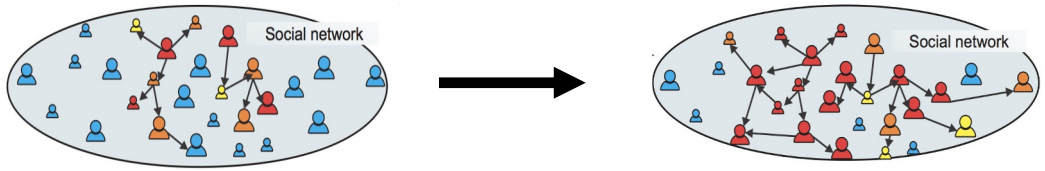
GNN for Graph Generation and Transformation



Drug Design



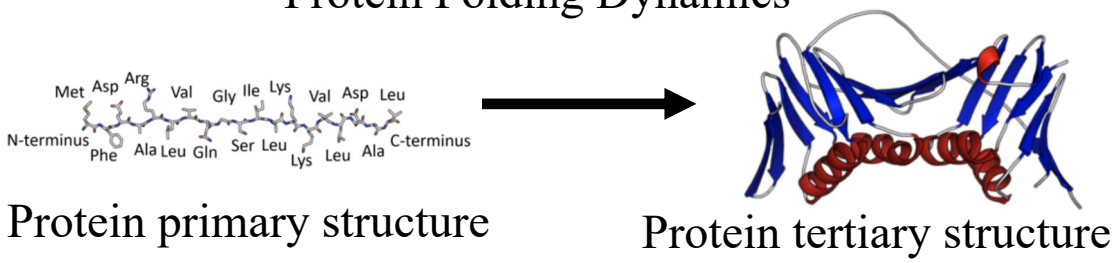
Social Network Synthesis



Without Social Event

With Social Event

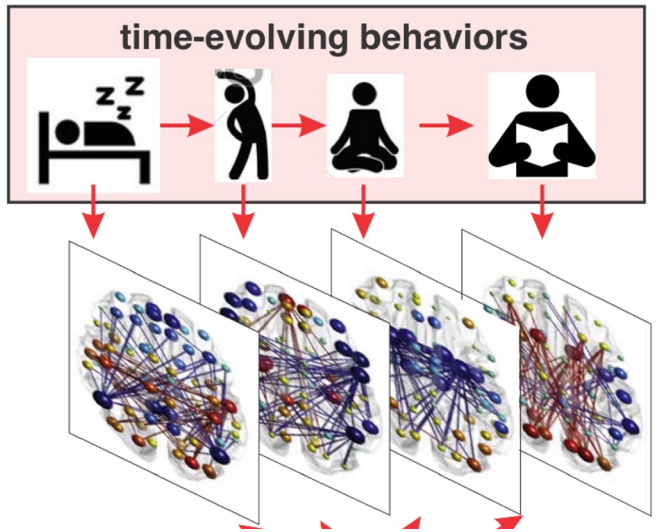
Protein Folding Dynamics



Protein primary structure

Protein tertiary structure

Brain Network Dynamics

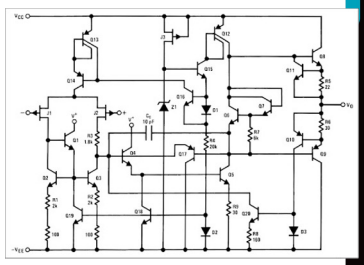


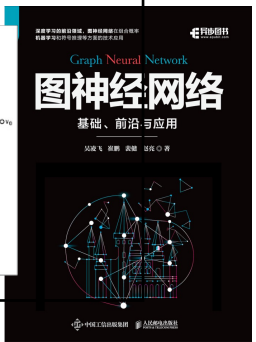
Integrated Circuit Design

A target function

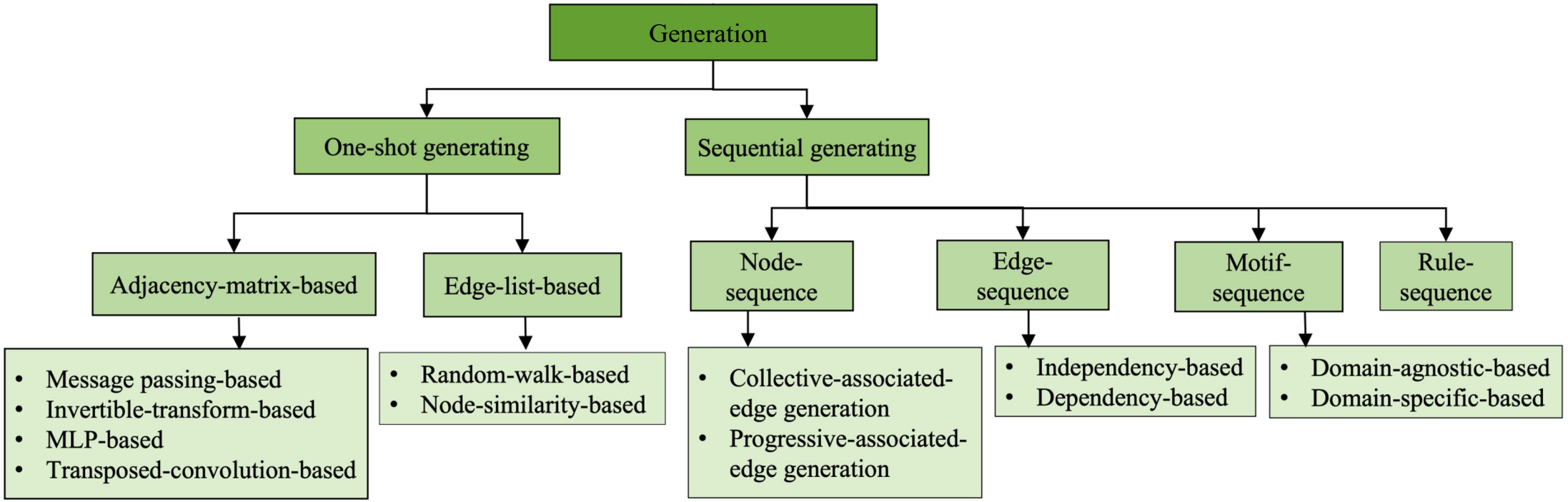
Low energy consumption

Small area

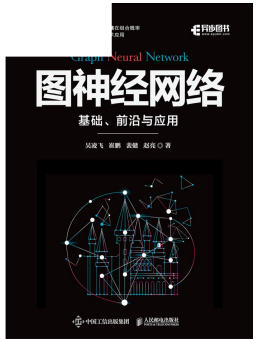




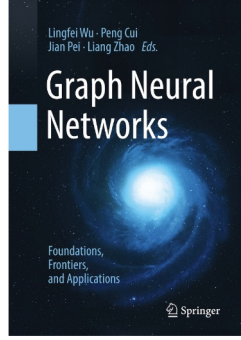
Taxonomy [Guo and Zhao 2022]



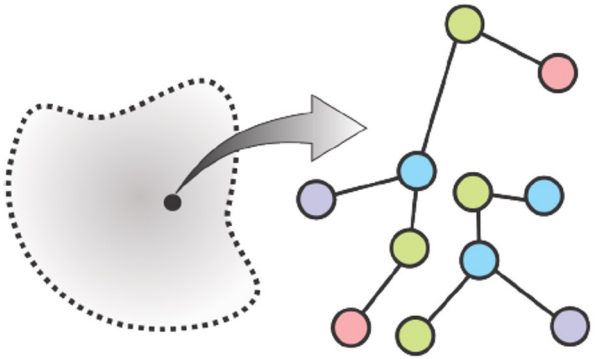
[Guo and Zhao 2022] Xiaojie Guo and Liang Zhao. 2020. A Systematic Survey on Deep Generative Models for Graph Generation. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI). Minor revision, 2022.



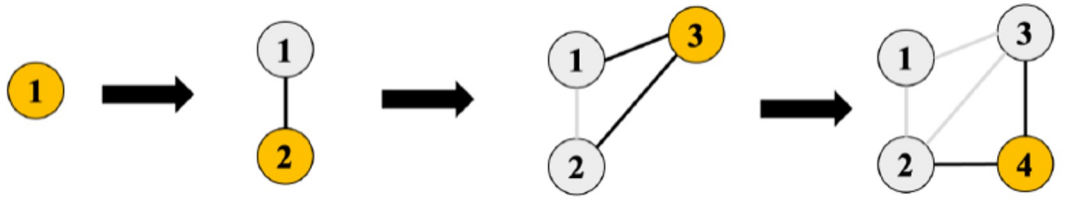
One-shot Generating vs Sequential Generating



One-shot Generation



Sequential Generating



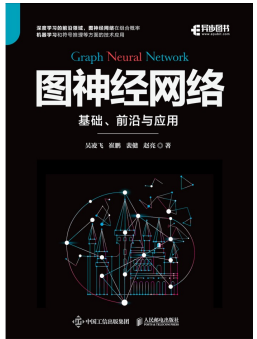
- Advantages and disadvantages

Pros:
capture global patterns
no need to order nodes

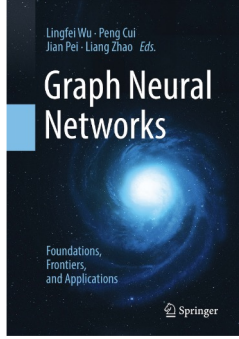
Cons:
time consuming

Pros:
efficient for each step
local patterns captured

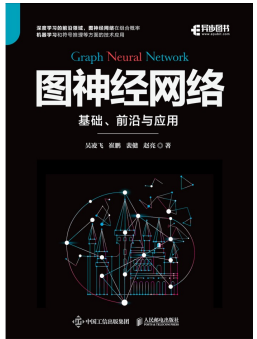
Cons:
hard to capture global patterns
predefined ordering of nodes

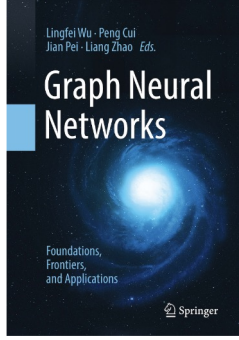


One-shot Generating



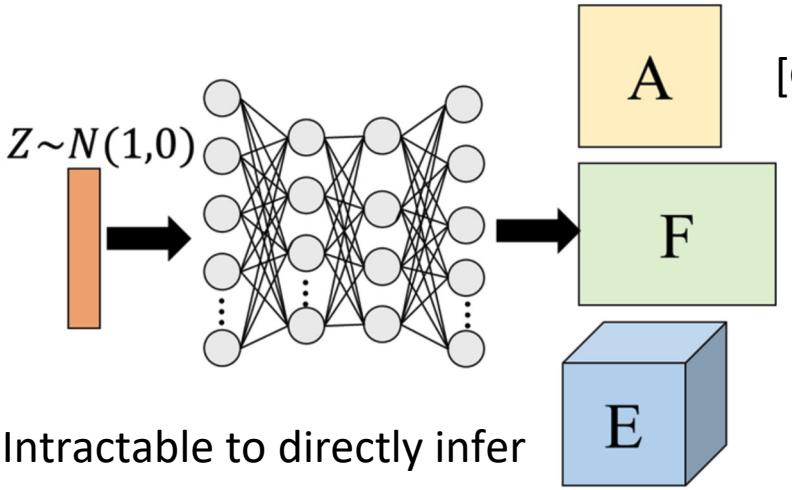
- Adjacency matrix-based
  - MLP-based decoder
  - Message-passing-based decoder
  - Invertible-transform-based decoder
- Edge list based
  - Random-walk-based
  - Node-similarity-based





Adjacency matrix-based: MLP as decoder

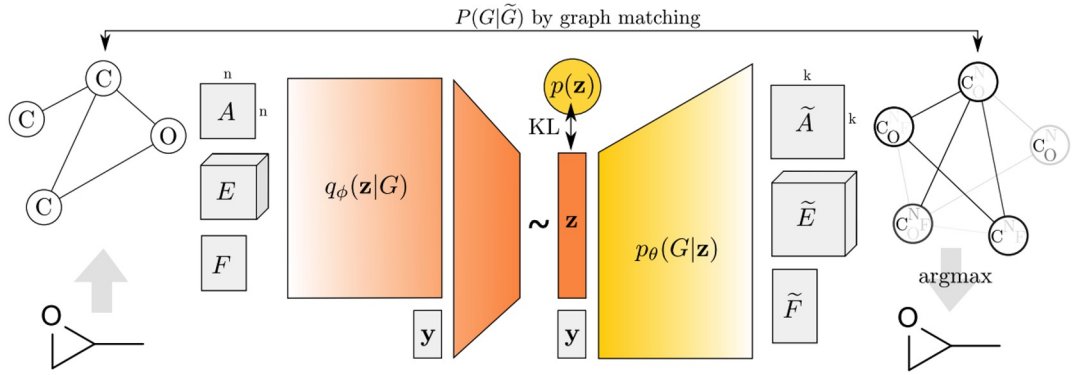
Goal: To learn the graph decoder as generator; typically use GNN as the encoder.



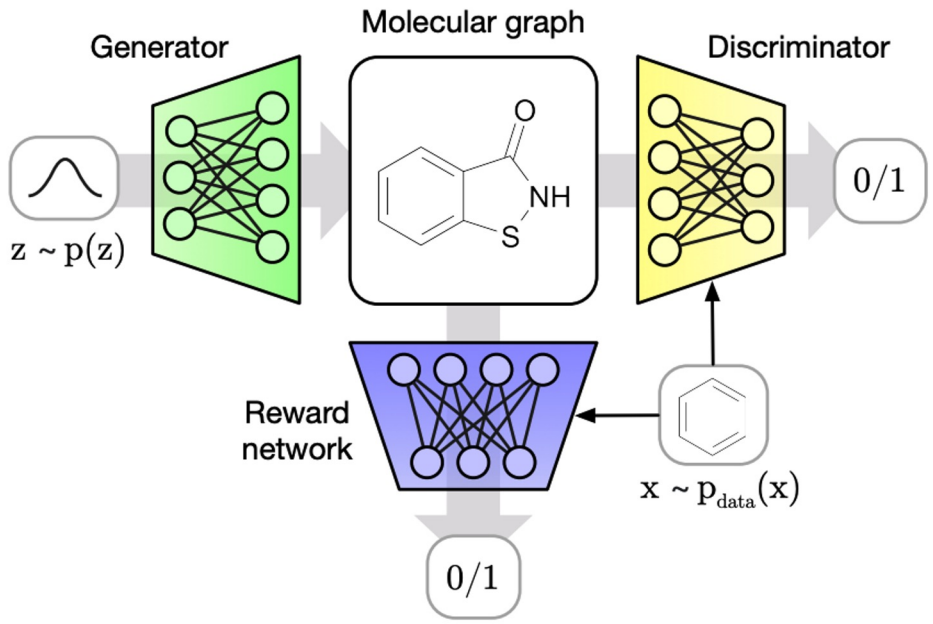
[Guo and Zhao 2022]

Intractable to directly infer

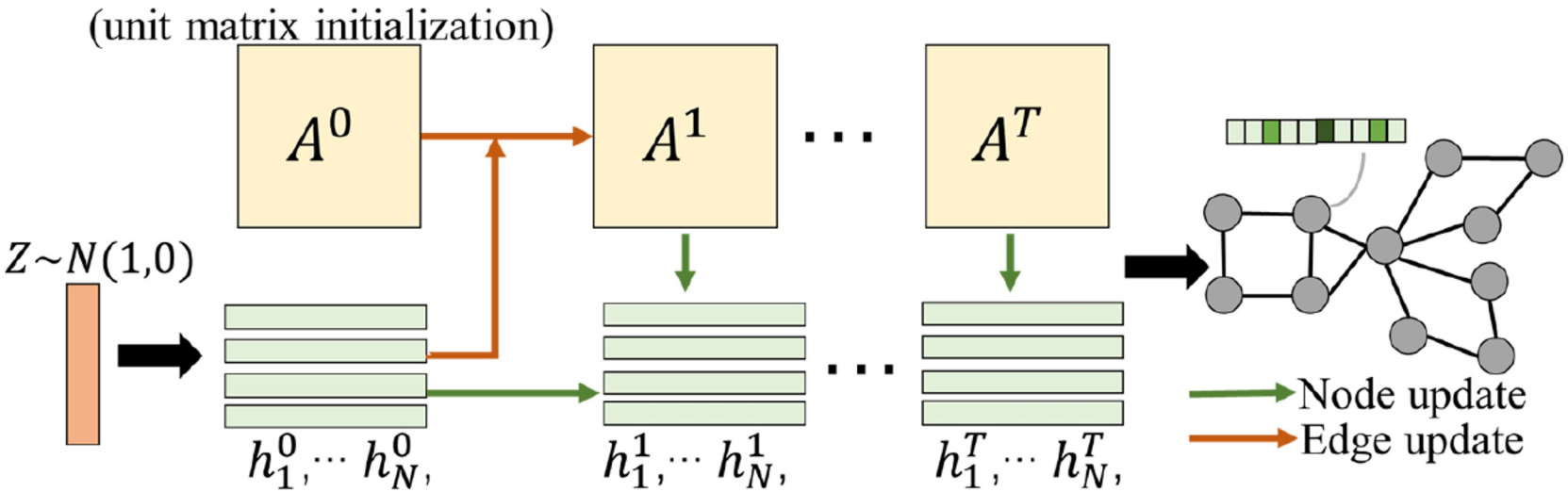
Variational inference
(e.g., GraphVAE [Simonovsky et al. 2018])



Adversarial training
(e.g., MolGAN [DeCao and Kipf 2018])

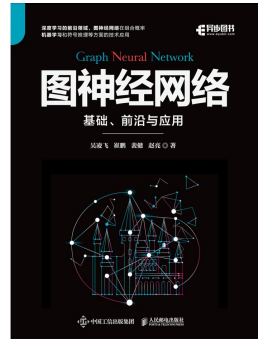


Adjacency matrix-based: Message-passing-based methods

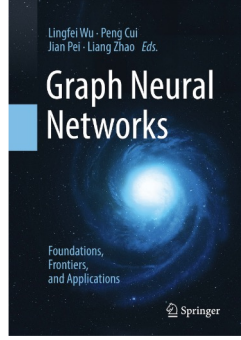


[Guo and Zhao 2022]

$$A_{i,j}^{l+1} = A_{i,j}^{l} + ReLu(\nu_1 A_{i,j}^{l} + \nu_2 h_i^l + \nu_3 h_j^l);$$
$$h_i^{l+1} = h_i^l + ReLu(w_1 h_i^l + \sum_j^N \eta_{i,j} w_2 h_j^l),$$

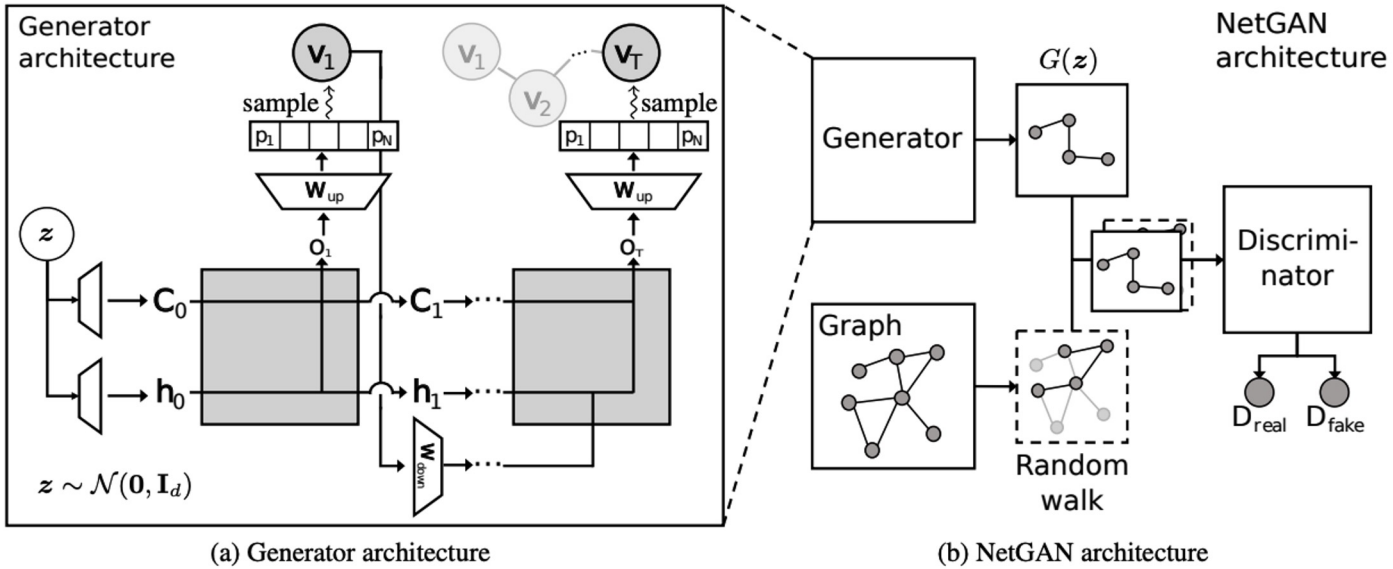


Edge-list based: Random-walk based



- NetGAN [Bojchevski 2018]

Phase 1: Generating random walks.



(a) Generator architecture (b) NetGAN architecture

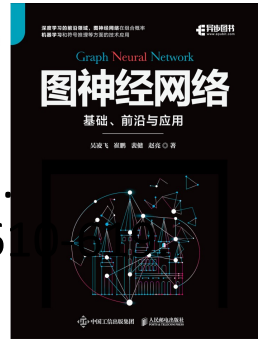
Phase 2: Assembling the Adjacency Matrix.

- Generate many random-walks
- Accumulate all into adjacency matrix
- To ensure no singletons:
  - Step 1: for each node, sample an edge

$$p_{ij} = \frac{s_{ij}}{\sum_v s_{iv}}$$

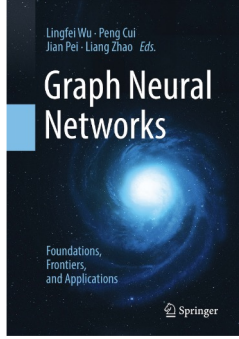
  - Step 2: sampling edges

$$p_{ij} = \frac{s_{ij}}{\sum_{u,v} s_{uv}}$$



•[Bojchevski 2018] Bojchevski, Aleksandar, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. "Netgan: Generating graphs via random walks." In International conference on machine learning, pp. 6[illegible] PMLR, 2018.

Edge-list based: Node-similarity-based



- Phase 1: Use GNN to learn the node embeddings.
- Phase 2: Generate each edge's probability $\tilde{A}_{i,j}$ based on pairs of nodes' embeddings

$$\tilde{A}_{i,j} = Sigmoid(Z_i Z_j^T) \quad \text{[Kipf et al. 2018]}$$

$$\tilde{A}_{i,j} = 1/(1 + exp(C(\| Z_i - Z_j \|_2^2 - 1))) \quad \text{[Liu et al. 2018]}$$

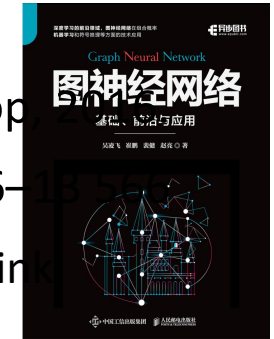
$$\tilde{A}_{i,j} = \text{Sigmoid}(m_j - log \| Z_i - Z_j \|_2^2) \quad \text{[Salha et al. 2019]}$$

- Phase 3: sample each edge using $\tilde{A}_{i,j}$

•[Kipf and Welling 2016] T. N. Kipf and M. Welling, "Variational graph auto-encoders," NeurIPS'2016 Workshop

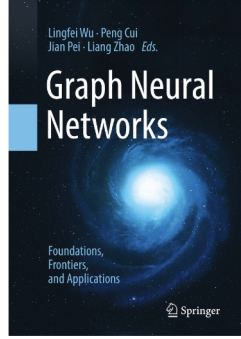
•[Liu et al. 2019] J. Liu, A. Kumar, and J. Ba et al, "Graph normalizing flows," in NeurIPS'2019, 2019, pp. 13 556–13 566.

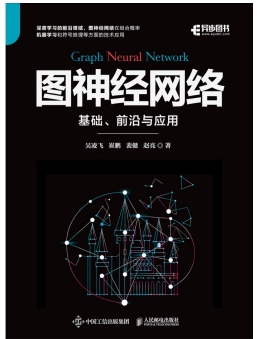
•[Salha et al. 2019] G. Salha, S. Limnios, and R. Hennequin et al, "Gravity-inspired graph autoencoders for directed link prediction," in CIKM'2019, 2019, pp. 589–598.



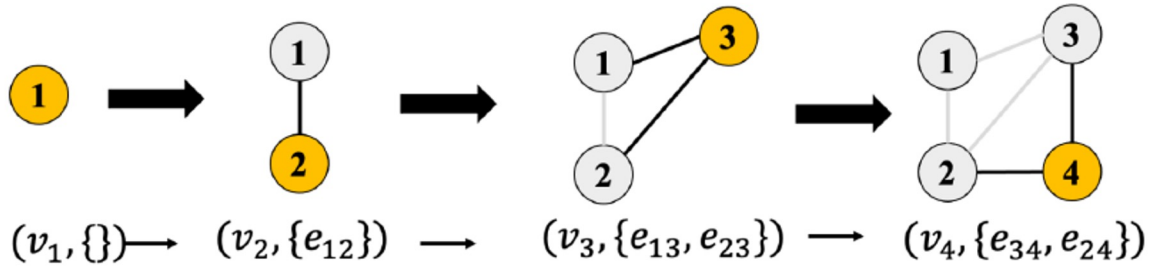
Sequential generating

- node sequence based
- edge sequence based
- graph motif sequence based
- rule-based (e.g., Junction Tree [Jin et al. 2018])





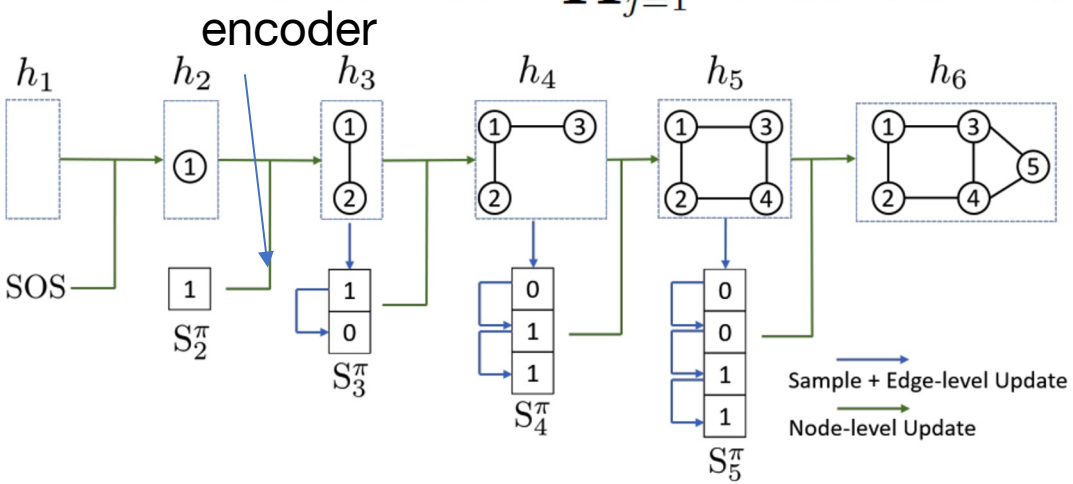
Node sequence based [You et al. 2018]



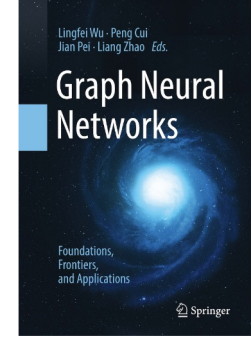
$$p(\mathcal{V}^\pi, A^\pi) = \prod_{i=1}^{N} p(v_i^\pi | v_{<i}^\pi, A_{<i,.}^\pi) p(A_{i,.}^\pi | v_{\leq i}^\pi, A_{<i,.}^\pi),$$

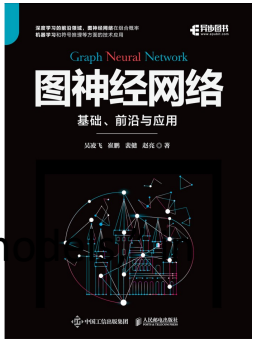
$$p(A_{i,.}^\pi | A_{<i,.}^\pi) = \prod_{j=1}^{i-1} p(A_{i,j}^\pi | A_{i,<j}^\pi, A_{<i,.}^\pi),$$

can be time-consuming for large graphs



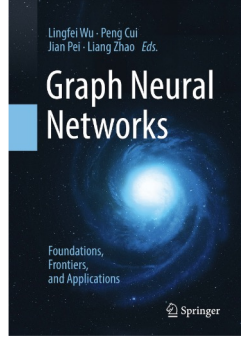
Ways to improve [Liu et al. 2018]
1) an addEdge function to determine the size of the edge set
2) a selectNode function to select the nodes to be connected from the existing graph



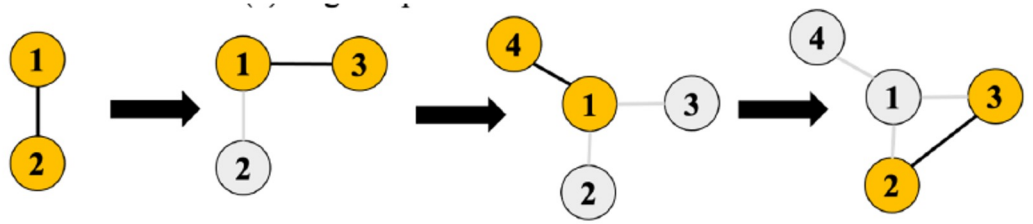


•[You et al. 2018] J. You, R. Ying, and X. Ren et al, "Graphrnn: generating realistic graphs with deep auto-regressive m
ICML'2018, 2018, pp. 5708–5717

Edge Sequence based



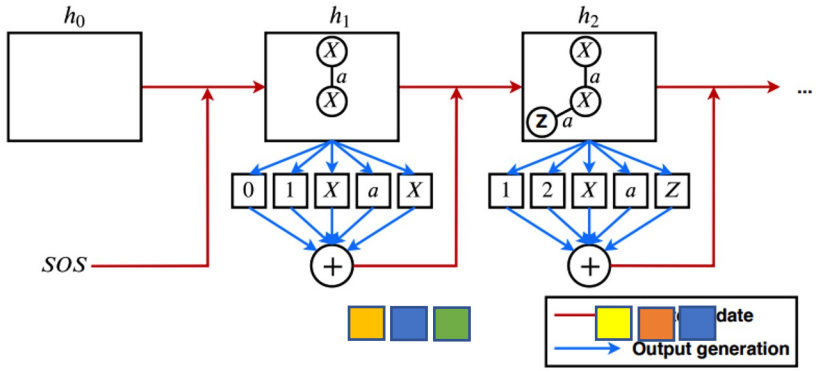
Sometimes edges contain rich information, and sometimes the graph is sparse.



$$(e_{12},\{v_1,v_2\}) \longrightarrow (e_{13},\{v_1,v_3\}) \longrightarrow (e_{14},\{v_1,v_4\}) \longrightarrow (e_{23},\{v_2,v_3\})$$

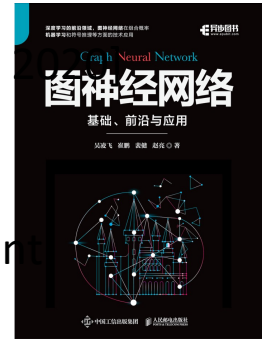
Consider graph as a sequence of tuples where each tuple is denoted as $s_i = (\alpha(u), \alpha(v), F_u, F_v, E^i_{u,v})$ [Goyal et al. 2020]

$$p(s_i|s_{<i}) = p((\alpha(u), \alpha(v), F_u, F_v, E^i_{u,v})|s_{<i})$$
$$= p(\alpha(u)|s_{<i})p(\alpha(v)|s_{<i})p(F_u|s_{<i})p(F_v|s_{<i})p(E^i_{u,v}|s_{<i}),$$



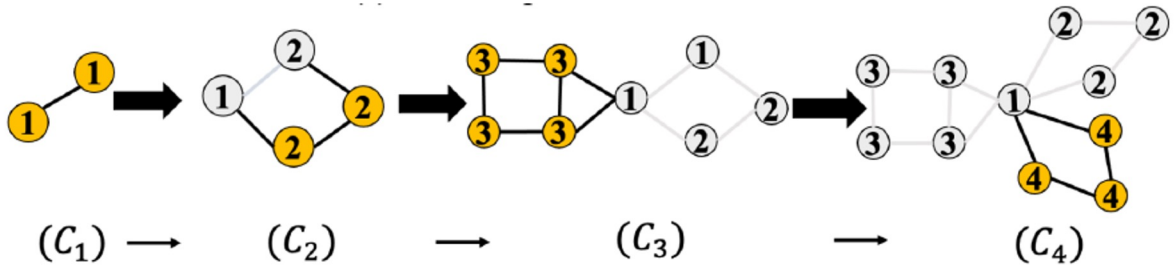
$$p(\alpha(u)|s_{<i})p(\alpha(v)|\alpha(u), s_{<i})$$

Consider the dependency between u and v. [Bacciu et al. 20

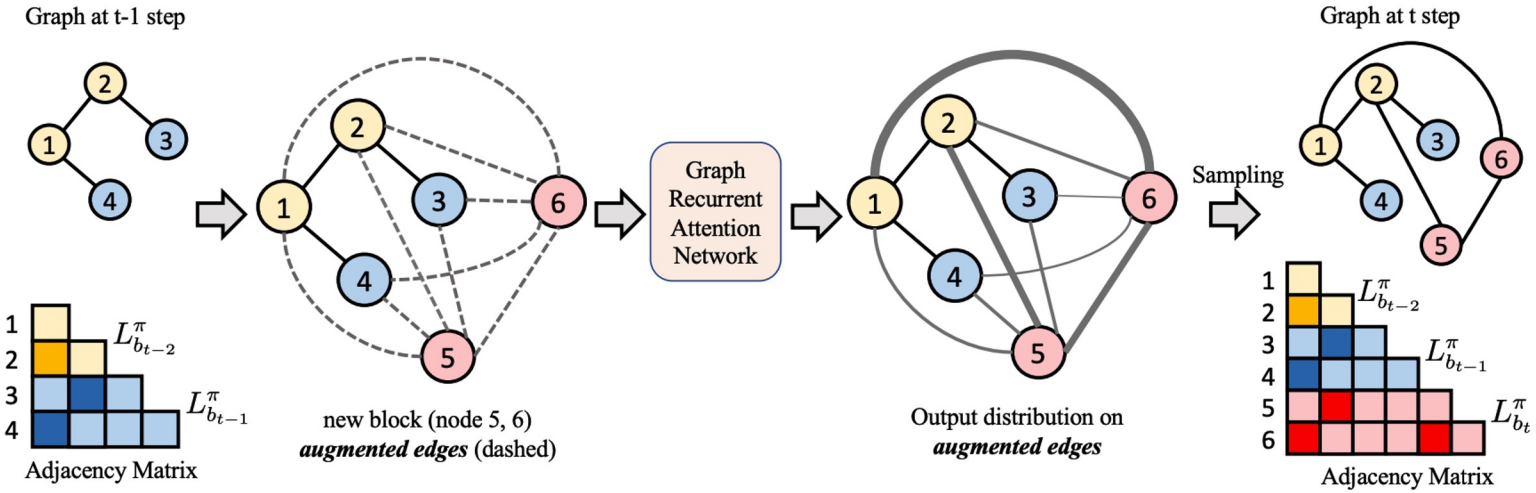


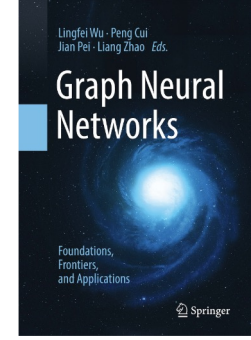
•[Bacciu et al. 2020] D. Bacciu, A. Micheli, and M. Podda, "Edge-based sequential graph generation with recurrent neural networks," Neurocomputing, 2020

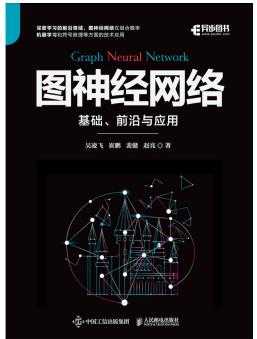
Graph Motif Based [Liao et al. 2019]



$$p(C_t|C_{<t}) = \prod_{B(t-1)<i\leq B} \prod_{1\leq j\leq i} p(A^{\pi}_{i,j}|C_{<t})$$

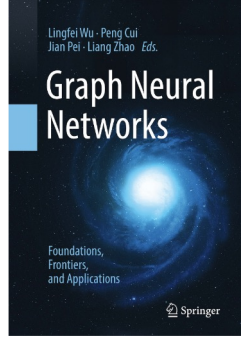






Graph Transformation

- Edge transformation
  - Applications: protein folding, structure-to-functional connectivity
  - Techniques:
    - GT-GAN [Guo et al. 2022b]
    - DAG2DAG [Sun and Li 2019]
- Node-edge joint transformation
  - Techniques:
    - GCPN [You et al. 2018]
    - NEC-DGT [Guo et al. 2019]

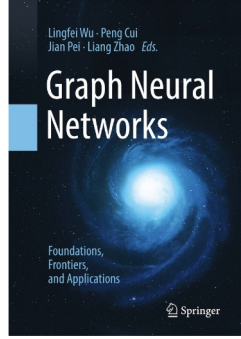


[You et al. 2018] You, Jiaxuan, Bowen Liu, Zhitao Ying, Vijay Pande, and Jure Leskovec. "Graph convolutional policy network for goal-directed molecular graph generation." Advances in neural information processing systems 31 (2018).

[Sun and Li 2019] Mingming Sun and Ping Li. 2019. Graph to graph: a topology aware approach for graph structures learning and generation. In AISTATS'2019. 2946–295

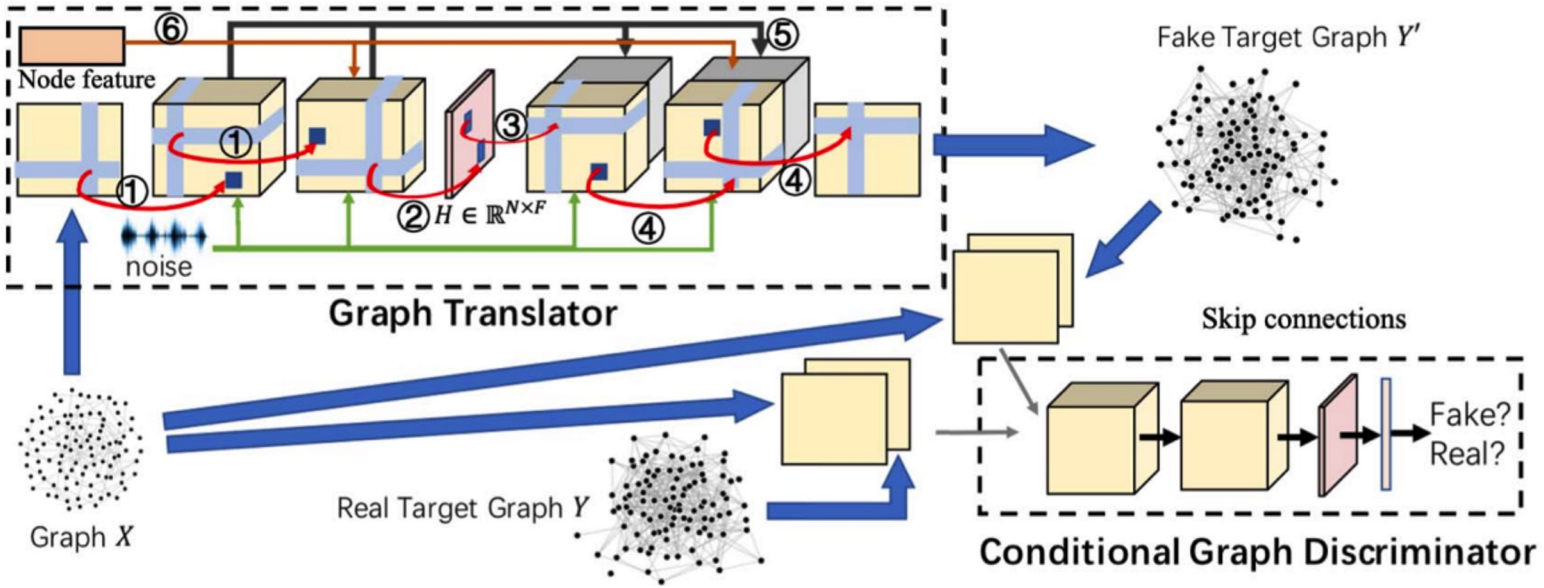
[Guo et al. 2022b] Xiaojie Guo, Lingfei Wu, Liang Zhao. 2022. Deep Graph Translation. IEEE Transactions on Neural Networks and Learning Systems (TNNLS), (Factor: 8.793), accepted.

[Guo et al. 2019] Xiaojie Guo, Liang Zhao, Cameron Nowzari, Setareh Rafatirad, Houman Homayoun, and Sai Dinakarrao. Deep Multi-attributed Graph Transla Edge Co-evolution. The 19th International Conference on Data Mining (ICDM 2019), long paper, , Beijing, China.



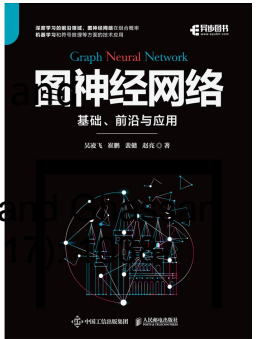
Graph Transformation

- Edge Transformation
  - GT-GAN [Guo et al. 2022], BrainnetCNN [Kawahara et al. 2017]



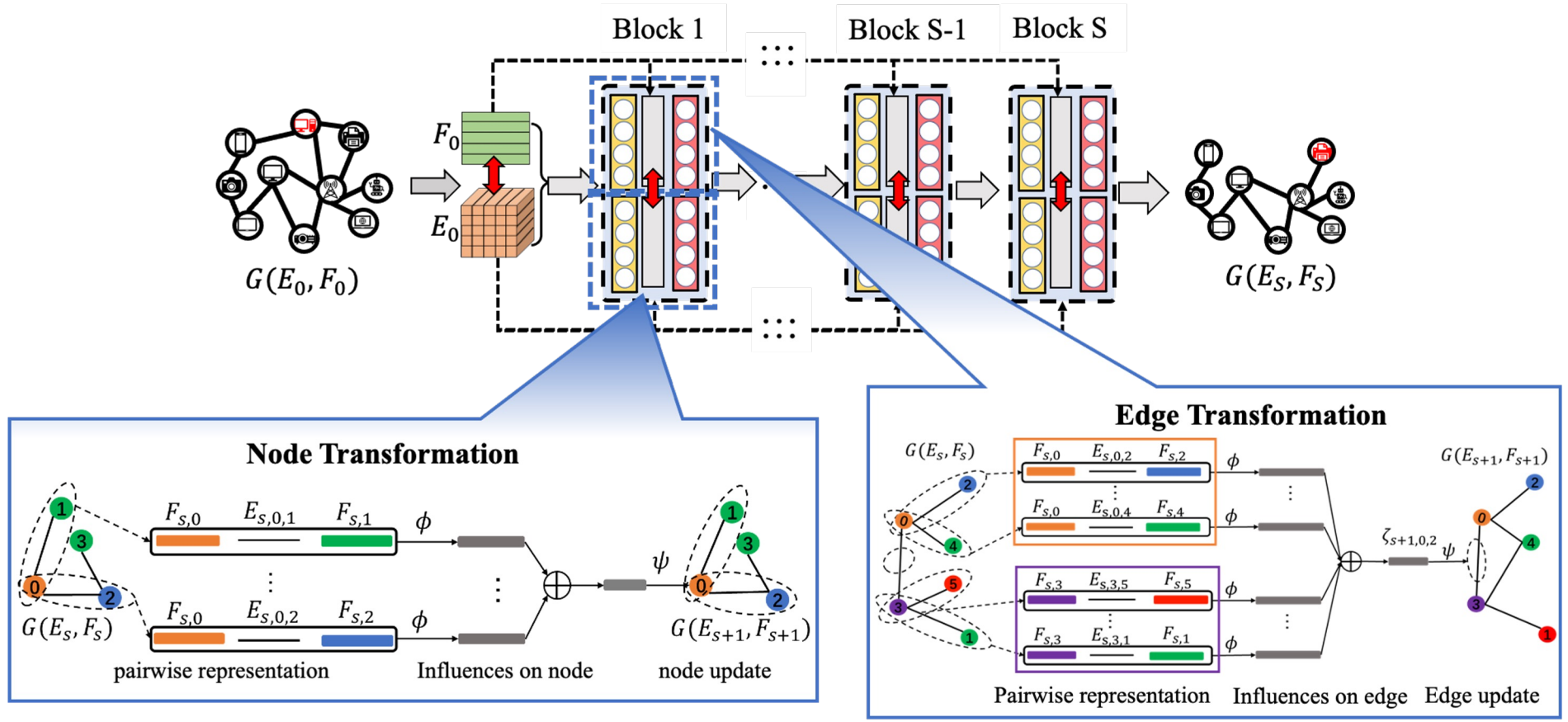
[Guo et al. 2022] Xiaojie Guo, Lingfei Wu, Liang Zhao. 2022. Deep Graph Translation. IEEE Transactions on Neural Networks and Learning Systems (TNNLS), (Impact Factor: 8.793), accepted.

[Kawahara et al. 2017] Kawahara, Jeremy, Colin J. Brown, Steven P. Miller, Brian G. Booth, Vann Chau, Ruth E. Grunau, Jill G. Zwicker, and Ghassan Hamarneh. "BrainNetCNN: Convolutional neural networks for brain networks; towards predicting neurodevelopment." NeuroImage 146 (2017): 1038-1049.

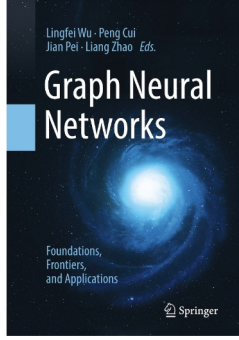


Graph Transformation

- Node-Edge-Co Transformation [Guo et al. 2019]

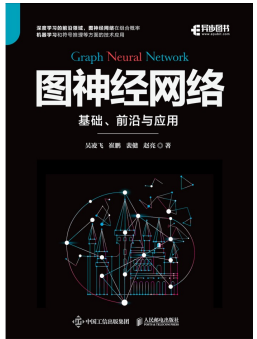


[Guo et al. 2019] Xiaojie Guo, Liang Zhao, Cameron Nowzari, Setareh Rafatirad, Houman Homayoun, and Sai Dinakarrao. Deep Multi-attrib
Graph Translation with Node-Edge Co-evolution. The 19th International Conference on Data Mining (ICDM 2019 Best Paper Award), Beijing



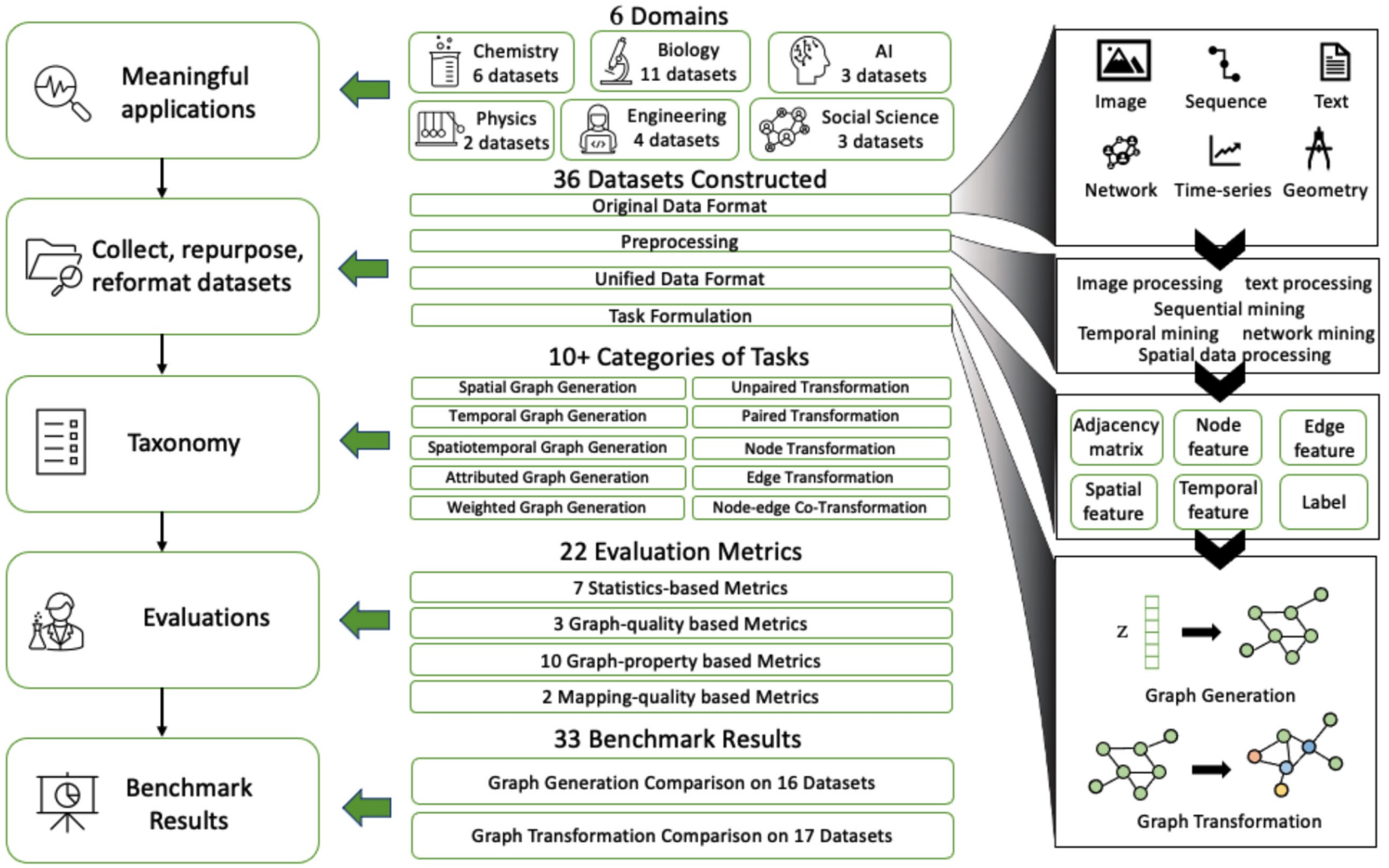
Evaluation metrics [Guo and Zhao 2022]

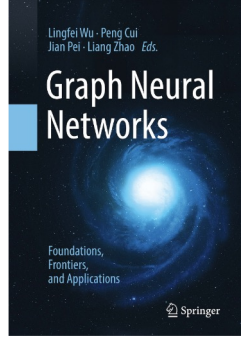
- Statistics-based
  - Difference between the distributions of generated graphs and real graphs
  - In the graph aspects
    - node degree, cluster coefficient, orbit count, triangle count, Characteristic path length, assortativity, largest component size, etc.
  - With difference measure:
    - Maximum Mean Discrepancy
    - Average KL Divergence
- Classification-based
  - A classifier is trained on a class of real graphs, then it is used to classify the generated graphs under the corresponding class.
  - Metrics: Accuracy and Fréchet Inception Distance
- Intrinsic-quality-based
  - Validity: the percentage of generated graphs that satisify required properties (e.g., cyclic, molecular validity, etc.)
  - Uniqueness: diversity of generated graphs
  - Novelty: percentage of the generated graphs unseen in training set



[Guo and Zhao 2022] Xiaojie Guo and Liang Zhao. 2020. A Systematic Survey on Deep Generative Models for Graph Generation. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI) 01 (2022): 1-20.

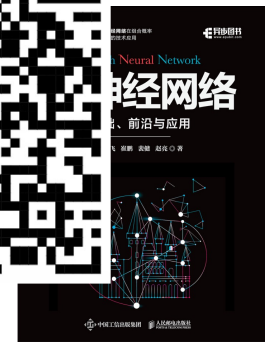
GraphGT: Benchmark Datasets Repository for Graph Generation [Du et al. 2021]



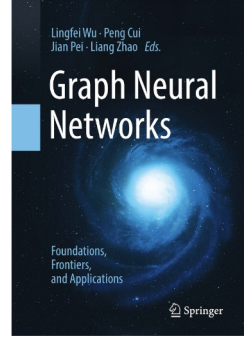


[Du et al. 2021] Du, et al. GraphGT: Machine Learning Datasets for Deep Graph Generation and Transformation. NeurIPS 2021.

Visit our website for datasets: https://graphgt.github.io/index.html
Or scan QR code:



GraphGT Benchmark datasets by domains



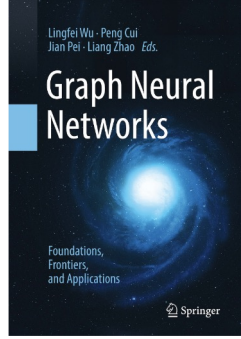
- **Biology**
  - **Protein**
    - Enzyme dataset
    - ProFold dataset
    - Protein dataset
  - **Brain network**
    - Brain-restingstate dataset
    - Brain-emotion dataset
    - Brain-gambling dataset
    - Brain-language dataset
    - Brain-motor dataset
    - Brain-relational dataset
    - Brain-social dataset
    - Brain-wm dataset
- **Engineering**
  - **Transportation**
    - METR-LA dataset
    - PeMS-BAY dataset
  - **ECE**
    - AuthNet dataset
    - IoTNet dataset
- **Social science**
  - **Social network**
    - CollabNet dataset
    - Ego dataset
    - TwitterNet dataset
- **Chemistry**
  - **Molecule**
    - ChEMBL dataset
    - ChemReact dataset
    - MolOpt dataset
    - MOSES dataset
    - QM9 dataset
    - ZINC250K dataset
- **Physics**
  - **Physical simulation**
    - N-body-charged dataset
    - N-body-spring dataset
- **AI**
  - **Vision**
    - CLEVR dataset
    - Skeleton (Kinectics) dataset
    - Skeleton (NTU) dataset
- **Others**
  - **Synthetic data**
    - Barab´asi-Albert Graphs dataset
    - Community dataset
    - Erdos-Renyi Graphs dataset
    - Scale-free dataset
    - Waxman Graphs dataset
    - Random Geometric dataset

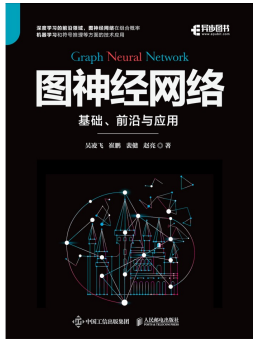
Du, et al. GraphGT: Machine Learning Datasets for Deep Graph Generation and Transformation. NeurIPS 2021.

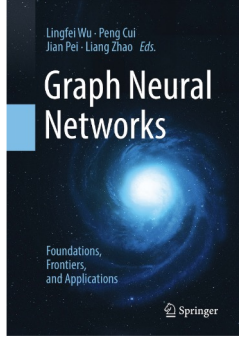
Visit our website for datasets: https://graphgt.github.io/index.html
Or scan QR code:

Future Opportunities

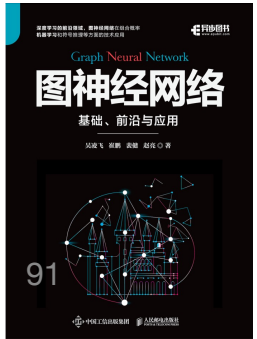
- Scalability.
- Validity constraint.
- Interpretability and Controllability.



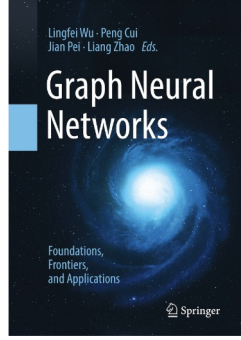




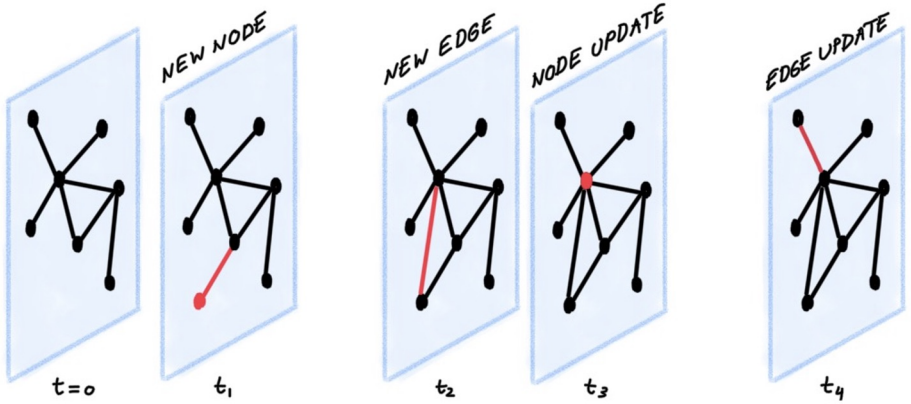
GNNs: Dynamic Graph (Chapter 15)



GNN for Dynamic graph: dynamic graph types



- Discrete-time dynamic graph (DTDG)

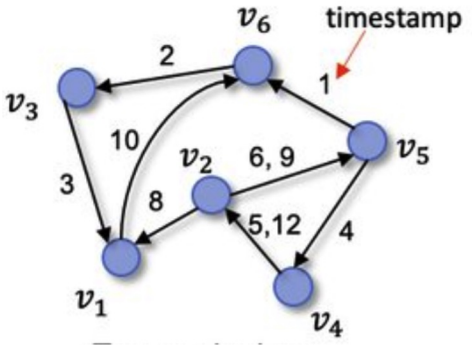


[Rossi 2020]

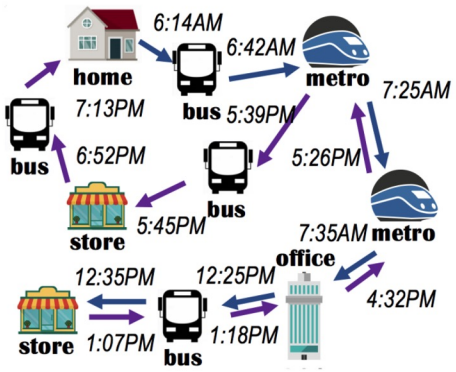
$$[G^{(1)}, G^{(2)}, \ldots, G^{(\tau)}]$$

$$G^{(t)} = (V^{(t)}, A^{(t)}, X^{(t)})$$

- Continuous-time dynamic graph (CTDG)



[Xu 2020]

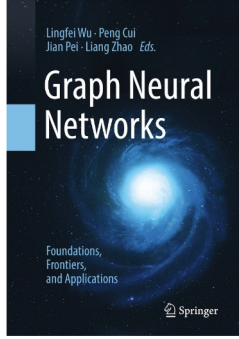


[Zhang et al. 2021]

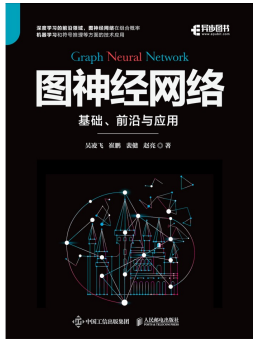
$$(G^{(t_0)}, O)$$

$$O = [(add\ node, v_4, 20\text{-}05\text{-}2020), (add\ edge, (v_2, v_4), 21\text{-}05\text{-}2020),$$
$$(Feature\ update, (v_1, [0.1, 2]), 28\text{-}05\text{-}2020), (add\ edge, (v_3, v_4), 04\text{-}06\text{-}2020)]$$

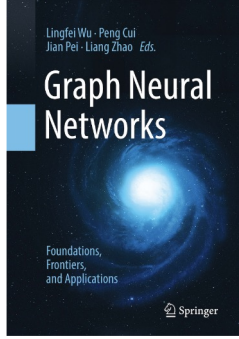
GNN for Dynamic graph: types of dynamics



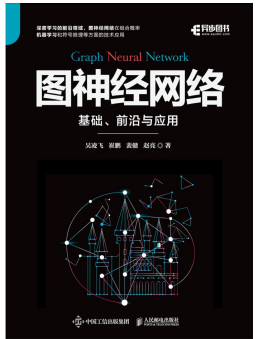
- Operations of graph dynamics:
  - Node addition/deletion
  - Feature update
  - edge addition/deletion
  - edge weight updates
- Another categorization of graph dynamics:
  - communication: dynamics on graphs. usually fast
  - association: dynamics of graphs. usually slower.



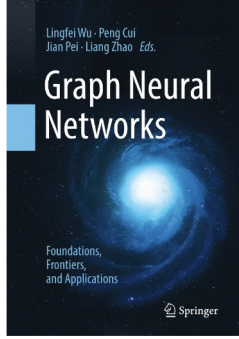
Types of tasks



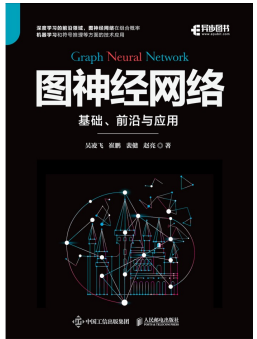
- Dynamic Node classification/regression
  - interpolation
  - extrapolation
- Dynamic graph classification
- Dynamic link prediction
  - interpolation
  - extrapolation
- Time prediction
  - interpolation
  - extrapolation



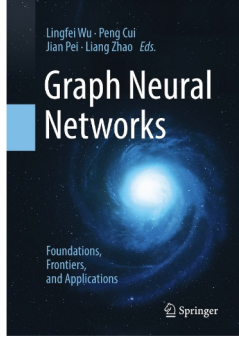
Modeling Dynamic Graphs with GNN



- Conversion to Static Graph [Yao et al. 2016]
- GNNs for DTDGs [Seo et al, 2018; Manessi et al, 2020; Xu et al, 2019a]
- GNNs for CTDGs [Kumar et al. 2019b]



Conversion to Static Graph



- Temporal aggregation [Yao et al. 2016]

Involving topology

weighted aggregation of the adjacency matrices

exponentially decaying $\boldsymbol{A}^{(agg)} = \sum_{t=1}^{\tau} \phi(t,\tau)\boldsymbol{A}^{(t)}$

$$\boldsymbol{X}^{(agg)} = \sum_{t=1}^{\tau} \phi(t,\tau)\boldsymbol{X}^{(t)}$$

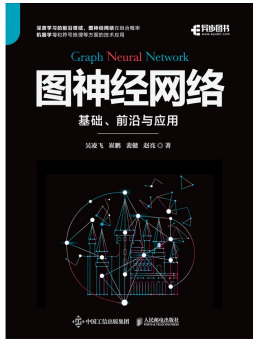
For extrapolation problems

$$\phi(t,\tau) = \exp(-\theta(\tau - t))$$

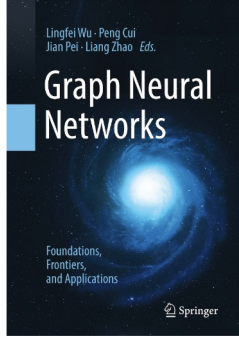
For interpolation problems

$$\phi(t,t') = \exp(-\theta|t' - t|)$$

θ is a hyperparameter controlling how fast the importance decays



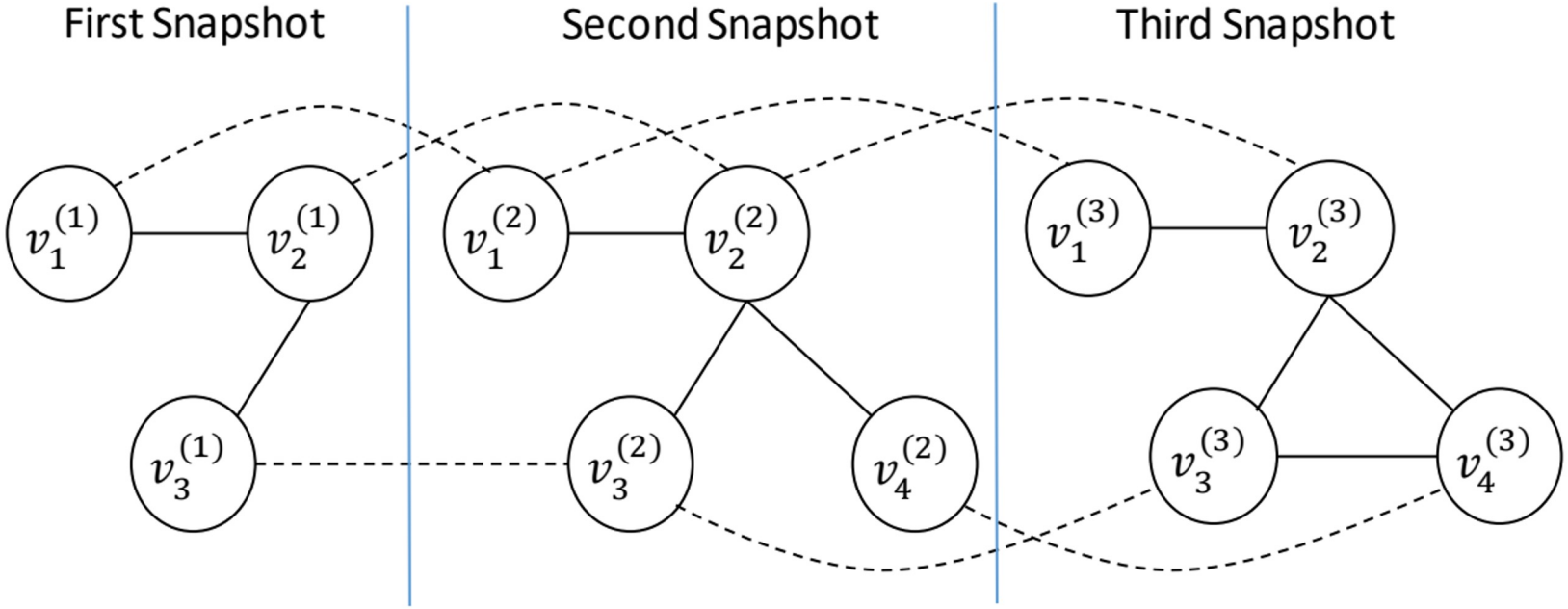
Conversion to Static Graph



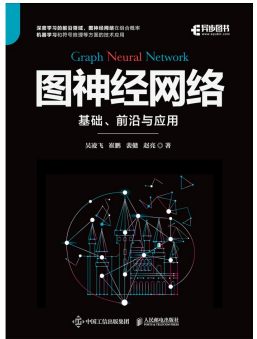
- Temporal unrolling

  Connecting the nodes corresponding to the same object across time

GNN(



)

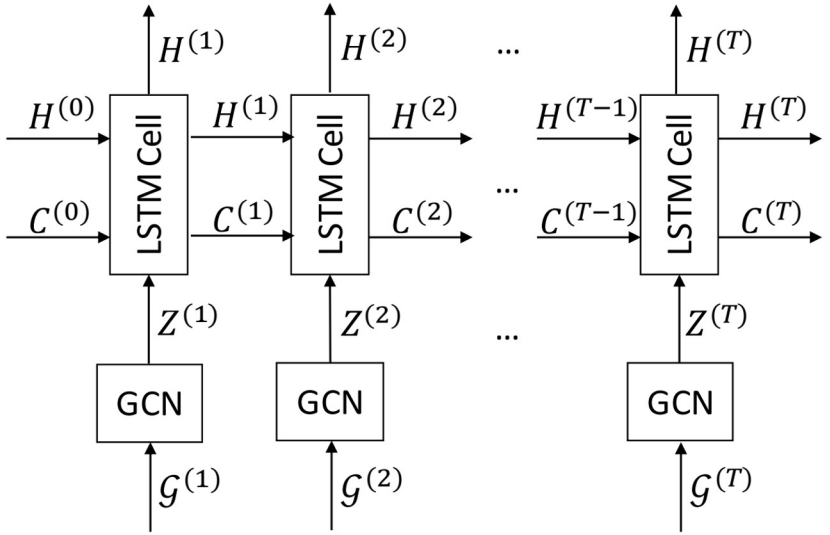


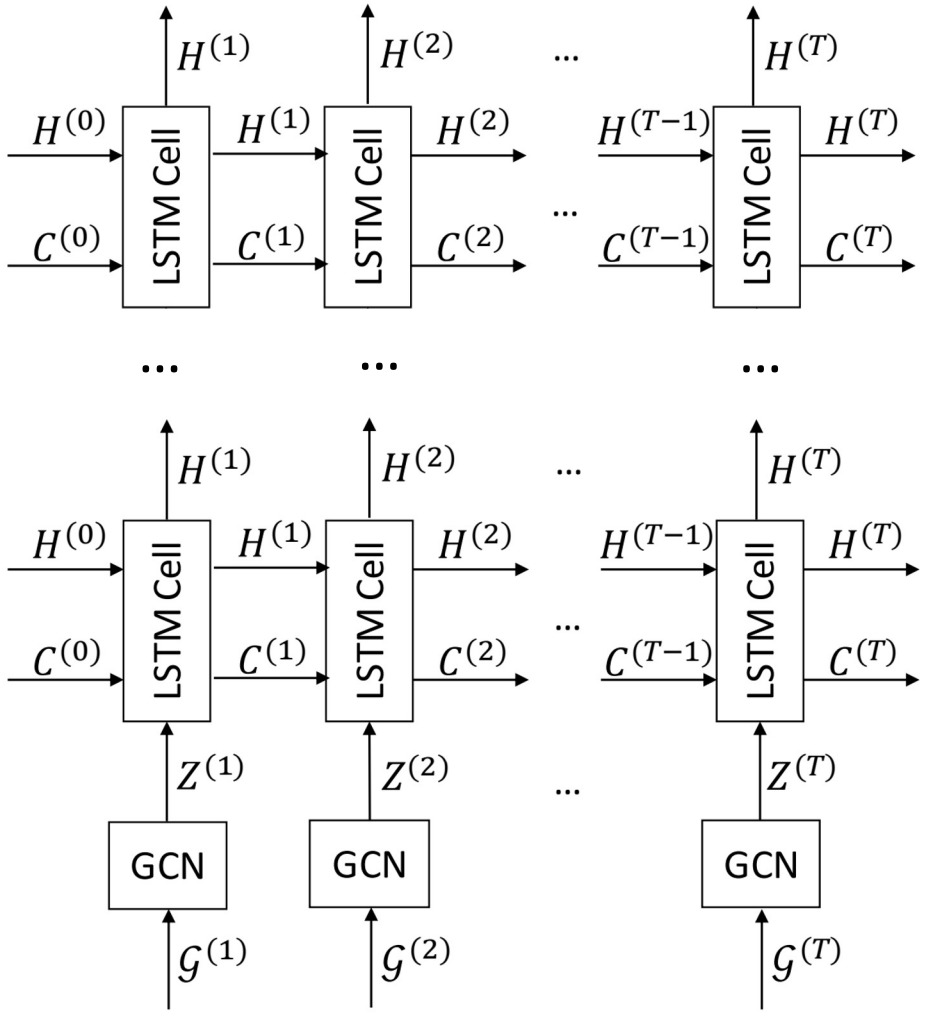
DTDG techniques: GNN-RNN

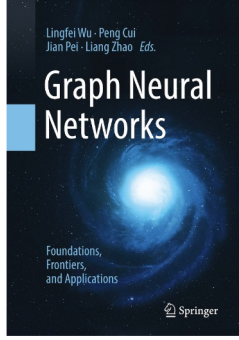
GNN-RNN: sequence of graphs
RNN can be vanilla, LSTM, biRNN, Transformer, etc.

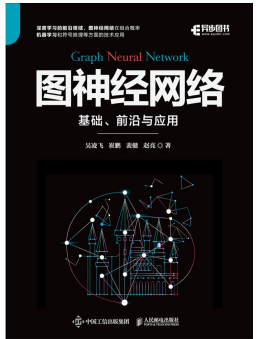
$$\boldsymbol{Z}^{(t)} = GCN(\boldsymbol{X}^{(t)}, \boldsymbol{A}^{(t)})$$

$$\boldsymbol{H}^{(t)}, \boldsymbol{C}^{(t)} = LSTM(\boldsymbol{Z}^{(t)}, \boldsymbol{H}^{(t-1)}, \boldsymbol{C}^{(t-1)})$$







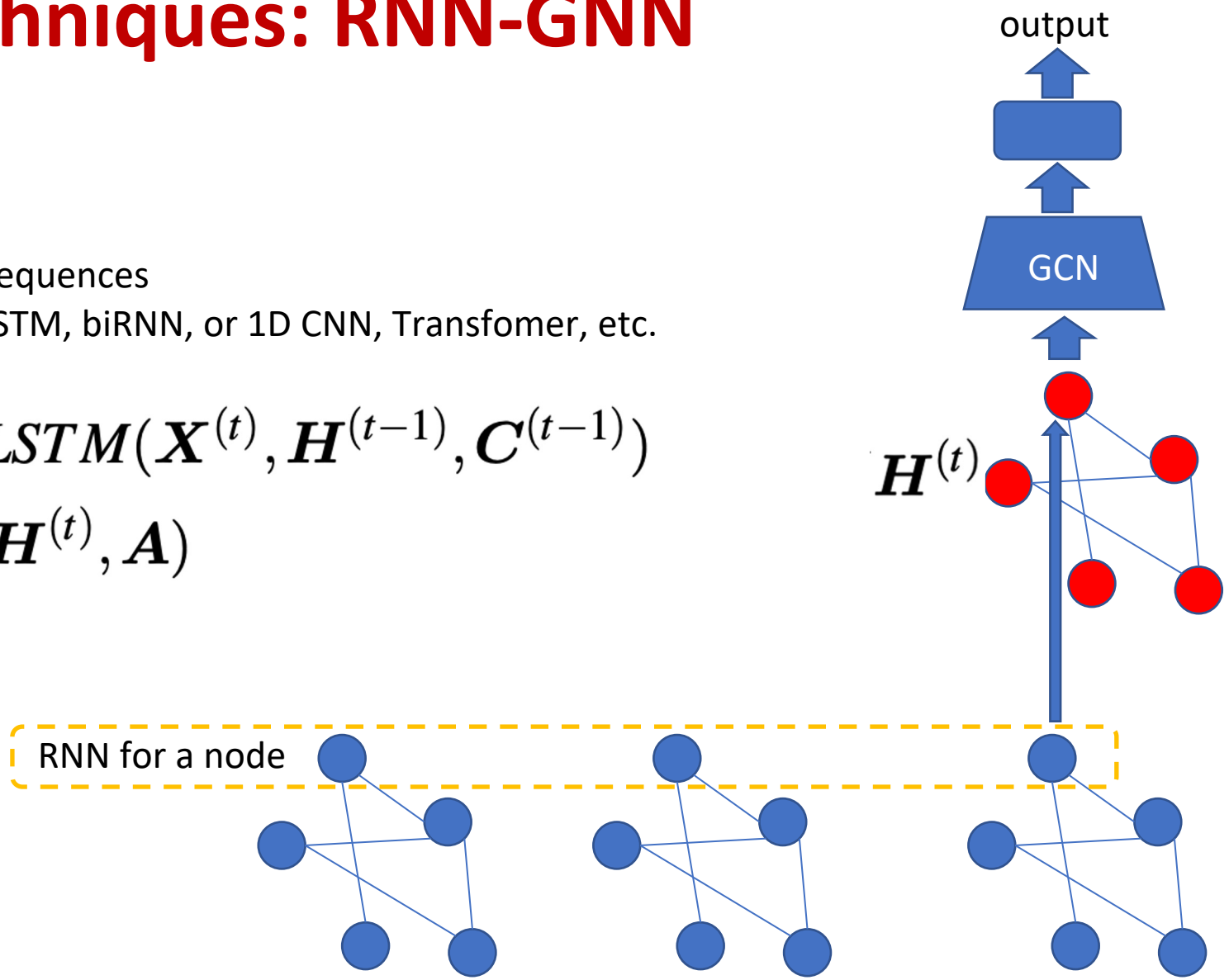


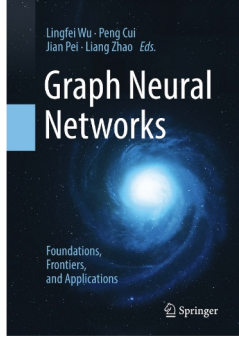
DTDG techniques: RNN-GNN

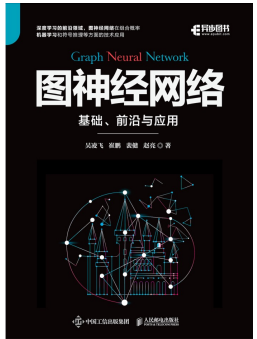
RNN-GNN: graph of sequences
RNN can be vanilla, LSTM, biRNN, or 1D CNN, Transfomer, etc.

$$\boldsymbol{H}^{(t)}, \boldsymbol{C}^{(t)} = LSTM(\boldsymbol{X}^{(t)}, \boldsymbol{H}^{(t-1)}, \boldsymbol{C}^{(t-1)})$$

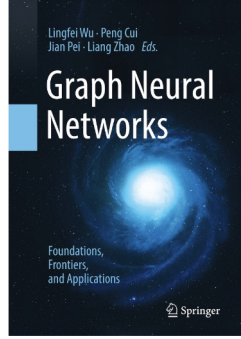
$$\boldsymbol{Z}^{(t)} = GCN(\boldsymbol{H}^{(t)}, \boldsymbol{A})$$







GNN for CTDG techniques [Kumar et al. 2019b]



Given an observation $(AddEdge, (v_i, v_j), t)$, the embedding of v_i and v_j can be updated as follows.

$$\boldsymbol{Z}_i^{(t)} = RNN_{source}((\boldsymbol{Z}_j^{(t-)} \,||\, \Delta t_i \,||\, \boldsymbol{f}),\ \boldsymbol{Z}_i^{(t-)})$$

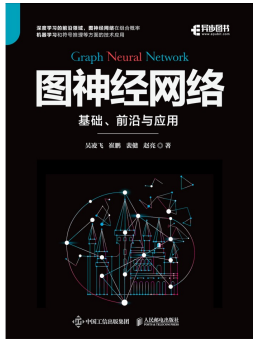
$$\boldsymbol{Z}_j^{(t)} = RNN_{target}((\boldsymbol{Z}_i^{(t-)} \,||\, \Delta t_j \,||\, \boldsymbol{f}),\ \boldsymbol{Z}_j^{(t-)})$$

Then $Z_i^{(t)}$ and $Z_j^{(t)}$ are concatenated to generate the final edge prediction and supervised by $(AddEdge, (v_i, v_j), t)$

1-hop neighbor is considered in [Trivedi et al. (2019],
multi-hop neighbor is considered in TGAT [Xu et al. 2020a].

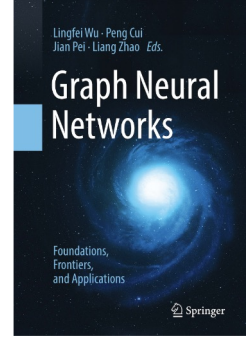
$$\boldsymbol{Z}_i^{(t)} = RNN((\boldsymbol{z}_{\mathcal{N}}(v_j)\Delta t_i),\ \boldsymbol{Z}_i^{(t-)})$$

$\boldsymbol{z}_{\mathcal{N}(v_j)}$ is the neighborhood of v_j

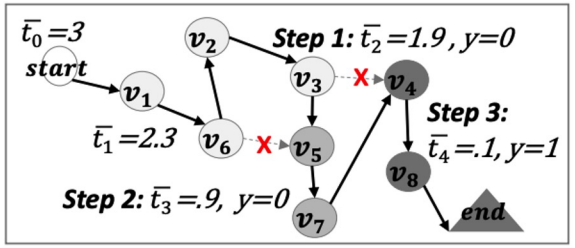


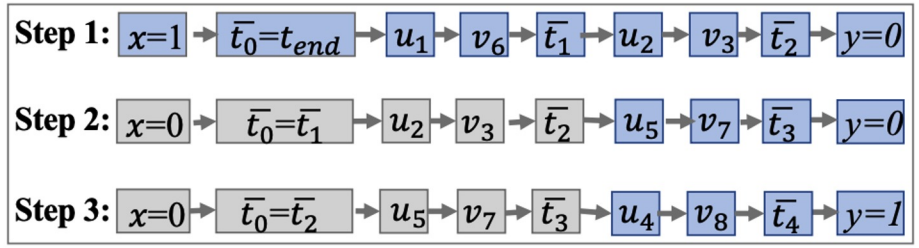
Dynamic Graph Generation for CTDG

[Zhang et al. 2021]

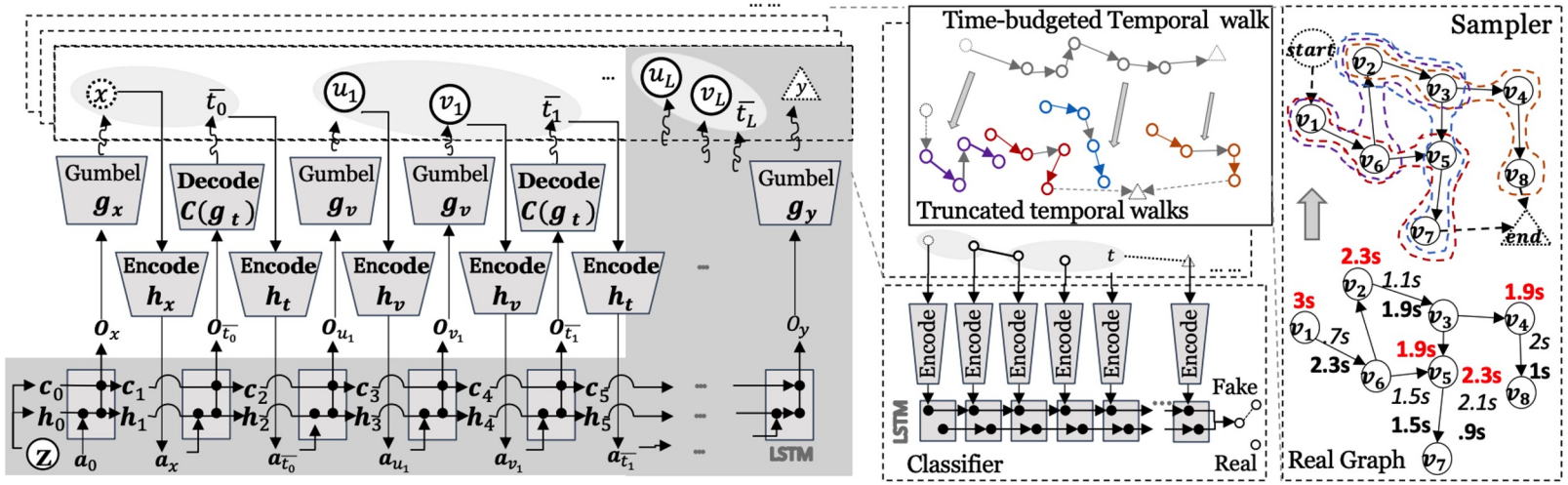


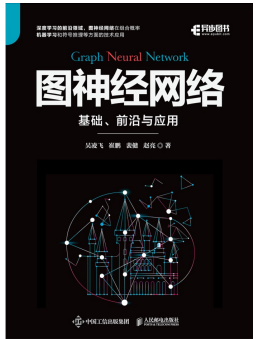
- Continuous-time dynamic graph generation
  - Temporal random walk generation and assembly
    - Challenge: large variance in random walk length: truncated temporal random walk is used





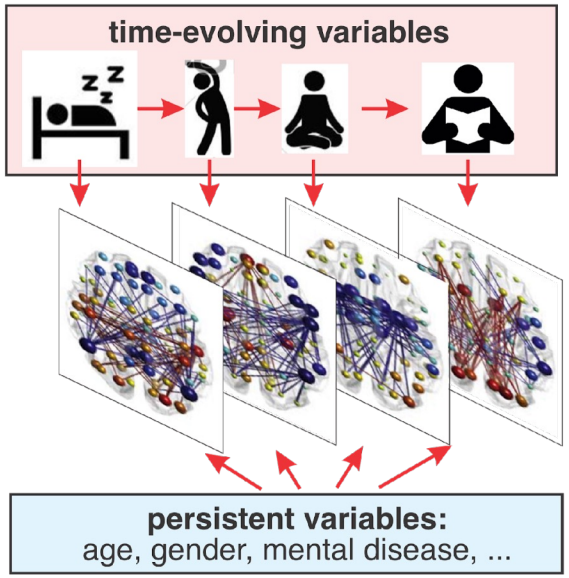
  - Truncated termpoal random walk generator



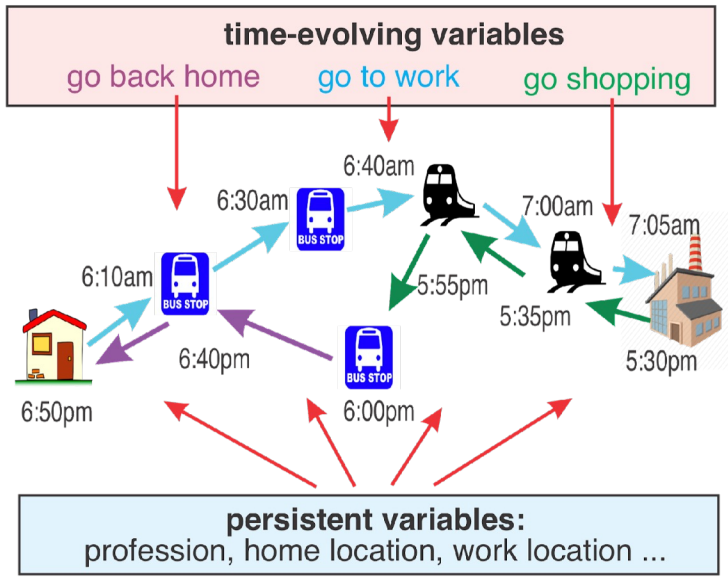


Dynamic Graph Generation for DTDG
[Zhang et al. 2021]

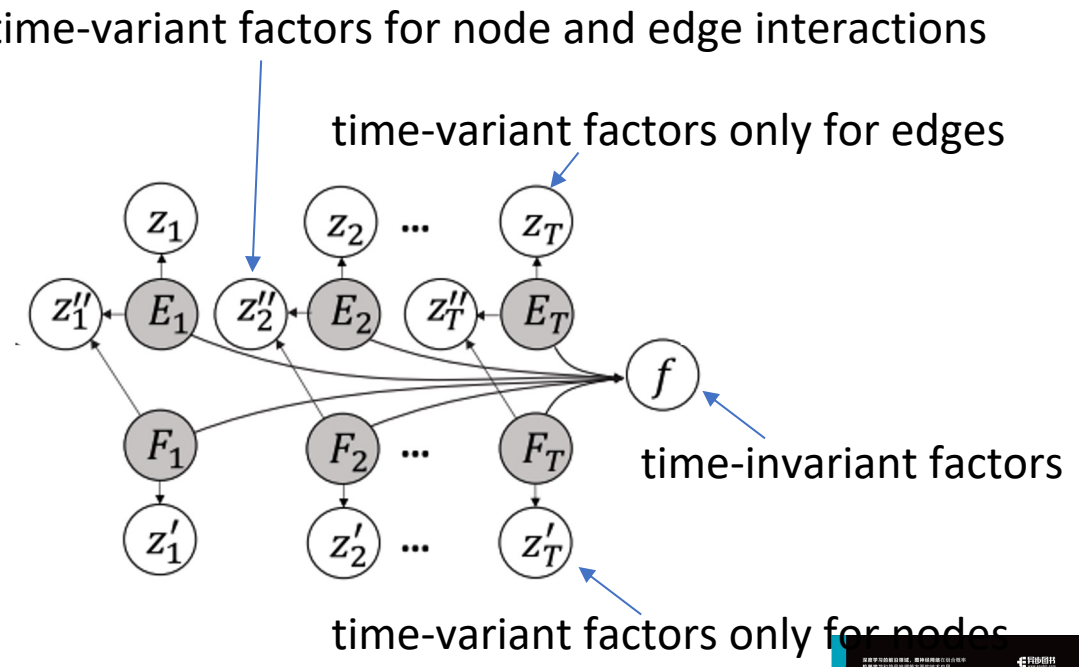
- Discrete-time dynamic graph generation

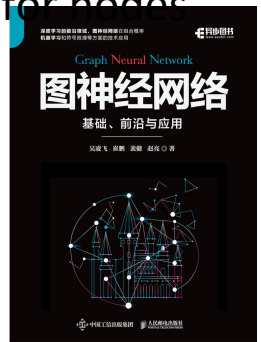


(d) Dynamic functional connectivity



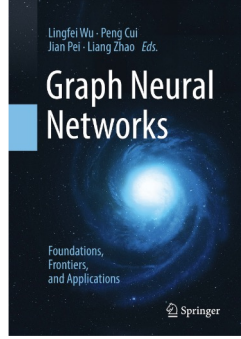
(e) Human mobility network

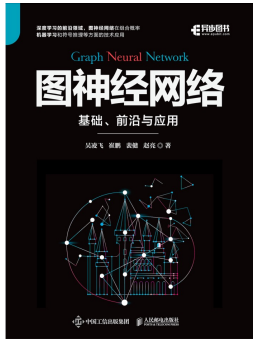


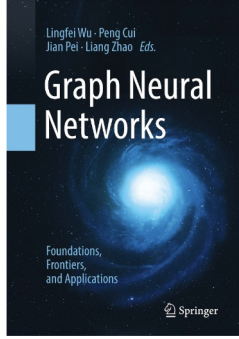


Future directions

- Node addition and removal are still challenges.
- Inverse problem of graph dynamics
- Spatiotemporal







References

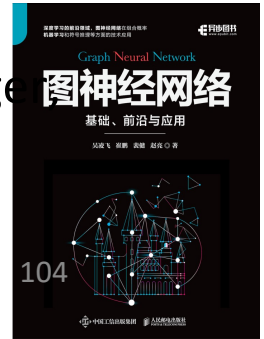
•[Rossi 2020] Rossi, Emanuele, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. "Temporal graph networks for deep learning on dynamic graphs." *arXiv preprint arXiv:2006.10637* (2020).

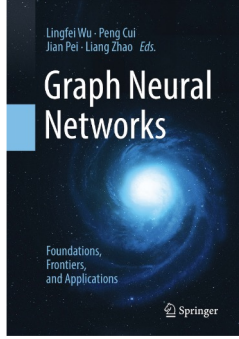
•[Xu 2020] Xu, Mengjia. "Understanding graph embedding methods and their applications." *SIAM Review* 63, no. 4 (2021): 825-853.

•[Zhang et al. 2021] Liming Zhang, Liang Zhao, Dieter Pfoser, Shan Qin and Chen Ling. TG-GAN: Continuous-time Temporal Graph Deep Generative Models with Time-Validity Constraints. The 30th International World Wide Web Conference, the Web Conference (WWW 2021), accepted.

•[Yao et al. 2016] Yao L, Wang L, Pan L, Yao K (2016) Link prediction based on common-neighbors for dynamic social network. Procedia Computer Science 83:82–89

•[Seo et al. 2018] Seo Y, Defferrard M, Vandergheynst P, Bresson X (2018) Structured sequence modeling with graph convolutional recurrent networks. In: Neural Information Processing, Springer, pp 362–373





References

[Manessi and Rozza 2020] Manessi F, Rozza A (2020) Graph-based neural network models with multiple selfsupervised auxiliary tasks. arXiv preprint arXiv: 201107267

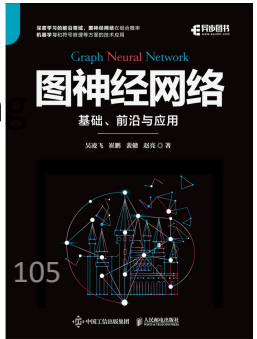
[Xu 2019a] Xu D, Cheng W, Luo D, Liu X, Zhang X (2019a) Spatio-temporal attentive rnn for node classification in temporal attributed graphs. In: International Joint Conference on Artificial Intelligence, pp 3947–3953

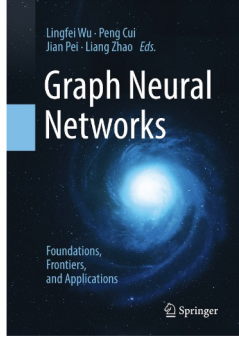
[Kumar et al. 2019b] Kumar S, Zhang X, Leskovec J (2019b) Predicting dynamic embedding trajectory in temporal interaction networks. In: ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp 1269–1278

[Zhang et al. 2021] Wenbin Zhang, Liming Zhang, Dieter Pfoser, Liang Zhao. Disentangled Dynamic Graph Deep Generation, SIAM International Conference on Data Mining (SDM 2021), (acceptance rate: 21.3%), accepted.

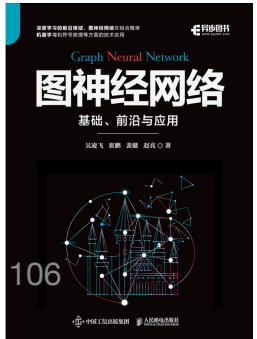
[Trivedi et al. 2019] Trivedi R, Farajtabar M, Biswal P, Zha H (2019) Dyrep: Learning representations over dynamic graphs. In: International Conference on Learning Representations

[Xu et al. 2020a] Xu D, Ruan C, Korpeoglu E, Kumar S, Achan K (2020a) Inductive representation learning on temporal graphs. In: International Conference on Learning Representations



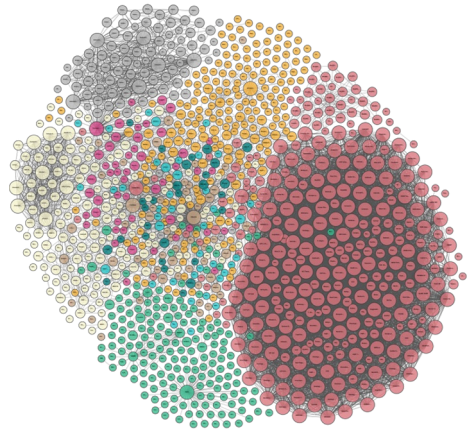


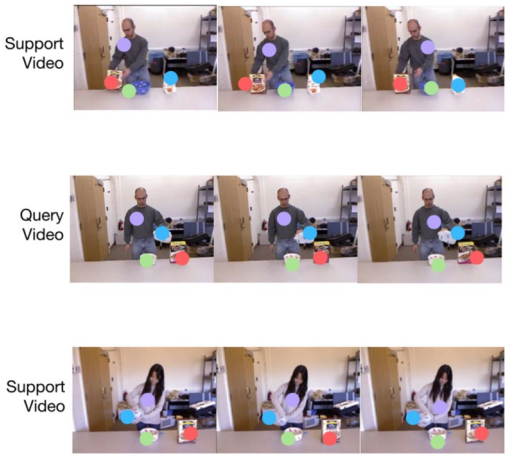
GNNs: Graph Matching (Chapter 13)

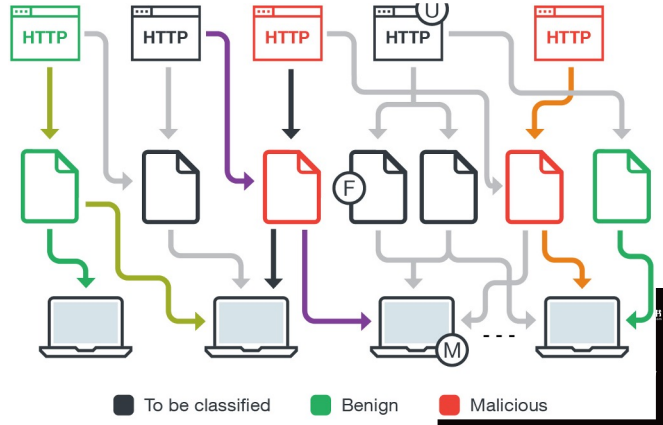


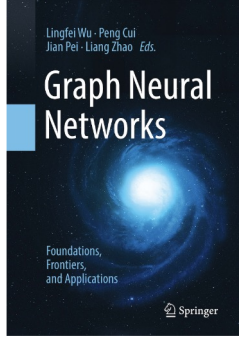
Graph Matching: Applications

- Graph similar searching in graph-based database
- 3D Action Recognition
- Unknown malware detection
- Promising selection in automatic theory proving





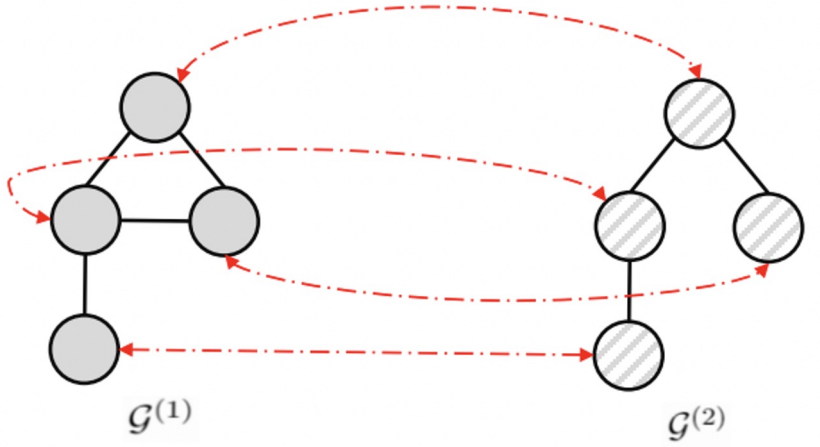


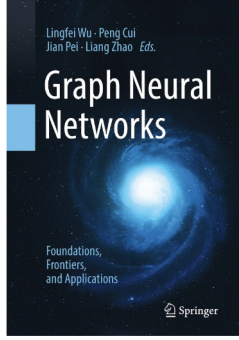


Graph Matching: Node Correspondence

Definition 13.1 (Graph Matching Problem). Given a pair of input graphs $\mathcal{G}^{(1)} = (\mathcal{V}^{(1)}, \mathcal{E}^{(1)}, A^{(1)}, X^{(1)}, E^{(1)})$ and $\mathcal{G}^{(2)} = (\mathcal{V}^{(2)}, \mathcal{E}^{(2)}, A^{(2)}, X^{(2)}, E^{(2)})$ of equal size n, the graph matching problem is to find a node-to-node correspondence matrix $S \in \{0,1\}^{n \times n}$ (*i.e.*, also called *assignment matrix* and *permutation matrix*) between the two graphs $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(2)}$. Each element $S_{i,a} = 1$ if and only if the node $v_i \in \mathcal{V}^{(1)}$ in $\mathcal{G}^{(1)}$ corresponds to the node $v_a \in \mathcal{V}^{(2)}$ in $\mathcal{G}^{(2)}$.

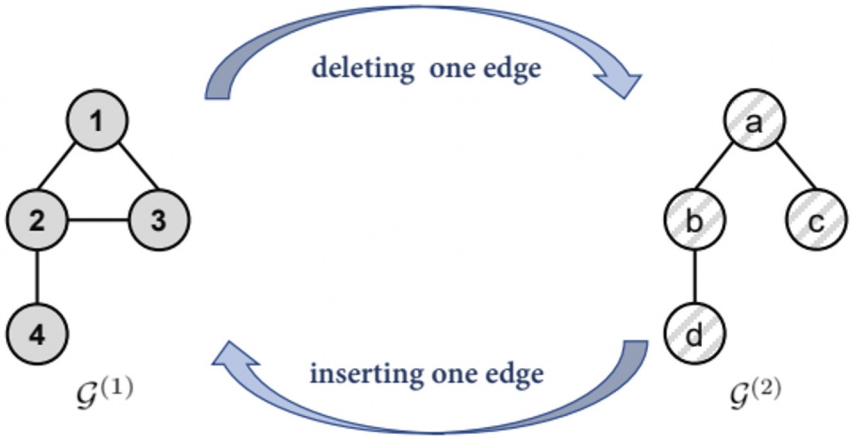
- NP-hard problem for exact and optimal match
- Computationally expensive, poor scalability for large graph
- Few datasets, hard to find ground truth
- Many modern methods based on GNNs



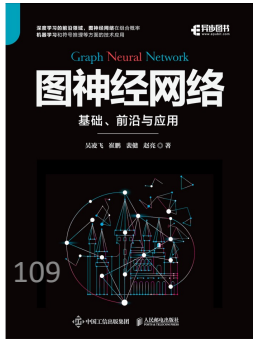


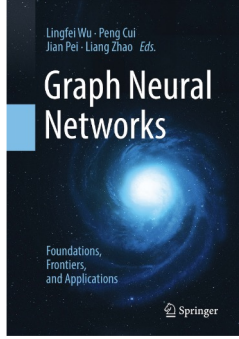
Graph Matching: Similarity Learning

Definition 13.3 (Graph Similarity Problem). Given two input graphs $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(2)}$, the purpose of graph similarity problem is to produce a similarity score s between $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(2)}$. In line with the notations in Section 13.2.1, the $\mathcal{G}^{(1)} = (\mathcal{V}^{(1)}, \mathcal{E}^{(1)}, A^{(1)}, X^{(1)})$ is represented as set of n nodes $v_i \in \mathcal{V}^{(1)}$ with a feature matrix $X^{(1)} \in \mathbb{R}^{n \times d}$, edges $(v_i, v_j) \in \mathcal{E}^{(1)}$ formulating an adjacency matrix $A^{(1)}$. Similarly, $\mathcal{G}^{(2)} = (\mathcal{V}^{(2)}, \mathcal{E}^{(2)}, A^{(2)}, X^{(2)}$ is represented as set of m nodes $v_a \in \mathcal{V}^{(2)}$ with a feature matrix $X^{(2)} \in \mathbb{R}^{m \times d}$, edges $(v_a, v_b) \in \mathcal{E}^{(2)}$ formulating an adjacency matrix $A^{(2)}$.



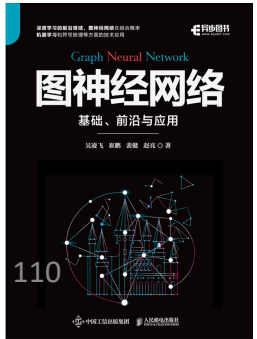
- NP-hard problem as well
- Sub-optimal heuristic solutions in polynomial time suffer from scalability issue
- GNN-based methods demonstrating superiority over traditional methods in both effectiveness and efficiency



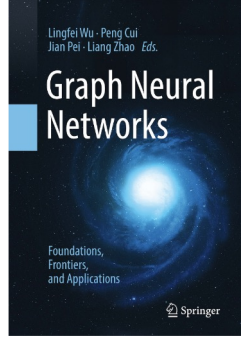


Graph Similarity Learning: Formulation

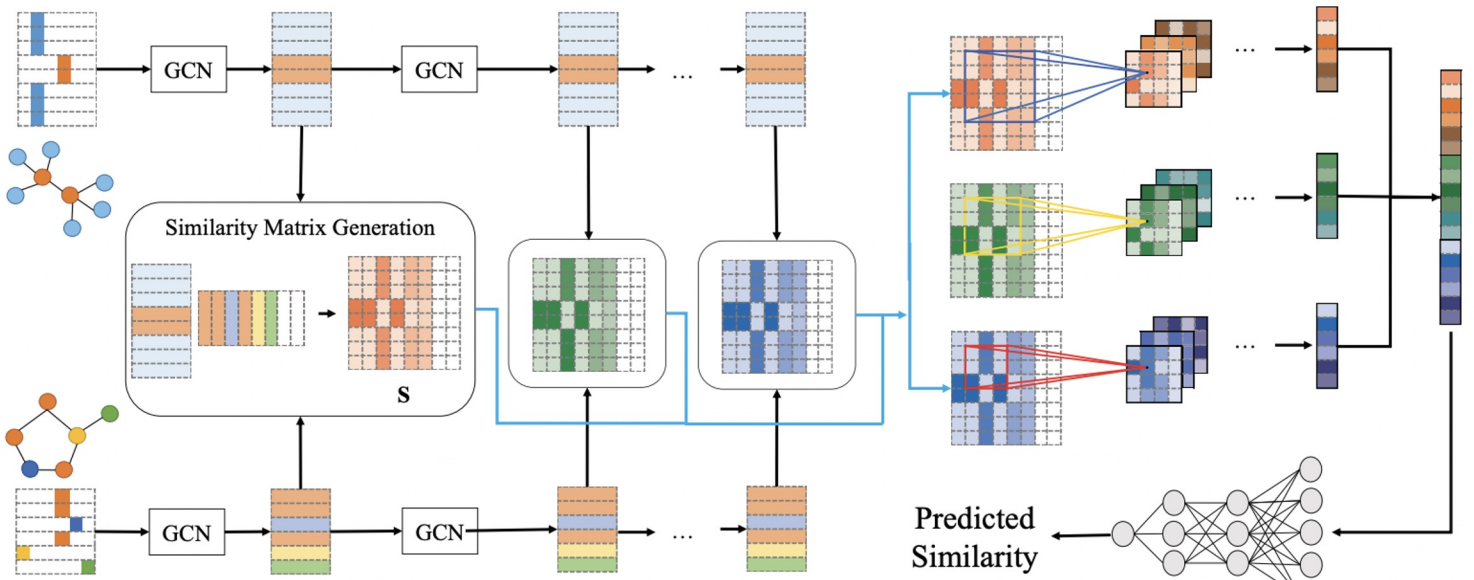
- Graph Similarity Learning
  - **Input:** a pair of graph inputs $(\mathrm{G}^1, \mathrm{G}^2) \in \boldsymbol{G}\times\boldsymbol{G}$
    - $\mathrm{G}^1 = (\mathrm{V}^1, \mathrm{E}^1)$ with $(\mathrm{X}^1, \mathrm{A}^1)$, where $\mathrm{X}^1 \in \mathrm{R}^{\mathrm{N}\times\mathrm{d}}$, $\mathrm{X}^1 \in \mathrm{R}^{\mathrm{N}\times\mathrm{N}}$
    - $\mathrm{G}^2 = (\mathrm{V}^2, \mathrm{E}^2)$ with $(\mathrm{X}^2, \mathrm{A}^2)$, where $\mathrm{X}^2 \in \mathrm{R}^{\mathrm{M}\times\mathrm{d}}, \mathrm{X}^2 \in \mathrm{R}^{\mathrm{M}\times M}$
  - **Output:** a similarity score $\mathrm{y} \in \boldsymbol{Y}$
    - $\mathbf{Y} = \{-1, 1\}$: graph-graph classification task
    - $\mathbf{Y} = [0, 1]$: graph-graph regression task



Graph Similarity Learning with Convolutional Set Matching

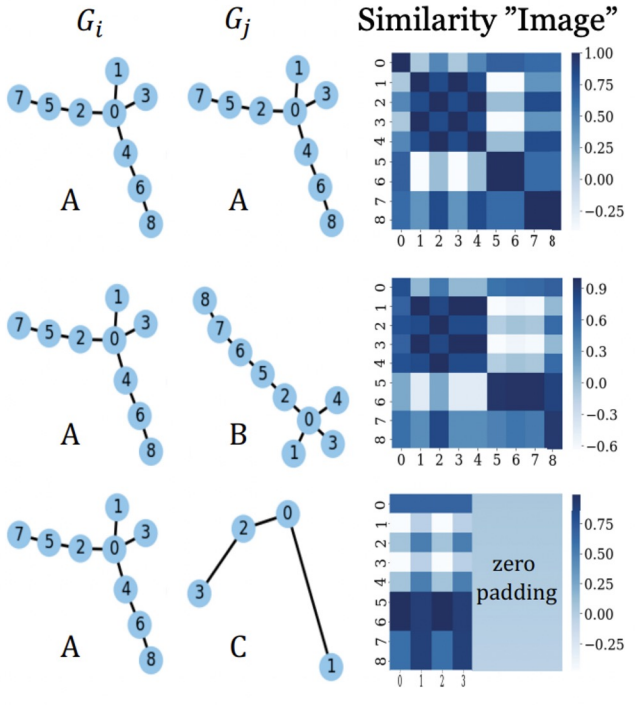


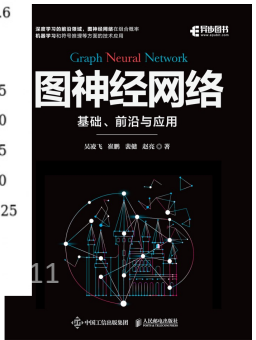
- GraphSim (Bai et al, 2020b): turn the computation of GED into an end-end learning problem rather than approximation methods with combinatorial search.



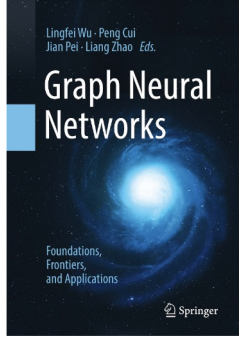
Stage I: Multi-Scale Neighbor Aggregation
Stage II: Similarity Matrix Generation

Stage III:
CNN Based Similarity Score Computation

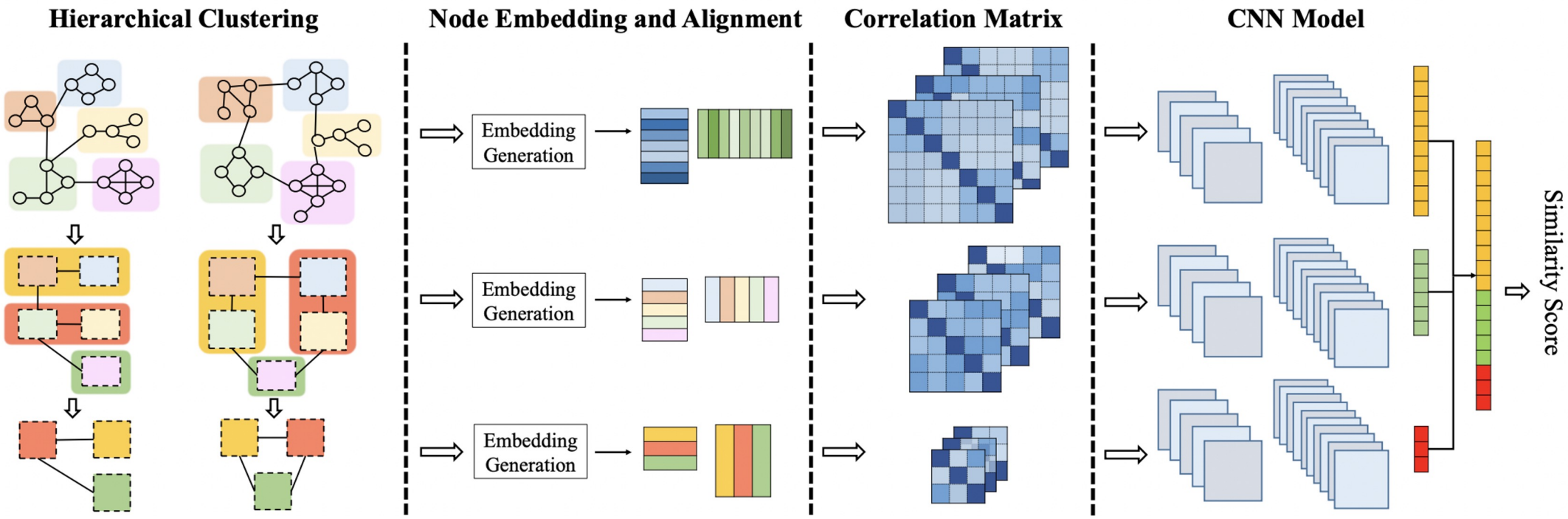


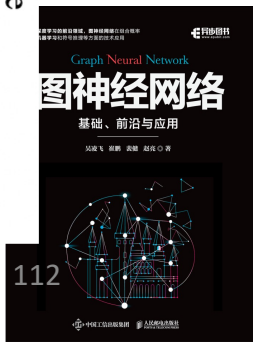


Graph Similarity Learning with Hierarchical Clustering

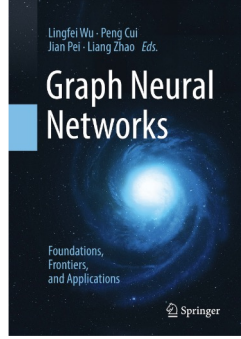


- Hierarchical graph matching network (HGMN) (Xiu et al, 2020): learn the graph similarity from a multi-scale view.

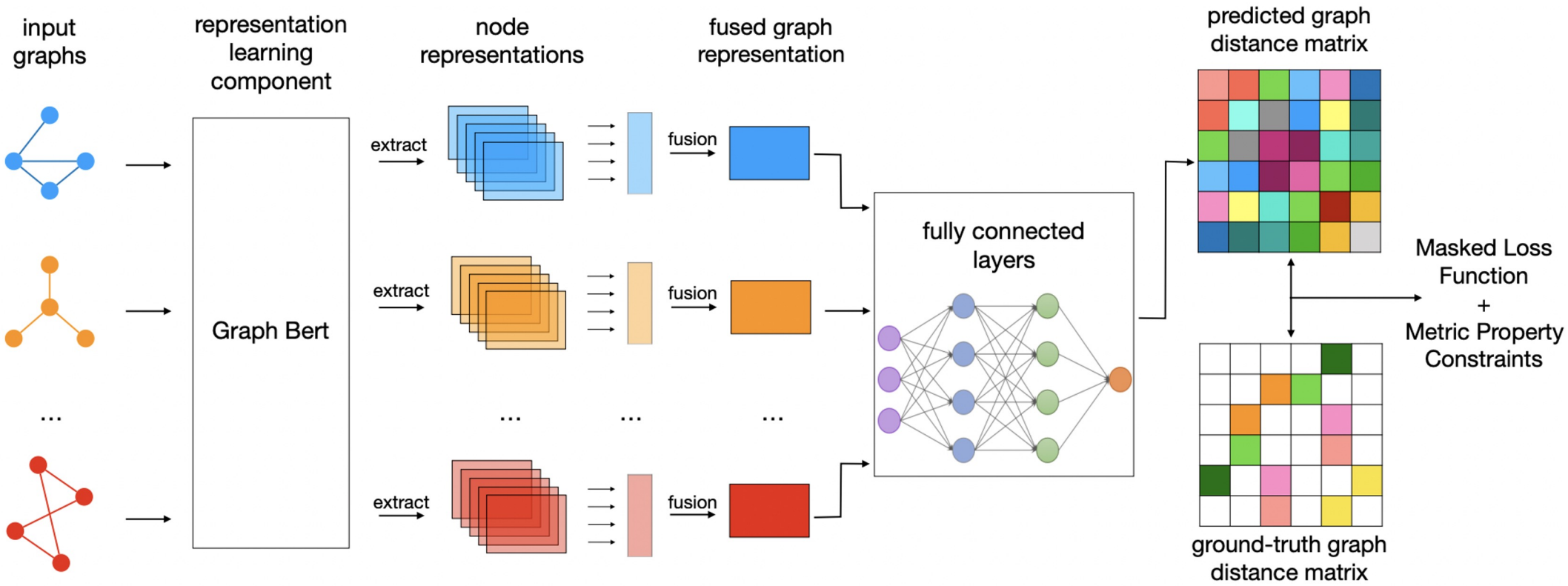


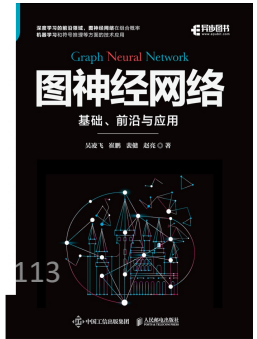


Graph Similarity Learning with Graph-BERT

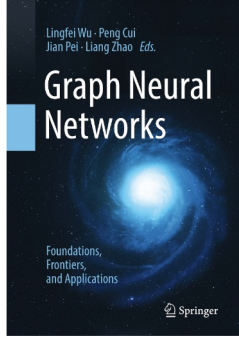


- GB-DISTANCE (Zhang, 2020): a new graph neural network-based distance metric learning approaches with Graph-BERT.

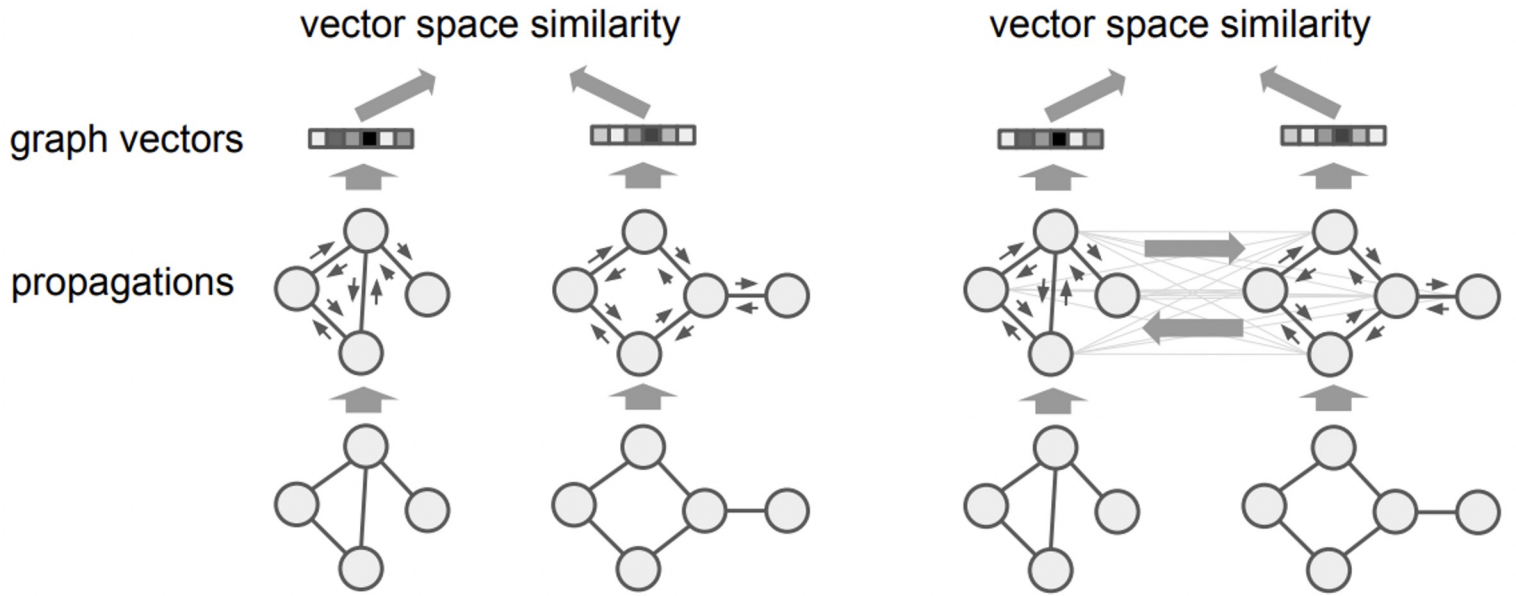


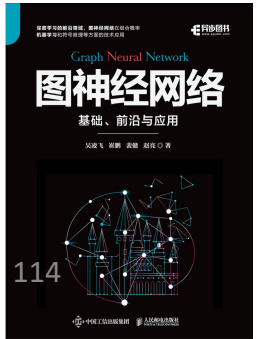


Graph Similarity Learning with Cross-graph Matching



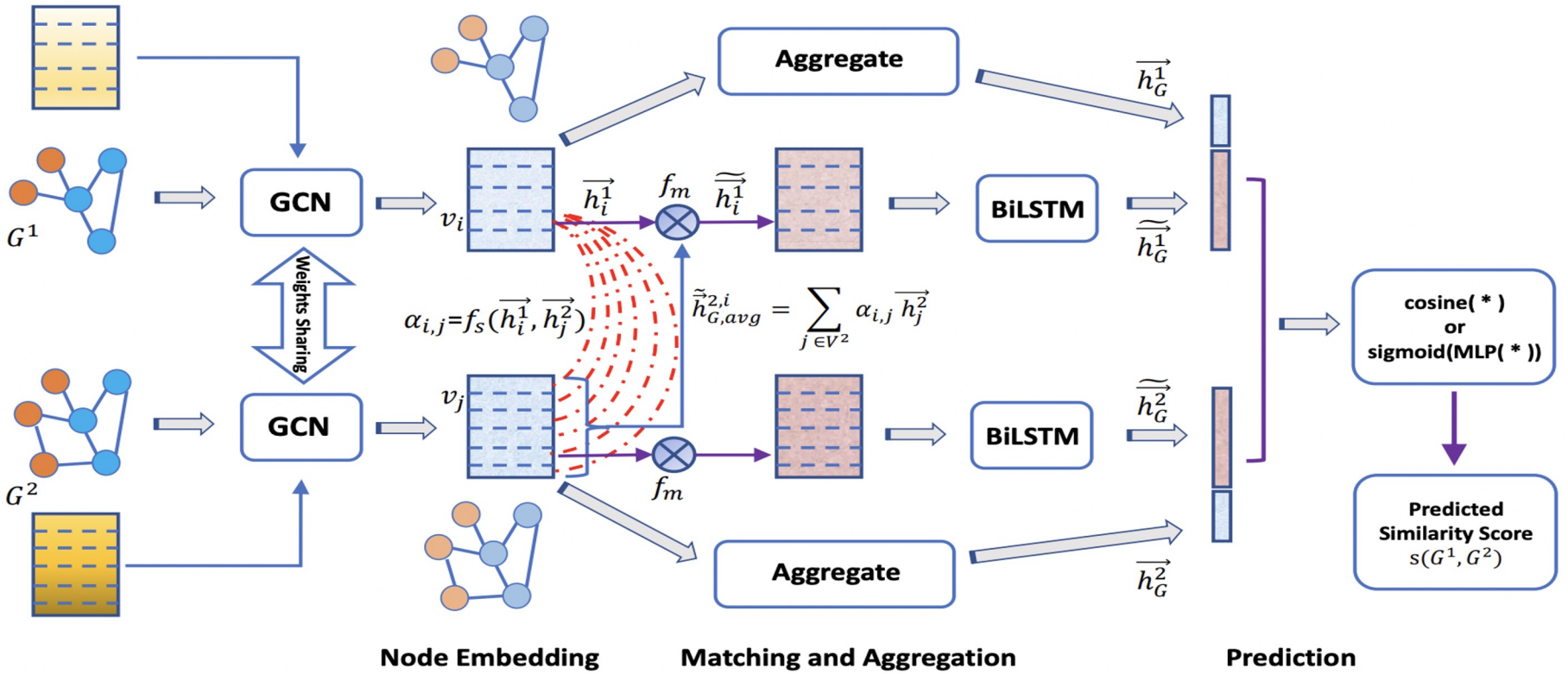
- Graph Matching Networks (GMN) (Li et al, 2020): employs a cross-graph matching network based on standard message-passing GNNs to iteratively generate more discriminative node embeddings.

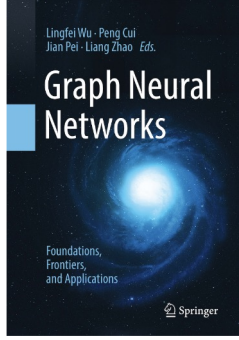




Graph Similarity Learning with Node-Graph Interactions

- Multi-level graph matching network (MGMN) (Ling and Wu et al, 2020): consisted of a siamese graph neural network (SGNN) and a novel node-graph matching network (NGMN).





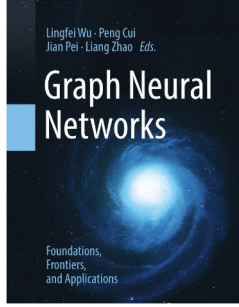
Experimental Setup

We evaluate our model on **four datasets** for both **classification** and **regression** tasks.

Table 1: Summary statistics of datasets for both classification & regression tasks.

Tasks	Datasets	Sub-datasets	# of Graphs	# of Functions	Min # Nodes	Max # Nodes	AVG # Nodes	Init Feature Dimensions
classif-ication	**FFmpeg**	[3, 200]	83008	10376	3	200	18.83	6
		[20, 200]	31696	7668	20	200	51.02	
		[50, 200]	10824	3178	50	200	90.93	
	OpenSSL	[3, 200]	73953	4249	3	200	15.73	6
		[20, 200]	15800	1073	20	200	44.89	
		[50, 200]	4308	338	50	200	83.68	
regre-ssion	**AIDS700**	-	700	-	2	10	8.90	29
	LINUX1000	-	1000	-	4	10	7.58	1

Experiments: Classification Tasks



Model	FFmpeg			OpenSSL		
	[3, 200]	[20, 200]	[50, 200]	[3, 200]	[20, 200]	[50, 200]
SimGNN	95.38±0.76	94.31±1.01	93.45±0.54	95.96±0.31	93.58±0.82	94.25±0.85
GMN	94.15±0.62	95.92±1.38	94.76±0.45	96.43±0.61	93.03±3.81	93.91±1.65
GraphSim	97.46±0.30	96.49±0.28	94.48±0.73	96.84±0.54	94.97±0.98	93.66±1.84
SGNN (Max)	93.92±0.07	93.82±0.28	85.15±1.39	91.07±0.10	88.94±0.47	82.10±0.51
SGNN (FCMax)	95.37±0.04	96.29±0.14	95.98±0.32	92.64±0.15	93.79±0.17	93.21±0.82
SGNN (BiLSTM)	96.92±0.13	97.62±0.13	96.35±0.33	95.24±0.06	96.30±0.27	93.99±0.62
NGMN (Max)	73.74±8.30	73.85±1.76	77.72±2.07	67.14±2.70	63.31±3.29	63.02±2.77
NGMN (FCMax)	97.28±0.08	96.61±0.17	96.65±0.30	95.37±0.19	96.08±0.48	**95.90±0.73**
NGMN (BiLSTM)	97.73±0.11	98.29±0.21	96.81±0.96	96.56±0.12	**97.60±0.29**	92.89±1.31
MGMN (Max + BiLSTM)	97.44±0.32	97.84±0.40	97.22±0.36	94.77±1.80	97.44±0.26	94.06±1.60
MGMN (FCMax + BiLSTM)	**98.07±0.06**	**98.29±0.10**	**97.83±0.11**	96.87±0.24	97.59±0.24	95.58±1.13
MGMN (BiLSTM + BiLSTM)	97.56±0.38	98.12±0.04	97.16±0.53	**96.90±0.10**	97.31±1.07	95.87±0.88

Classification Tasks:

- ✓ detecting whether two binaries are compiled from the same one source code function
- ✓ learning a similarity score $y \in \mathbf{Y} = \{-1, 1\}$ between two control flow graphs G^1 and G^2

Evaluation metric: *Area Under the ROC Curve* (**AUC**)

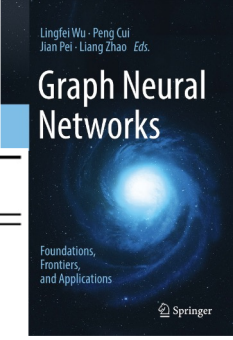
Experiments: Regression Tasks

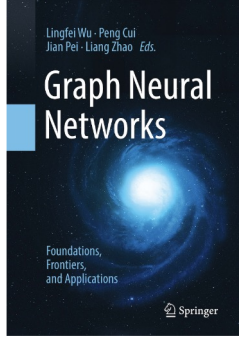
Datasets	Model	mse (10^{-3})	ρ	τ	p@10	p@20
AIDS700	SimGNN	1.376±0.066	0.824±0.009	0.665±0.011	0.400±0.023	0.489±0.024
	GMN	4.610±0.365	0.672±0.036	0.497±0.032	0.200±0.018	0.263±0.018
	GraphSim	1.919±0.060	0.849±0.008	0.693±0.010	0.446±0.027	0.525±0.021
	SGNN (Max)	2.822±0.149	0.765±0.005	0.588±0.004	0.289±0.016	0.373±0.012
	SGNN (FCMax)	3.114±0.114	0.735±0.009	0.554±0.008	0.278±0.021	0.364±0.017
	SGNN (BiLSTM)	1.422±0.044	0.881±0.005	0.718±0.006	0.376±0.020	0.472±0.014
	NGMN (Max)	2.378±0.244	0.813±0.015	0.642±0.013	**0.578±0.199**	**0.583±0.169**
	NGMN (FCMax)	2.220±1.547	0.808±0.145	0.656±0.122	0.425±0.078	0.504±0.064
	NGMN (BiLSTM)	1.191±0.048	0.904±0.003	0.749±0.005	0.465±0.011	0.538±0.007
	MGMN (Max + BiLSTM)	1.210±0.020	0.900±0.002	0.743±0.003	0.461±0.012	0.534±0.009
	MGMN (FCMax + BiLSTM)	1.205±0.039	0.904±0.002	0.749±0.003	0.457±0.014	0.532±0.016
	MGMN (BiLSTM + BiLSTM)	**1.169±0.036**	**0.905±0.002**	**0.751±0.003**	0.456±0.019	0.539±0.018
LINUX1000	SimGNN	2.479±1.038	0.912±0.031	0.791±0.046	0.635±0.328	0.650±0.283
	GMN	2.571±0.519	0.906±0.023	0.763±0.035	0.888±0.036	0.856±0.040
	GraphSim	0.471±0.043	0.976±0.001	**0.931±0.003**	**0.956±0.006**	0.942±0.007
	SGNN (Max)	11.832±0.698	0.566±0.022	0.404±0.017	0.226±0.106	0.492±0.190
	SGNN (FCMax)	17.795±0.406	0.362±0.021	0.252±0.015	0.239±0.000	0.241±0.000
	SGNN (BiLSTM)	2.140±1.668	0.935±0.050	0.825±0.100	0.878±0.012	0.865±0.007
	NGMN (Max)*	16.921±0.000	-	-	-	-
	NGMN (FCMax)	4.793±0.262	0.829±0.006	0.665±0.011	0.764±0.170	0.767±0.166
	NGMN (BiLSTM)	1.561±0.020	0.945±0.002	0.814±0.003	0.743±0.085	0.741±0.086
	MGMN (Max + BiLSTM)	1.054±0.086	0.962±0.003	0.850±0.008	0.877±0.054	0.883±0.047
	MGMN (FCMax + BiLSTM)	1.575±0.627	0.946±0.019	0.817±0.034	0.807±0.117	0.784±0.108
	MGMN (BiLSTM + BiLSTM)	**0.439±0.143**	**0.985±0.005**	0.919±0.016	0.955±0.011	**0.943±0.014**

* As all duplicated experiments running on this setting do not converge in their training processes, their corresponding results cannot be calculated.

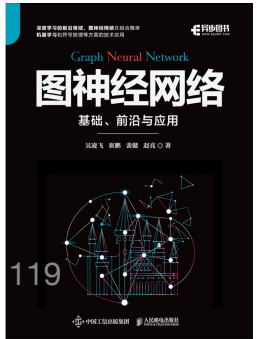
Regression Tasks: learning the graph edit distance $y \in \mathbf{Y} = [0,1]$ between two graphs G^1 and G^2

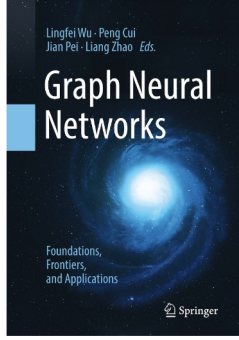
Evaluation metrics: mean square error (**mse**), spearman's rank correlation coefficient (**ρ**), kendall's rank correlation coefficient precision at k (**p@k**, $k = 10/20$)





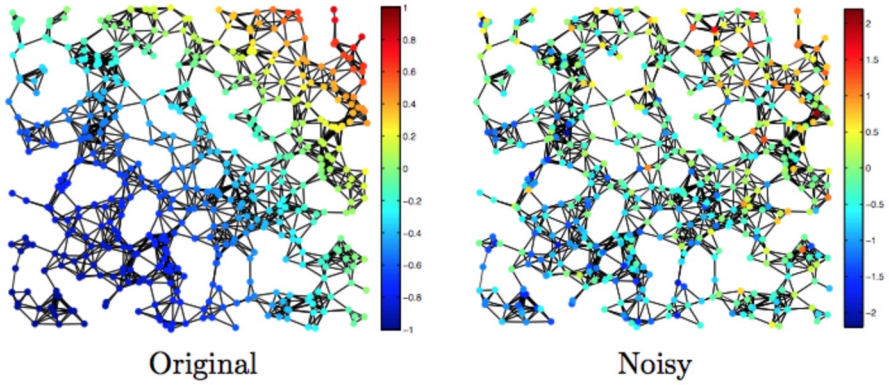
GNNs: Graph Structure Learning (Chapter 14)

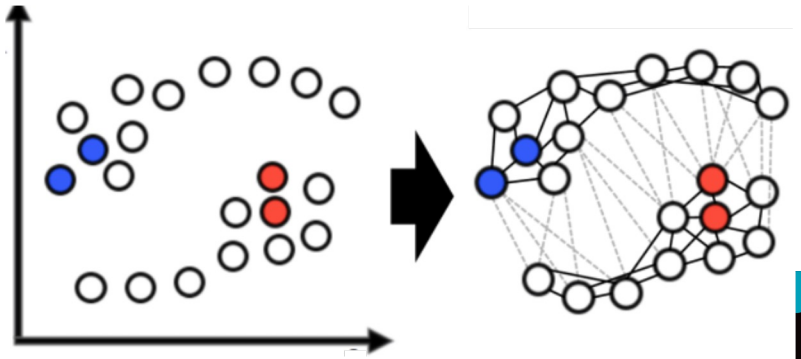


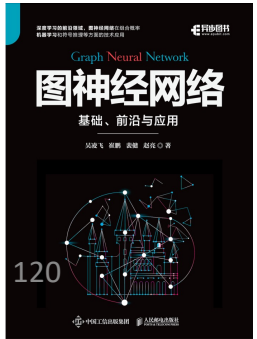


Graph Learning: Motivations

- GNNs are powerful, unfortunately, it requires graph-structured data available.
- Questionable if the given intrinsic graph-structures are optimal (i.e., noisy, incomplete, etc.) for the downstream tasks.
- Many applications (e.g., NLP tasks) may only have non-graph structured data or even just the original feature matrix, requiring additional graph construction.

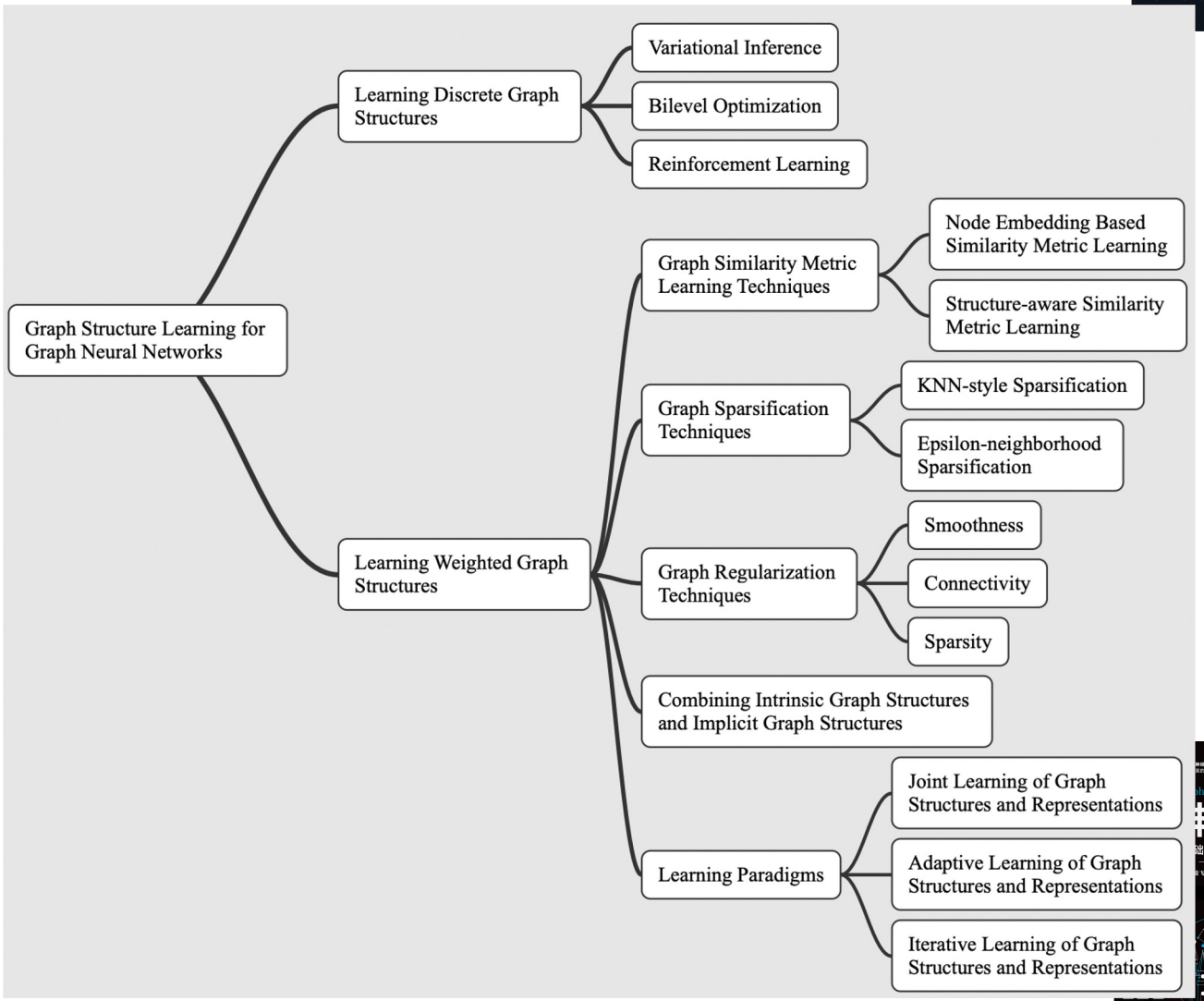


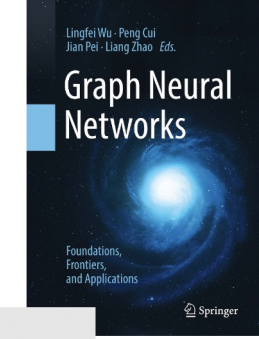


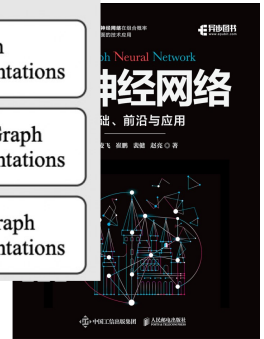


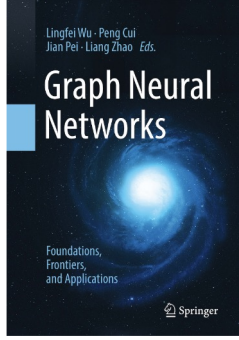
Graph Learning: Taxonomy

- Learning Discrete Graph Structures
- Learning Weighted Graph Structures
- Y. Chen and L. Wu, “Graph neural networks: Graph structure learning,” in Graph Neural Networks: Foundations, Frontiers, and Applications, L. Wu, P. Cui, J. Pei, and L. Zhao, Eds. Singapore: Springer Singapore, 2022, pp. 297–321.



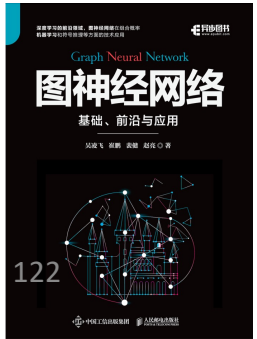




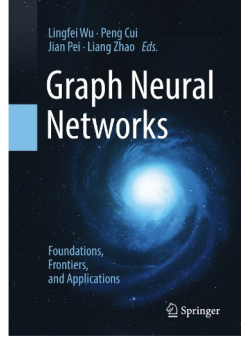


Graph Learning: Formulation

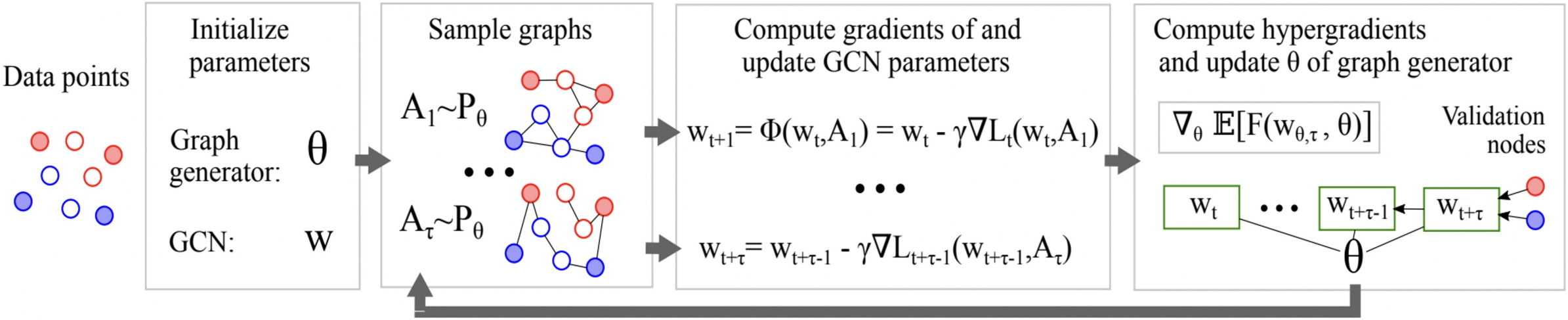
- Deep graph Learning
  - **Input:** A raw graph input $G \in \boldsymbol{G}$
    - Given a noisy $G = (V, E)$ with (X, A^0), where $X \in R^{N \times d}$, $A^0 \in R^{N \times N}$
    - Given initial feature matrix X, where $X \in R^{N \times d}$
  - **Output:** An optimal graph adjacency $A \in R^{N \times N}$ and node embeddings $Z \in R^{N \times d'}$

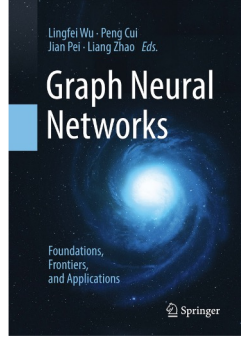


Graph Learning with bilevel-optimization



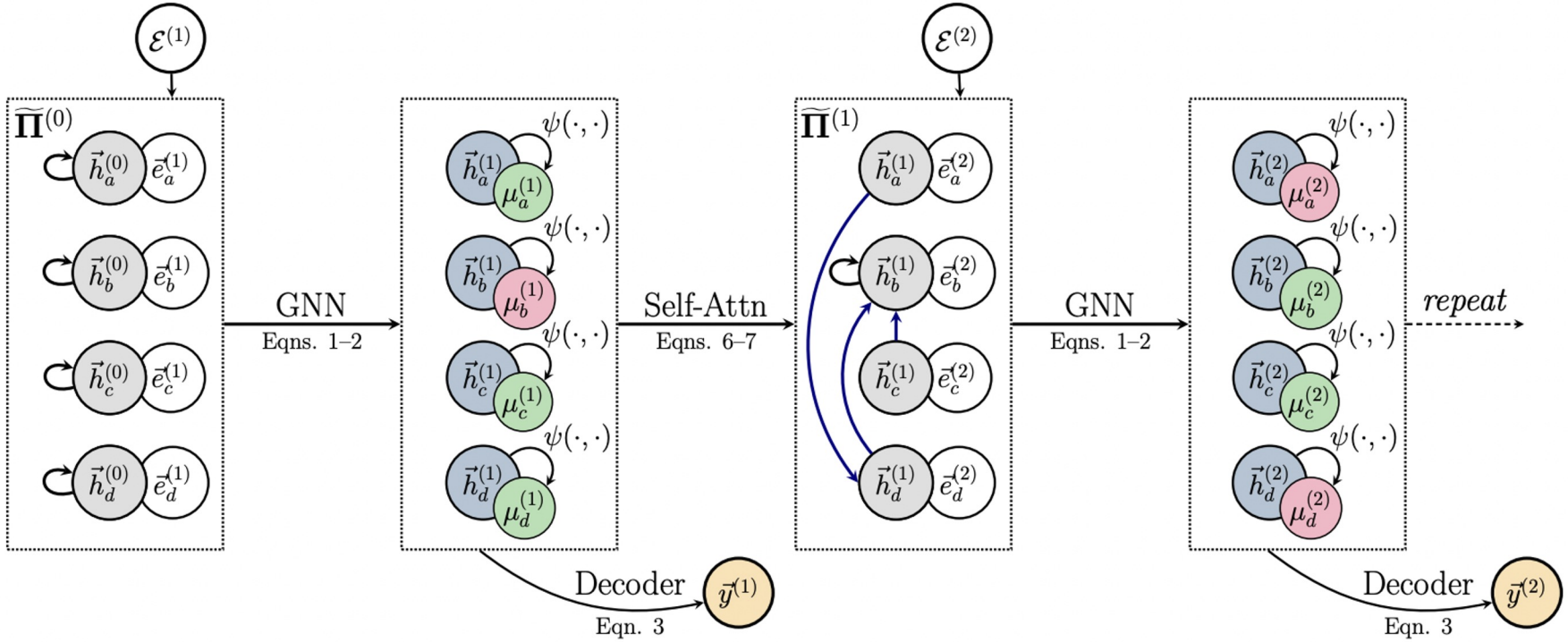
- LDS-GNN (Franceschi et al., 2019): jointly learn a discrete probability distribution on the edges of the graph and the parameters of GNNs by treating the task as a bilevel optimization problem

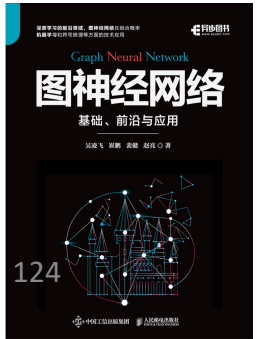


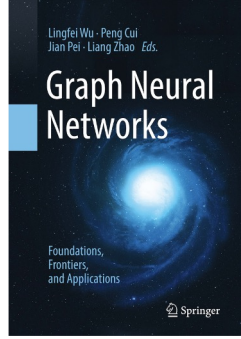


Graph Learning with Pointer Networks

- Pointer Graph Networks (Velickovic et al., 2021): computed the pair-wise node similarity with attention and obtained an "aggregated" adjacency matrix by deriving a new edge with rules.

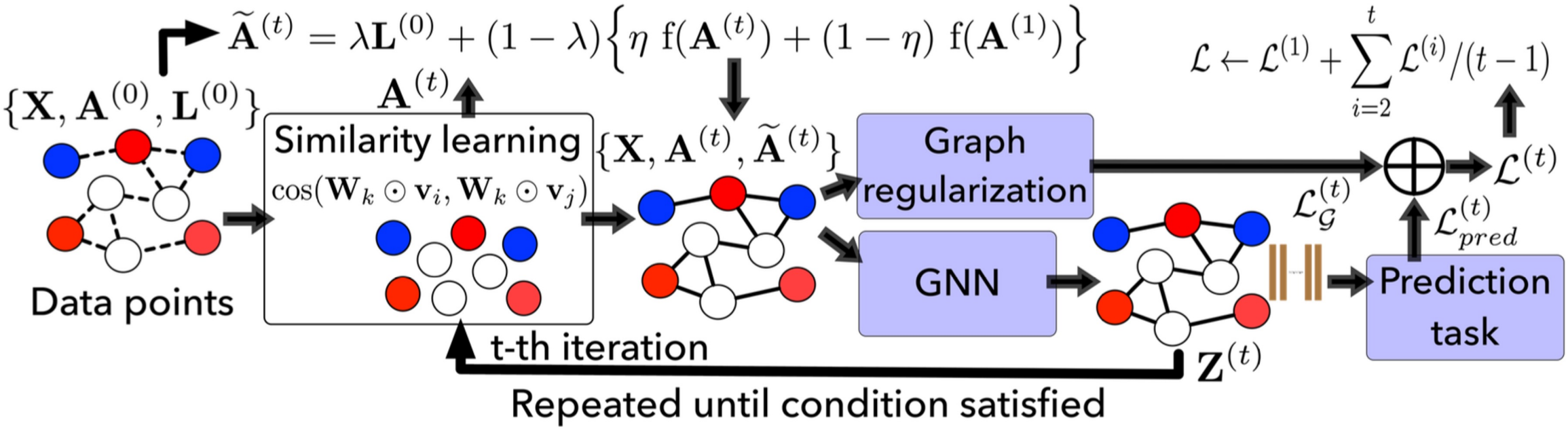






Graph Learning with Iterative Learning

- Iterative Deep Graph Learning (IDGL) (Chen and Wu et al., 2020), for jointly and iteratively learning graph structures and representations.



IDGL: Graph Learning as Similarity Metric Learning

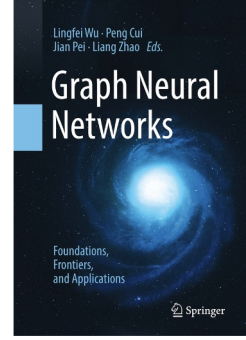
- We design a multi-head weighted cosine similarity metric function to learn a similarity matrix S for all pairs of nodes.

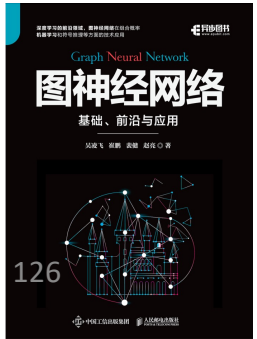
$$s_{ij}^k = \cos(\mathbf{W}_k \odot \mathbf{x}_i, \mathbf{W}_k \odot \mathbf{x}_j) \qquad s_{ij} = \frac{1}{m} \sum_{k=1}^{m} s_{ij}^k$$

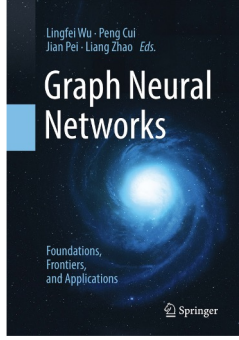
- We proceed to extract a symmetric sparse adjacency matrix from the similarity matrix S by considering only the ε-neighborhood for each node.

$$\mathbf{A}_{ij} = \begin{cases} \mathbf{S}_{ij} & \mathbf{S}_{ij} > \varepsilon \\ 0 & \text{otherwise} \end{cases} \qquad \tilde{\mathbf{A}} = \lambda \mathbf{L}_0 + (1-\lambda) \frac{\mathbf{A}_{ij}}{\sum_j \mathbf{A}_{ij}}$$

where $\mathbf{L}_0$ is the normalized adjacency matrix of the initial graph (or kNN-graph).







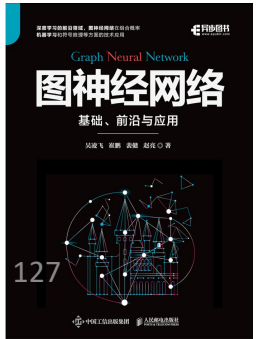
IDGL: Graph Regularization

- We adapt the techniques designed for learning graphs from smooth and apply them as regularization for controlling smoothness, connectivity and sparsity

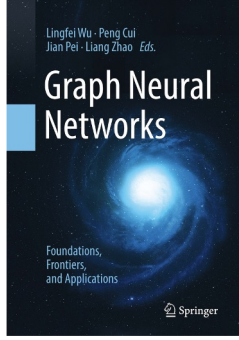
$$\Omega(\mathbf{A}, \mathbf{X}) = \frac{1}{2}\sum_{i,j}\mathbf{A}_{ij}||\mathbf{x}_i - \mathbf{x}_j||^2 = \mathrm{tr}(\mathbf{X}^T\mathbf{L}\mathbf{X})$$ Smoothness

$$f(\mathbf{A}) = -\beta\mathbf{1}^T\log(\mathbf{A}\mathbf{1}) + \gamma||\mathbf{A}||_F^2$$ Connectivity & sparsity

$$\mathcal{L}_{\mathcal{G}} = \alpha\Omega(\mathbf{A}, \mathbf{X}) + f(\mathbf{A})$$ Graph regularization loss

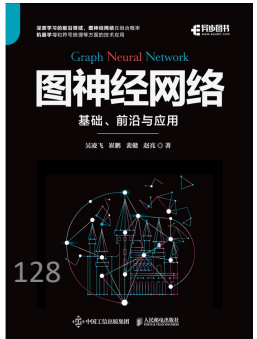


IDGL: Iterative Method for Joint Graph Structure and Representation Learning



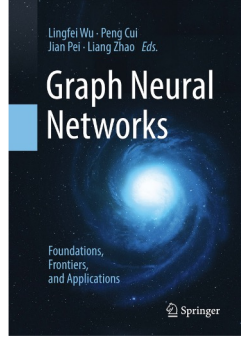
- Iterative method repeatedly
  - refines the adjacency matrix with the updated node embeddings
  - refines the node embeddings with the updated adjacency matrix
- Iterative procedure dynamically stops
  - the learned adjacency matrix converges with certain threshold
  - the maximal number of iterations is reached

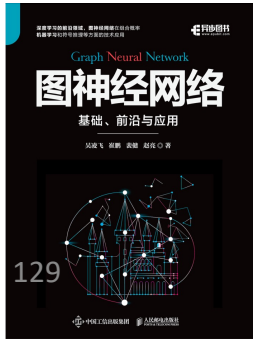
while $(t == 0 \quad or \quad ||\mathbf{A}^{(t)} - \mathbf{A}^{(t-1)}||_F^2 > \delta||\mathbf{A}^{(0)}||_F^2)$ **do**
  $t \leftarrow t + 1$
  $\mathbf{A}^{(t)}, \tilde{\mathbf{A}}^{(t)} \leftarrow \{\mathbf{Z}^{(t-1)}, \mathbf{A}_0\}$ using Eqs. (2), (3) and (10)
  $\tilde{\mathbf{A}}^{(t)} \leftarrow \eta\tilde{\mathbf{A}}^{(t)} + (1-\eta)\tilde{\mathbf{A}}^{(0)}$
  $\mathbf{Z}^{(t)} \leftarrow \{\tilde{\mathbf{A}}^{(t)}, \mathbf{X}\}$ using Eq. (7)
  $\hat{\mathbf{y}} \leftarrow \{\tilde{\mathbf{A}}^{(t)}, \mathbf{Z}^{(t)}\}$ using Eq. (8)
  $\mathcal{L}_{\text{pred}}^{(t)} \leftarrow \{\hat{\mathbf{y}}, \mathbf{y}\}$ using Eq. (9)
  $\mathcal{L}_{\mathcal{G}}^{(t)} \leftarrow \{\mathbf{A}^{(t)}, \mathbf{X}\}$ using Eqs. (4)–(6)
  $\mathcal{L}^{(t)} \leftarrow \mathcal{L}_{\text{pred}}^{(t)} + \mathcal{L}_{\mathcal{G}}^{(t)}$
end



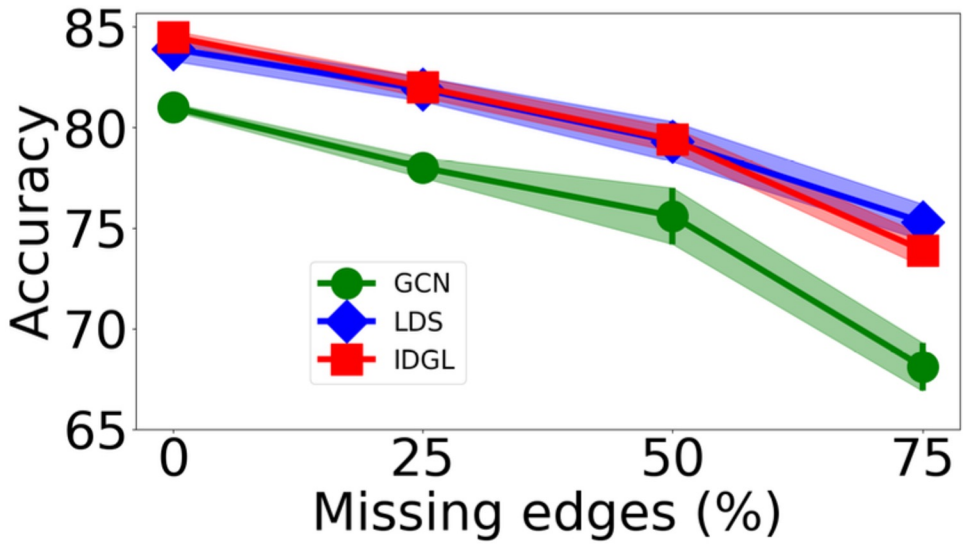
Results (Transudative Setting)

Model	Cora	Citeseer	Pubmed	ogbn-arxiv	Wine	Cancer	Digits
GCN	81.5	70.3	79.0	71.7 (0.3)	—	—	—
GAT	83.0 (0.7)	72.5 (0.7)	79.0 (0.3)	—	—	—	—
GraphSage	77.4 (1.0)	67.0 (1.0)	76.6 (0.8)	71.5 (0.3)	—	—	—
APPNP	—	**75.7 (0.3)**	79.7 (0.3)	—	—	—	—
H-GCN	**84.5 (0.5)**	72.8 (0.5)	79.8 (0.4)	—	—	—	—
GCN+GDC	83.6 (0.2)	73.4 (0.3)	78.7 (0.4)	—	—	—	—
LDS	84.1 (0.4)	75.0 (0.4)	—	—	97.3 (0.4)	94.4 (1.9)	92.5 (0.7)
GCN_{kNN}*	—	—	—	—	95.9 (0.9)	94.7 (1.2)	89.5 (1.3)
LDS*	83.9 (0.6)	74.8 (0.3)	—	—	96.9 (1.4)	93.4 (2.4)	90.8 (2.5)
IDGL	**84.5 (0.3)**	74.1 (0.2)	—	—	97.8 (0.6)	**95.1 (1.0)**	93.1 (0.5)
IDGL-ANCH	84.4 (0.2)	72.0 (1.0)	**82.7 (0.4)**	**72.0 (0.3)**	**98.1 (1.1)**	94.8 (1.4)	**93.2 (0.9)**

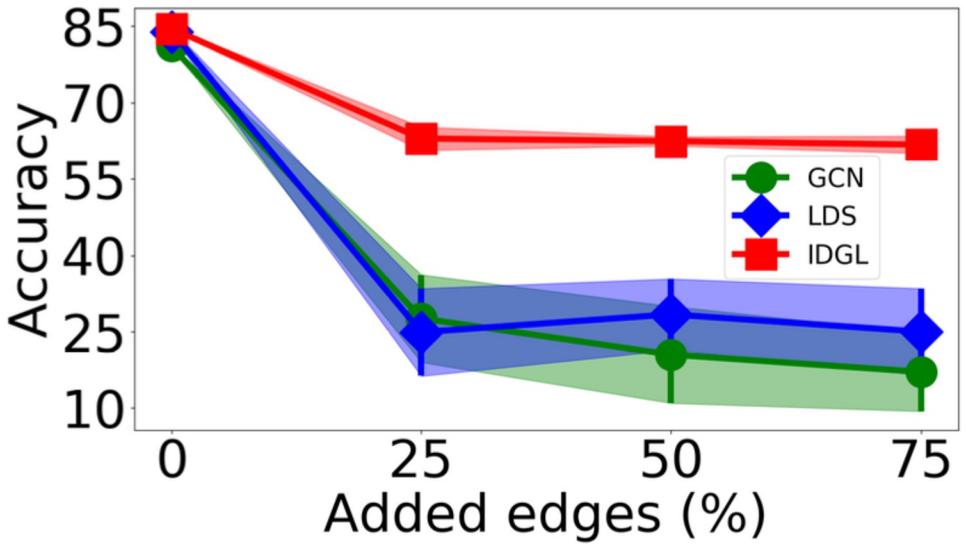




Results (Robustness to Missing/Adding Edges)



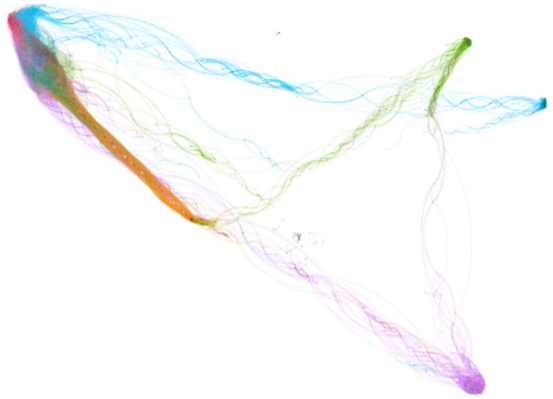
(a) Edge deletion



(b) Edge addition

Visualization of the Initial and Learned Graphs

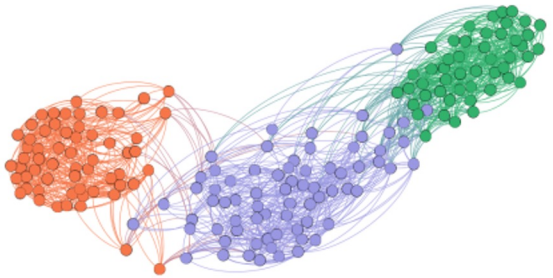
(a) Initial graph ($\mathbf{A}^{(0)}$)



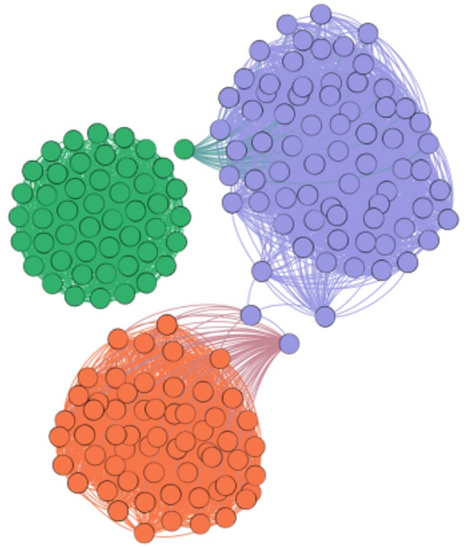
(b) Learned graph ($\mathbf{A}^{(t)}$)

Dataset: Cora (having initial graph data)

Dataset: Wine (having no initial graph data)



(a) kNN graph ($\mathbf{A}^{(0)}$)



(b) Learned graph ($\mathbf{A}^{(t)}$)

GNNs: Graph Classification (Chapter 9)

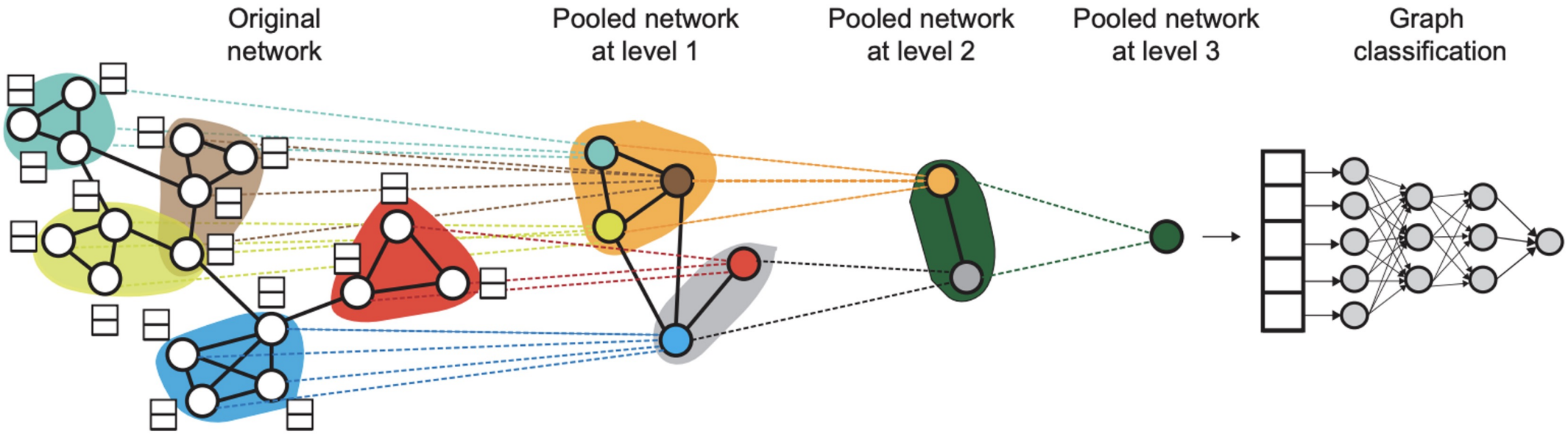
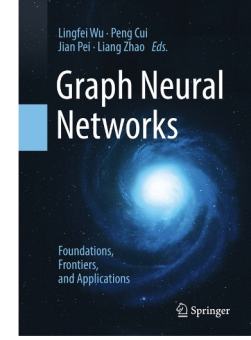
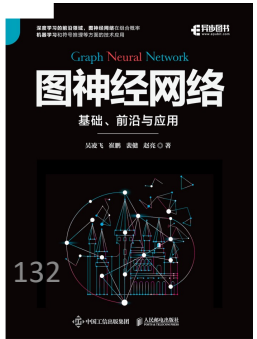
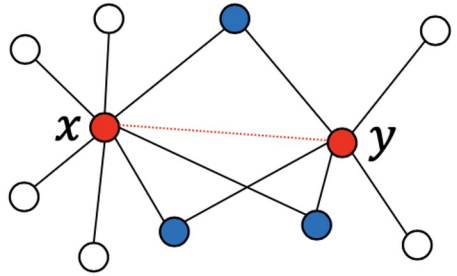


Image credit: Rex Ying et al., “Hierarchical Graph Representation Learning with Differentiable Pooling”, NeurIPS 2018.

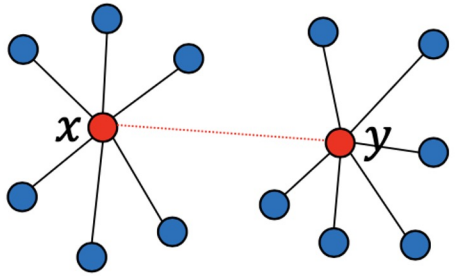




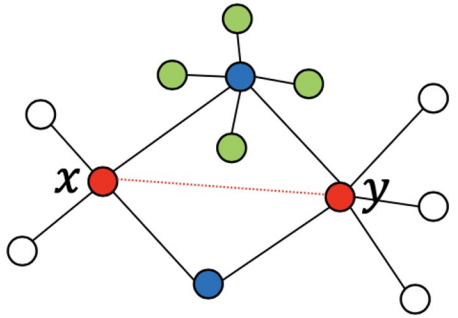
GNNs: Link Prediction (Chapter 10)



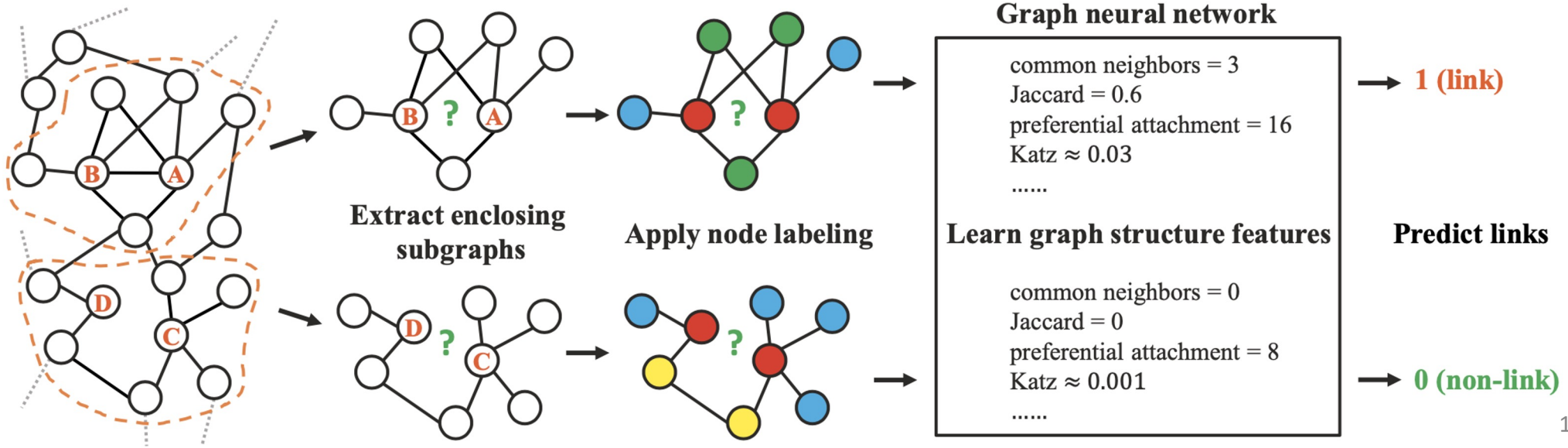
common neighbors (CN):
$|\Gamma(x) \cap \Gamma(y)|$



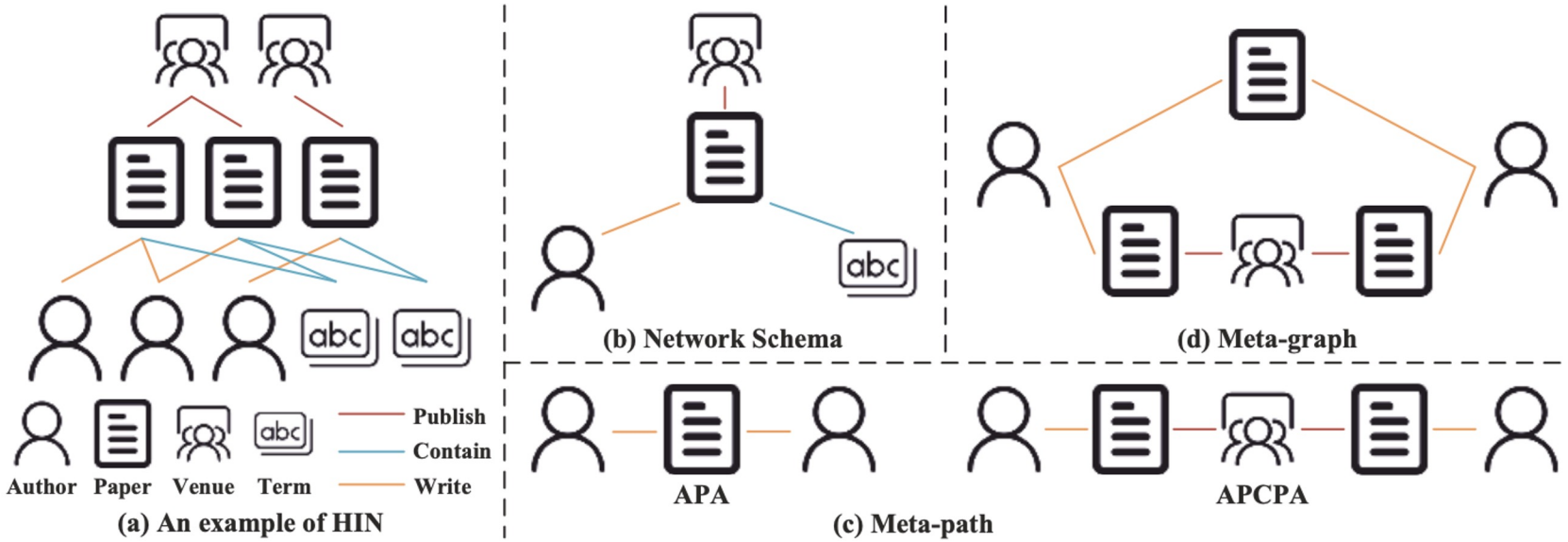
preferential attachment (PA):
$|\Gamma(x)| \cdot |\Gamma(y)|$

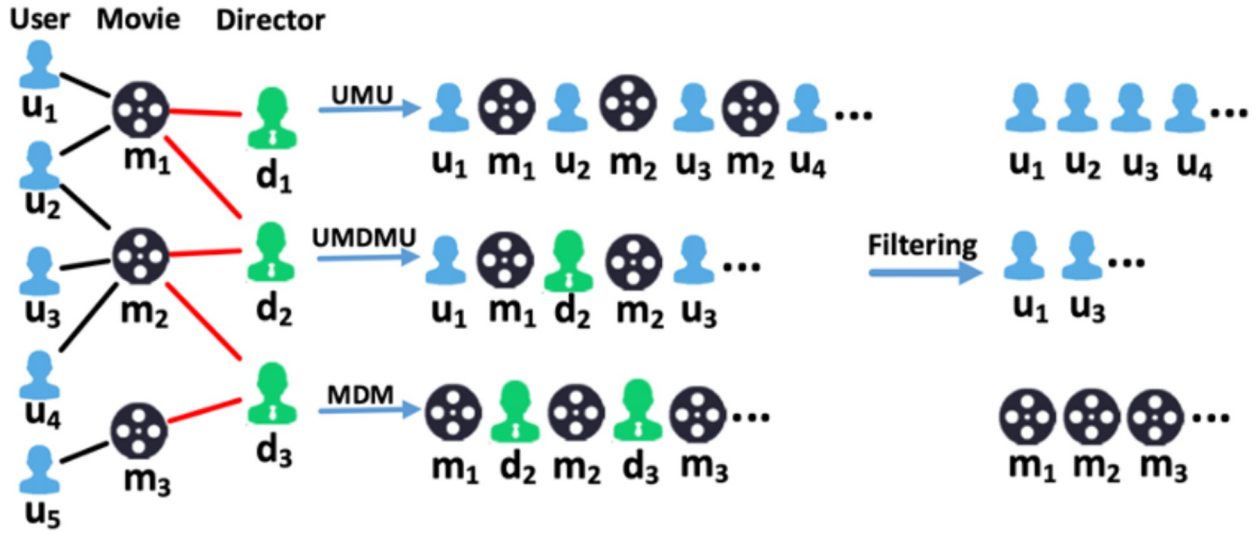


Adamic-Adar (AA):
$\sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{\log |\Gamma(z)|}$

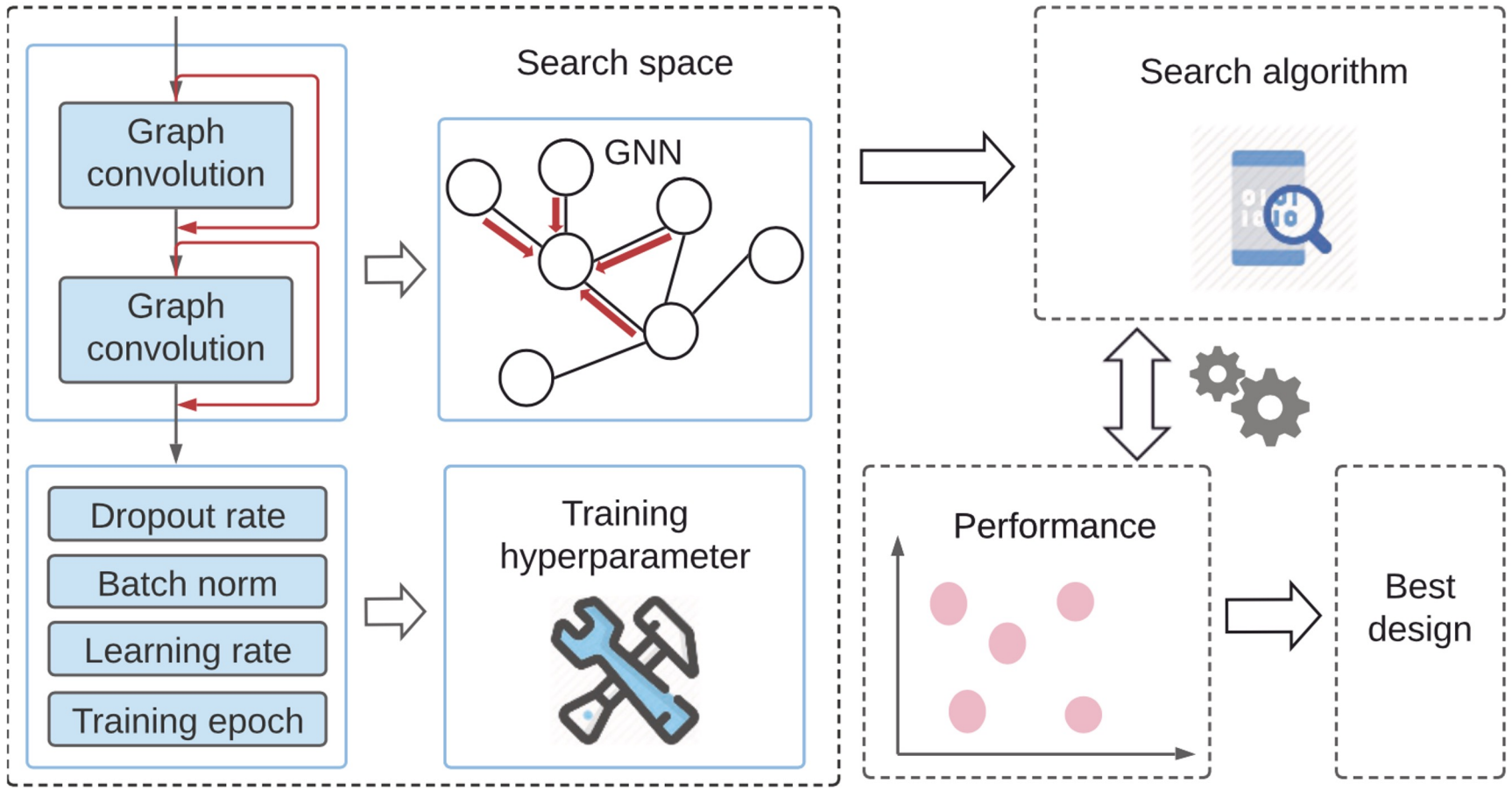


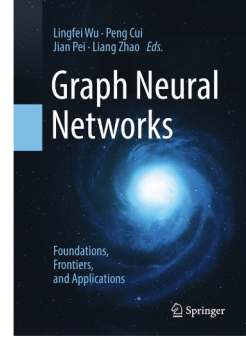
Heterogeneous GNNs (Chapter 16)

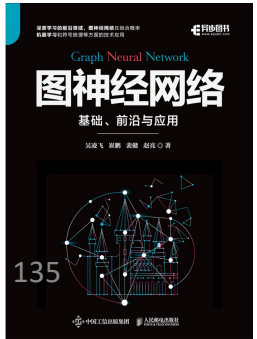




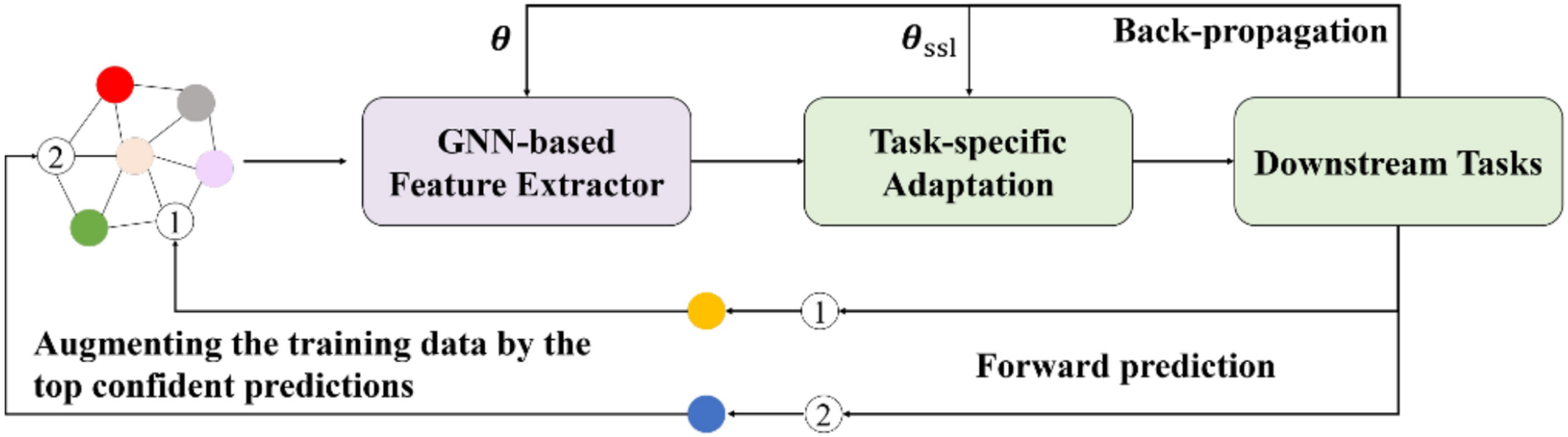
GNNs: AutoML (Chapter 17)







GNNs: Self-supervised Learning (Chapter 18)



Outline

GNNs Foundations

- Time History of GNNs
- GNNs: Foundations and Models
- GNNs: Theory, Scalability, Interpretability

GNNs Frontiers

- Graph Generation and Transformation
- Dynamic Graph Neural Networks
- Graph Matching
- Graph Structure Learning

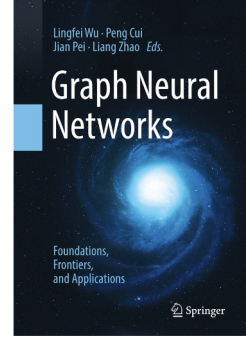
GNNs Applications

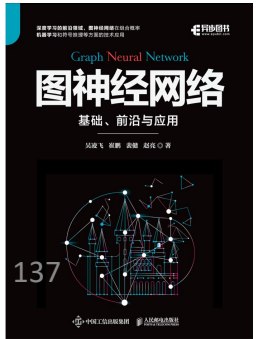
- GNNs in Recommendation
- GNNs in Natural Language Processing
- GNNs in Program Analysis
- GNNs in Protein Modeling

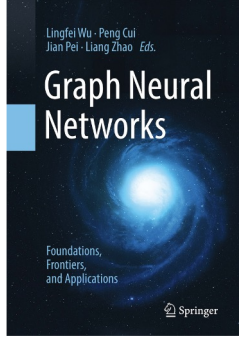
GNN book website：
https://graph-neural-networks.github.io/index.html

Amazon：
https://www.amazon.com/Graph-Neural-Networks-Foundations-Applications/dp/9811660530

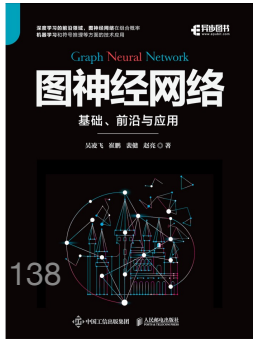
JD.com (京东商城)：
https://item.jd.com/13536841.html

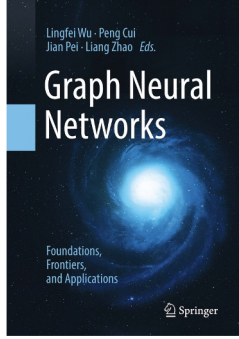




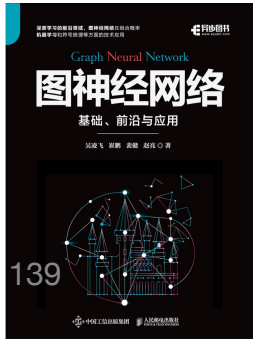


GNNs
Applications



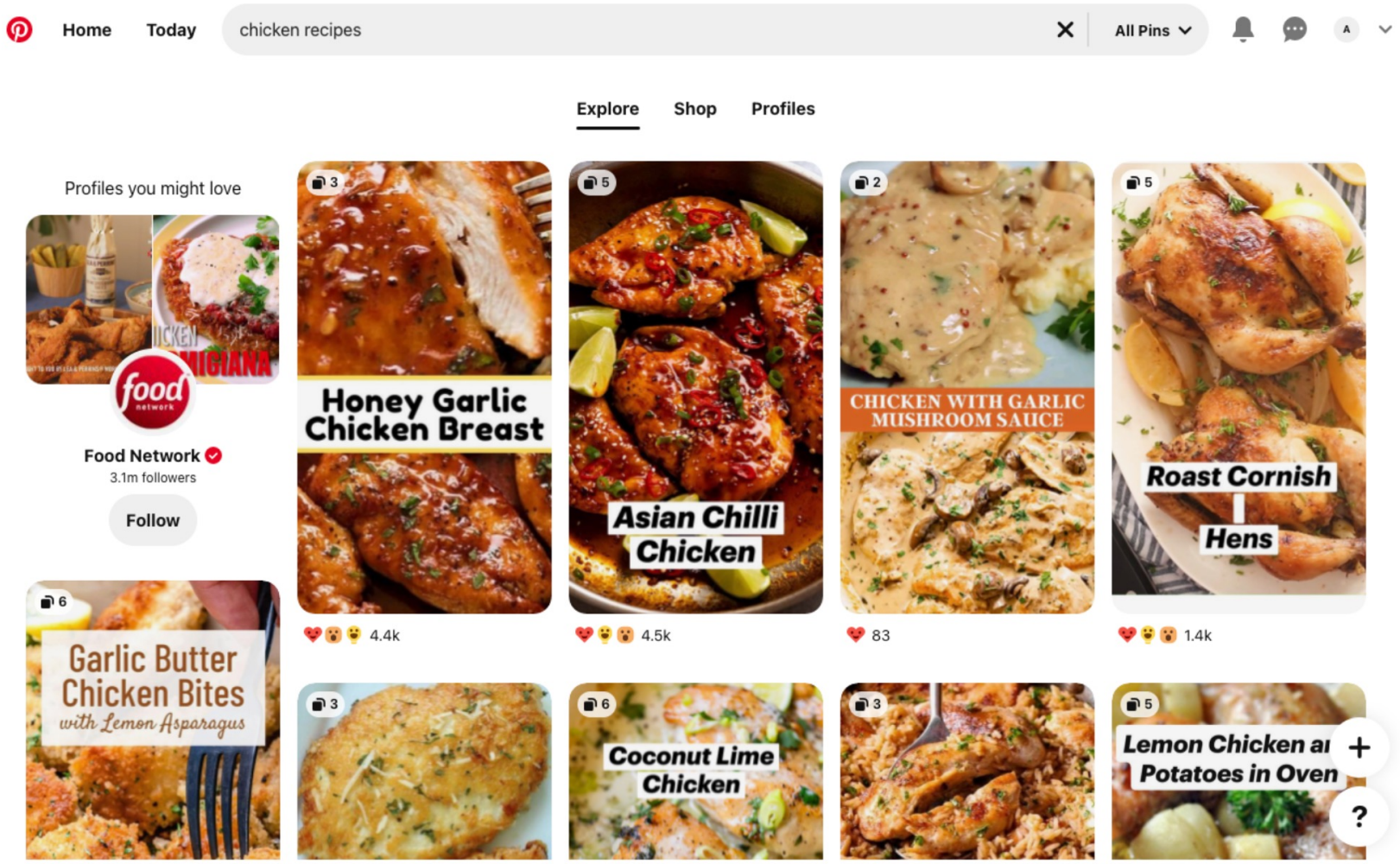


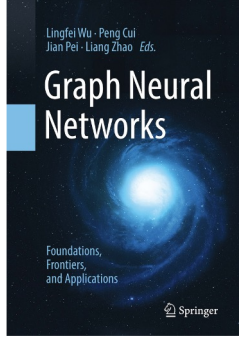
GNNs in Recommendation (Chapter 19)

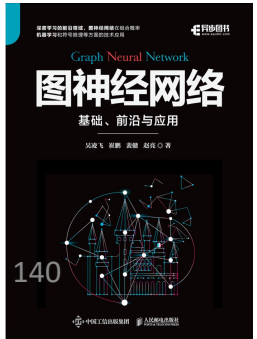


Pinterest: Content Understanding

Bring everyone the inspiration to create a life they love!



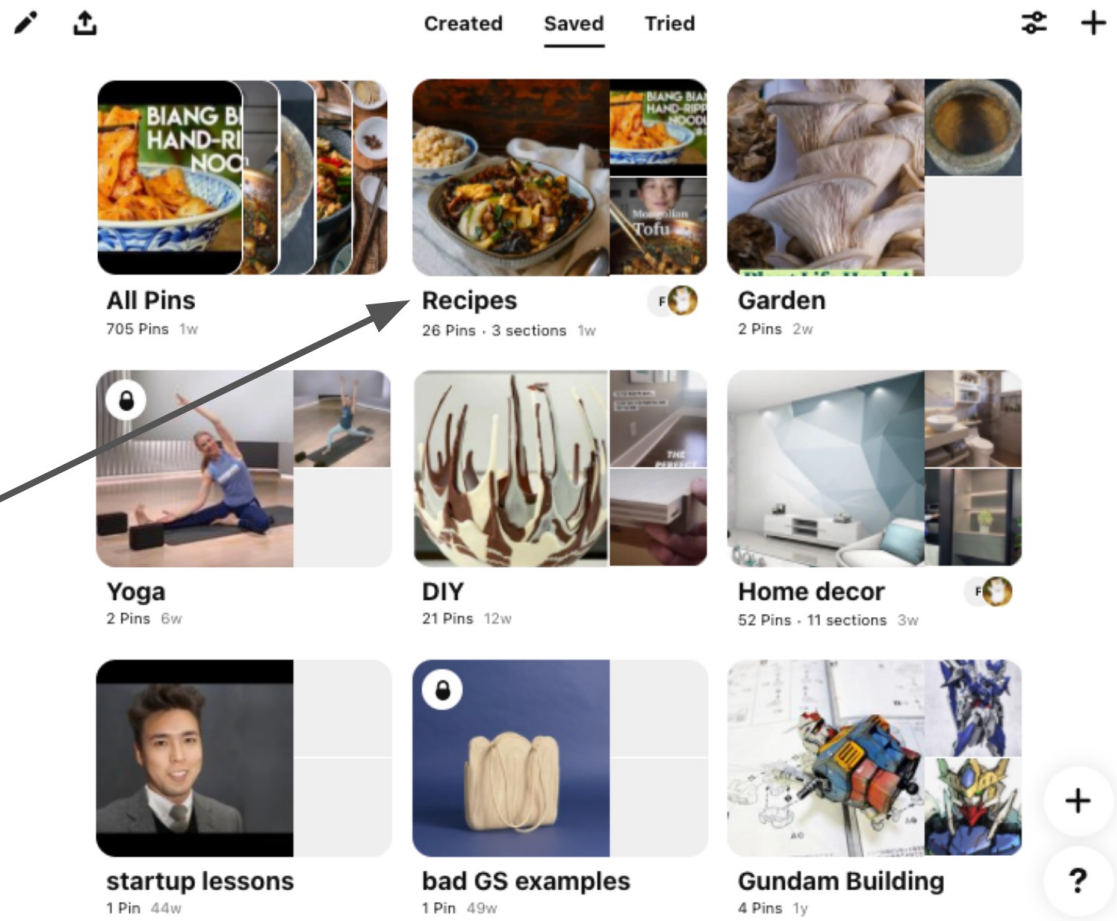


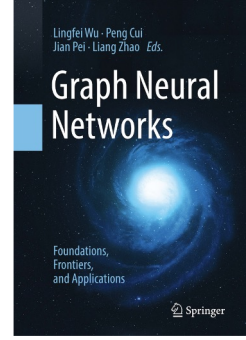


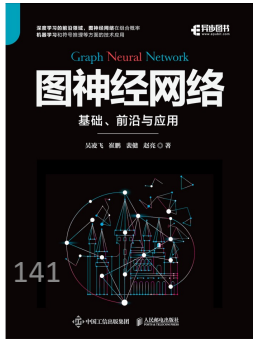
Pinterest: Pin and Board

- 300B Pins saved
- 6 B+ Boards

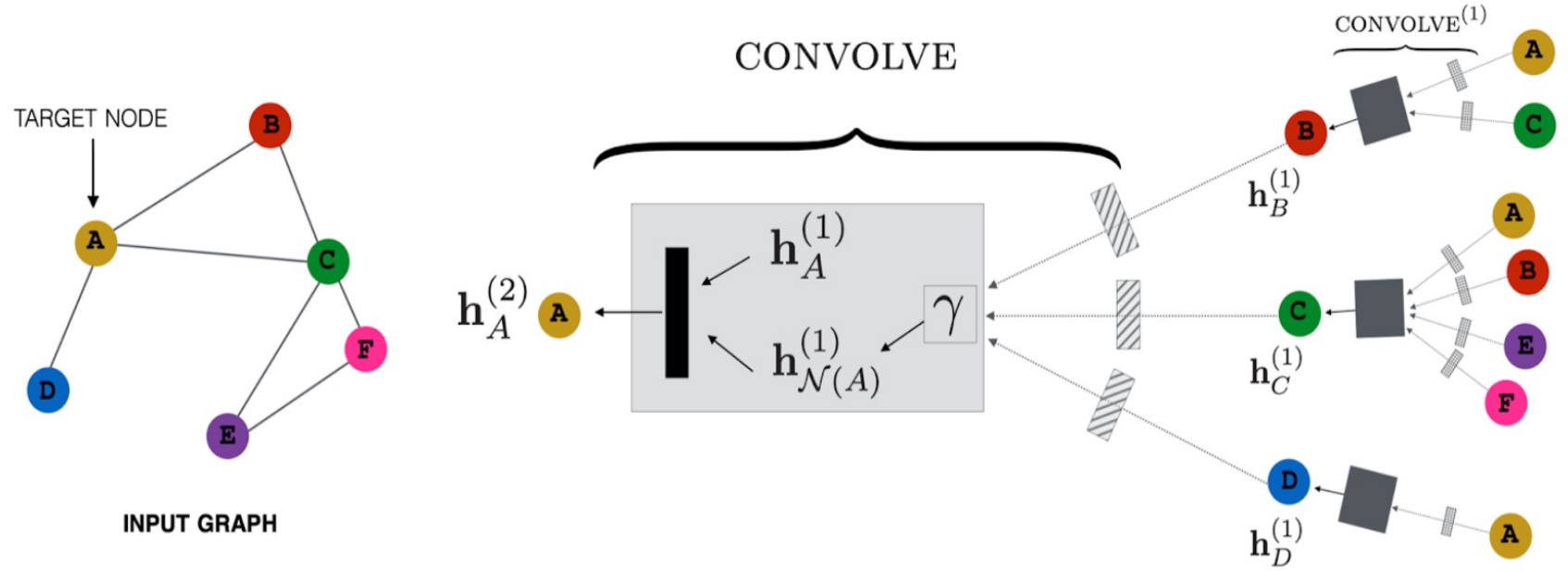
save



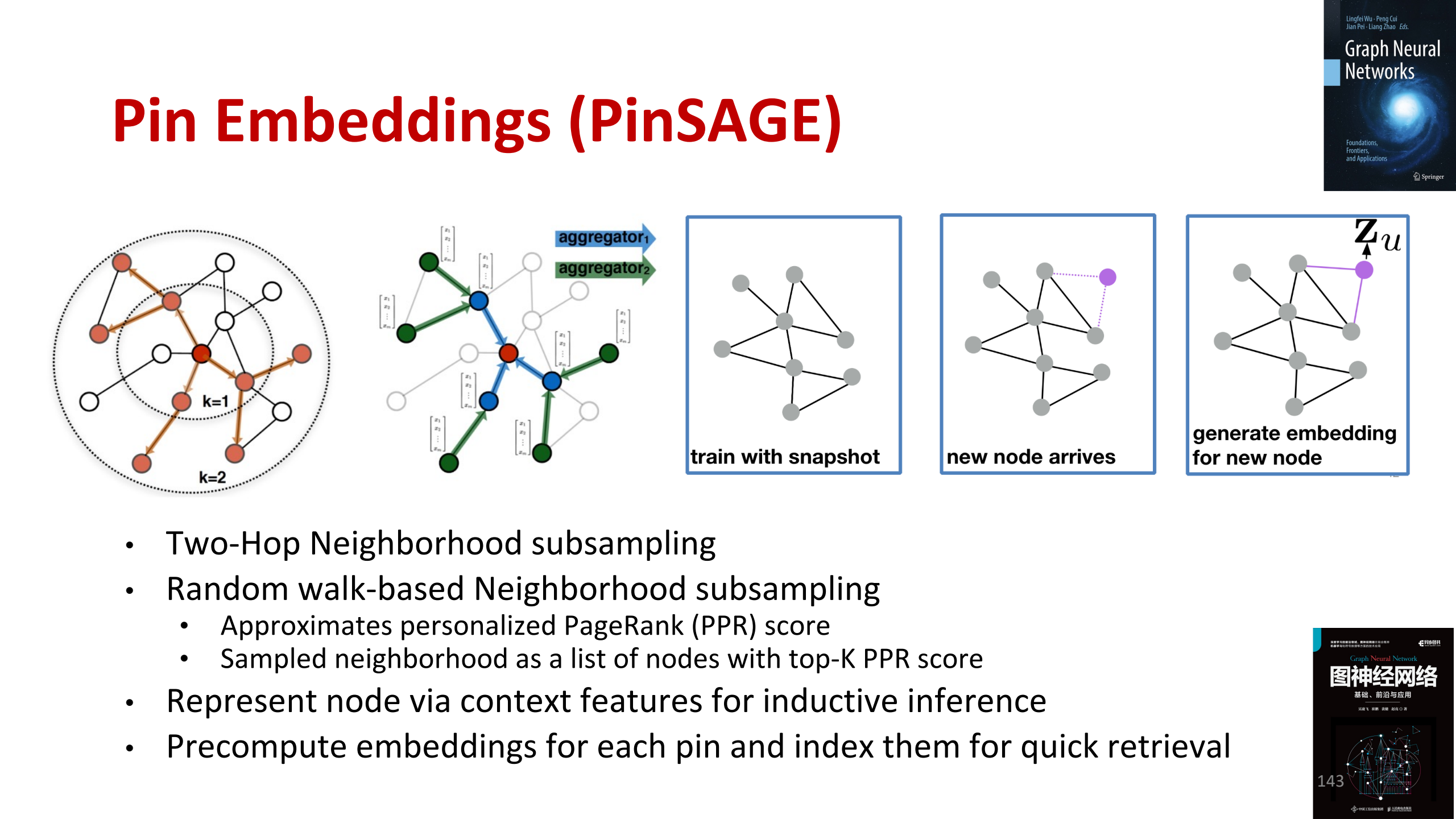




Pin Embeddings (PinSAGE)

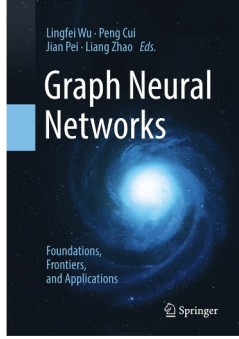


- A Pin is defined by its content and engagement
  - Raw features (visual & text embeddings) to represent content
  - Pin-board graph to represent engagement
- Many use-cases such as retrieval, ranking, search, recommendation, etc.

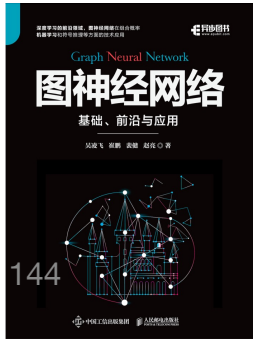


Pin Embeddings (PinSAGE)

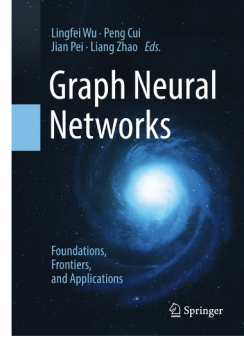
- Two-Hop Neighborhood subsampling
- Random walk-based Neighborhood subsampling
  - Approximates personalized PageRank (PPR) score
  - Sampled neighborhood as a list of nodes with top-K PPR score
- Represent node via context features for inductive inference
- Precompute embeddings for each pin and index them for quick retrieval

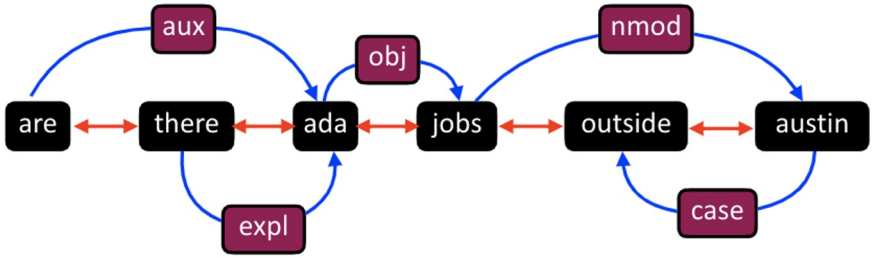


GNNs in Natural Language Processing (Chapter 21)

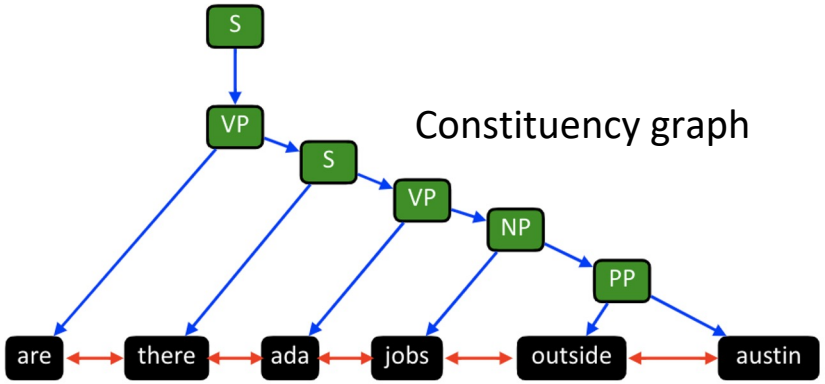


Graphs are ubiquitous in NLP As Well

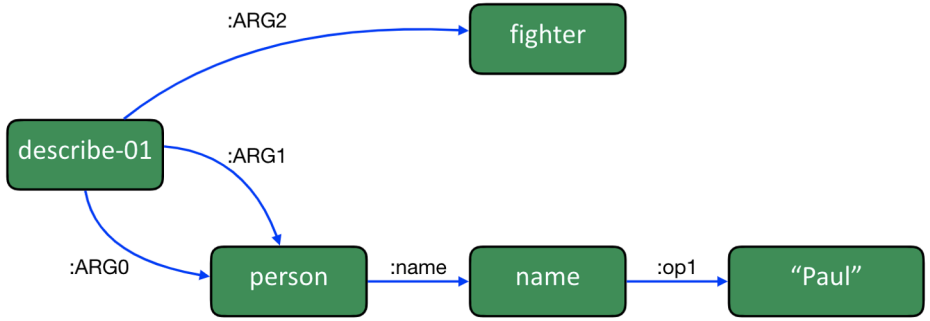




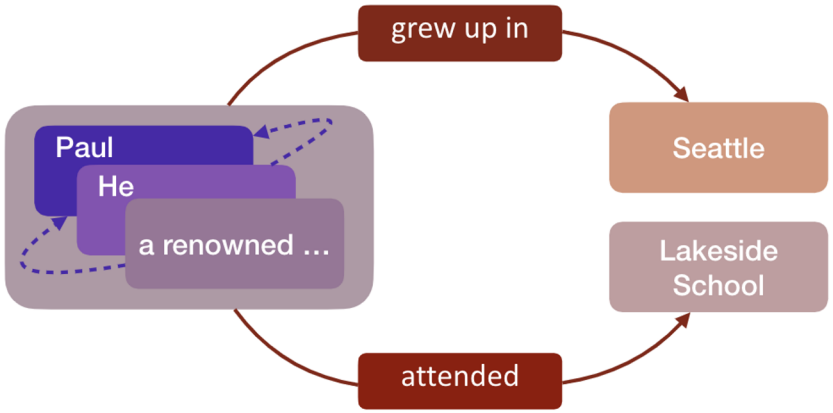
Dependency graph



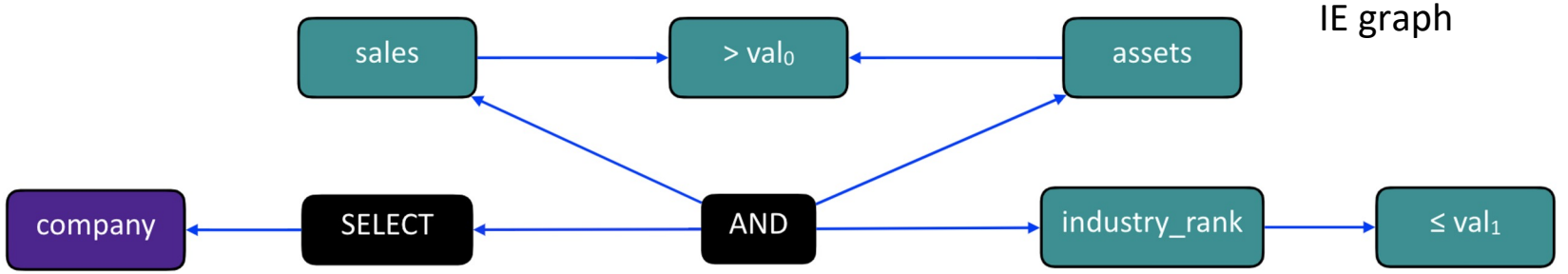
Constituency graph



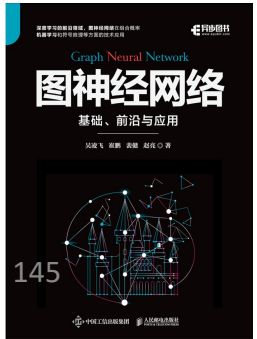
AMR graph



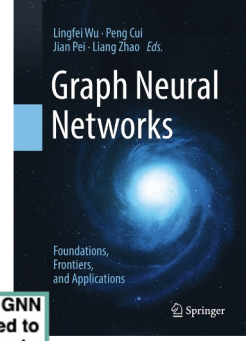
IE graph



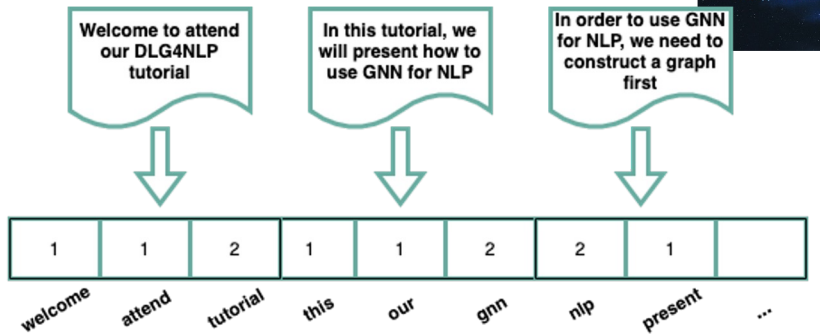
SQL graph



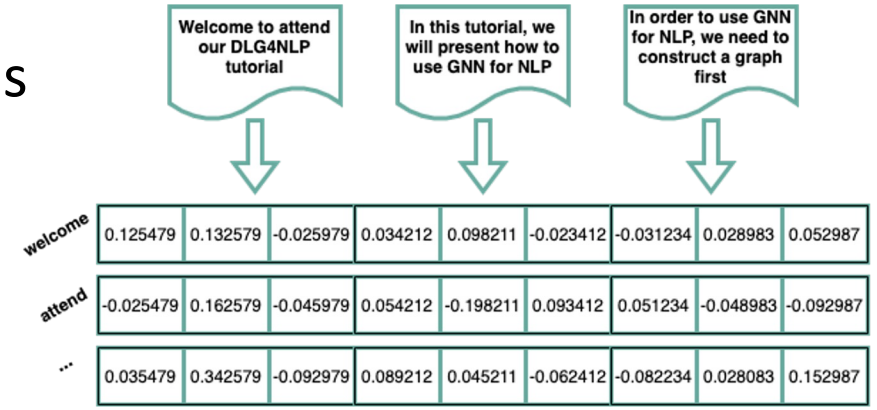
Natural Language Processing: A Graph Perspective



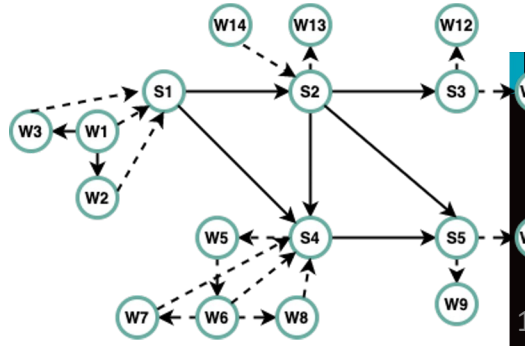
- Represent natural language as a bag of tokens
  - BOW, TF-IDF
  - Topic Modeling: text as a mixture of topics



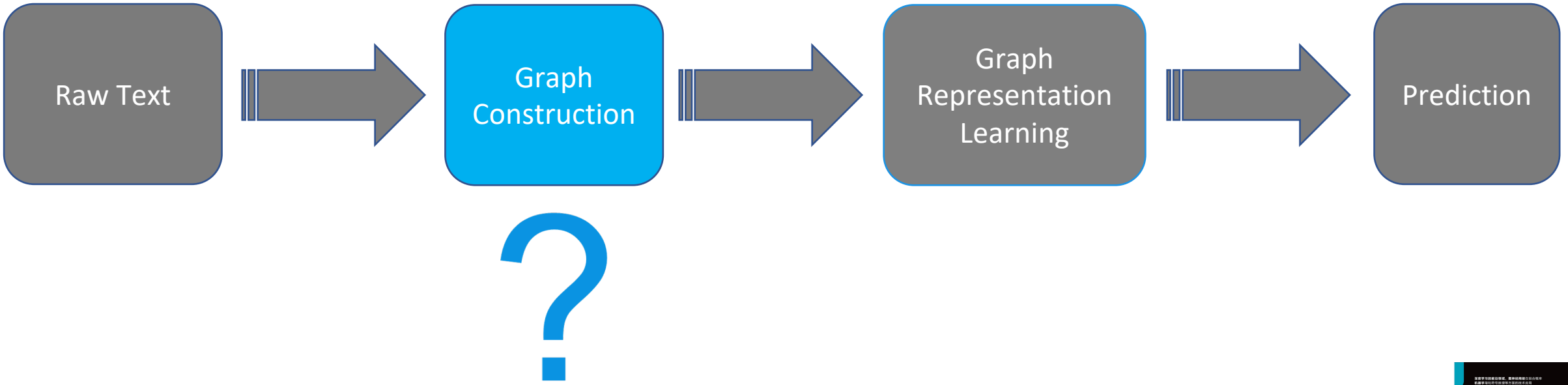
- Represent natural language as a sequence of tokens
  - Linear-chain CRF
  - Word2vec, Glove

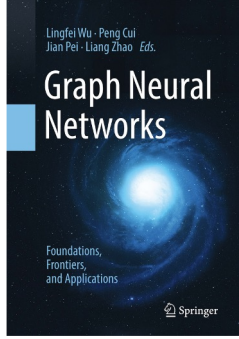


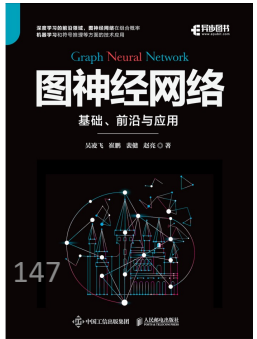
- Represent natural language as a graph
  - Dependency graphs, constituency graphs, AMR graphs, IE graphs, and knowledge graphs
  - Text graph containing multiple hierarchies of elements, i.e. document, sentence and word

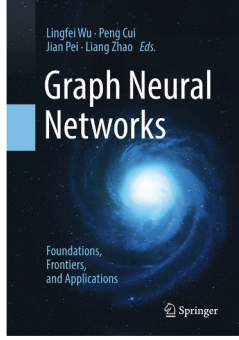


GNNs for Natural Language Processing



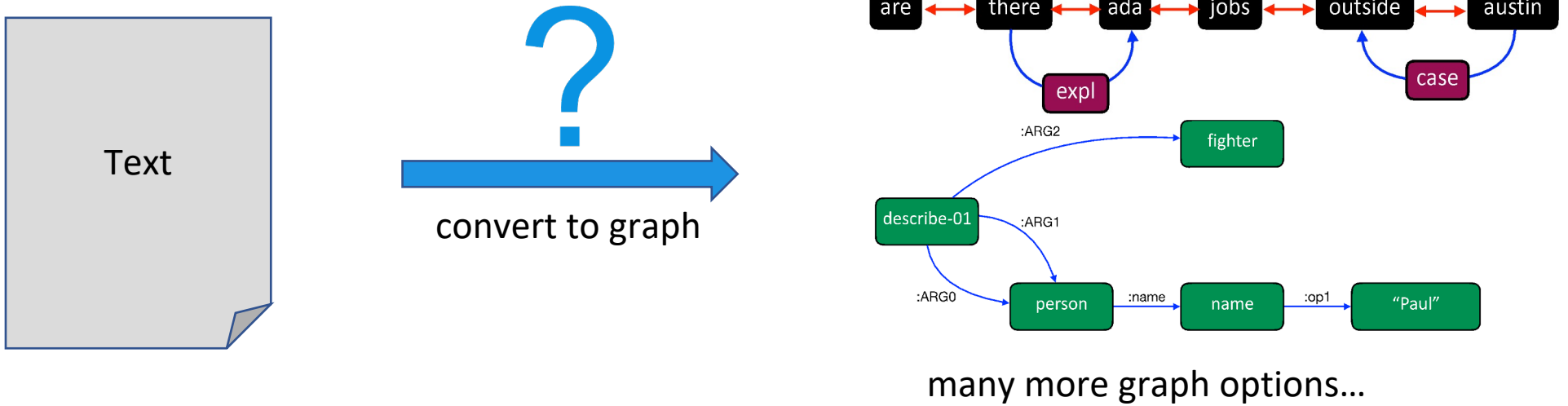


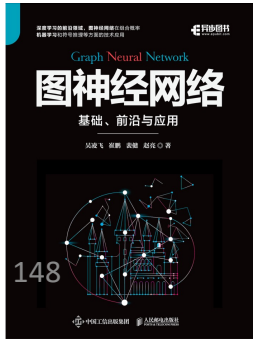


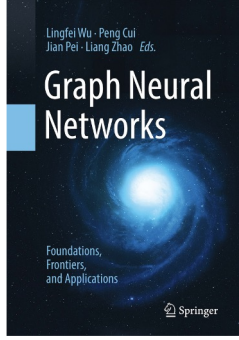


Why Graph Construction for NLP?

- Representation power: graph > sequence > bag
- Different NLP tasks require different aspects of text , e.g., syntax, semantics.
- Different graphs capture different aspects of the text
- Two categories: static vs dynamic graph construction
- Goal: good downstream task performa

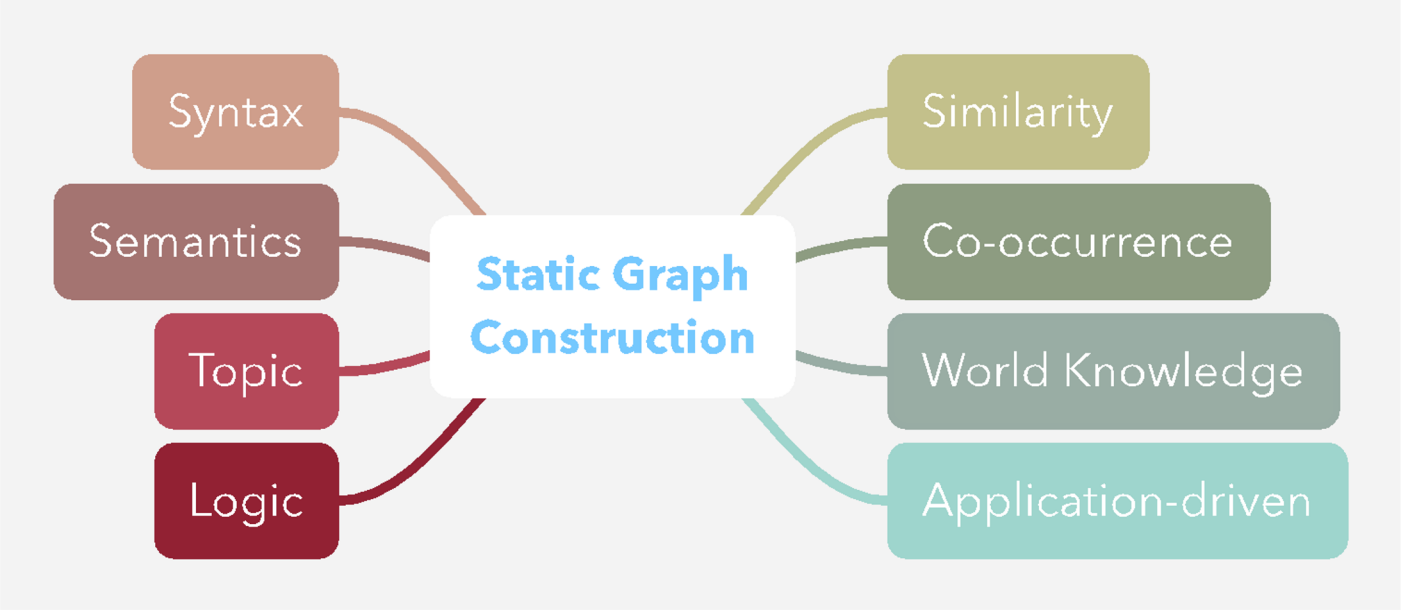


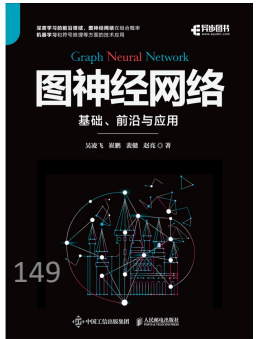




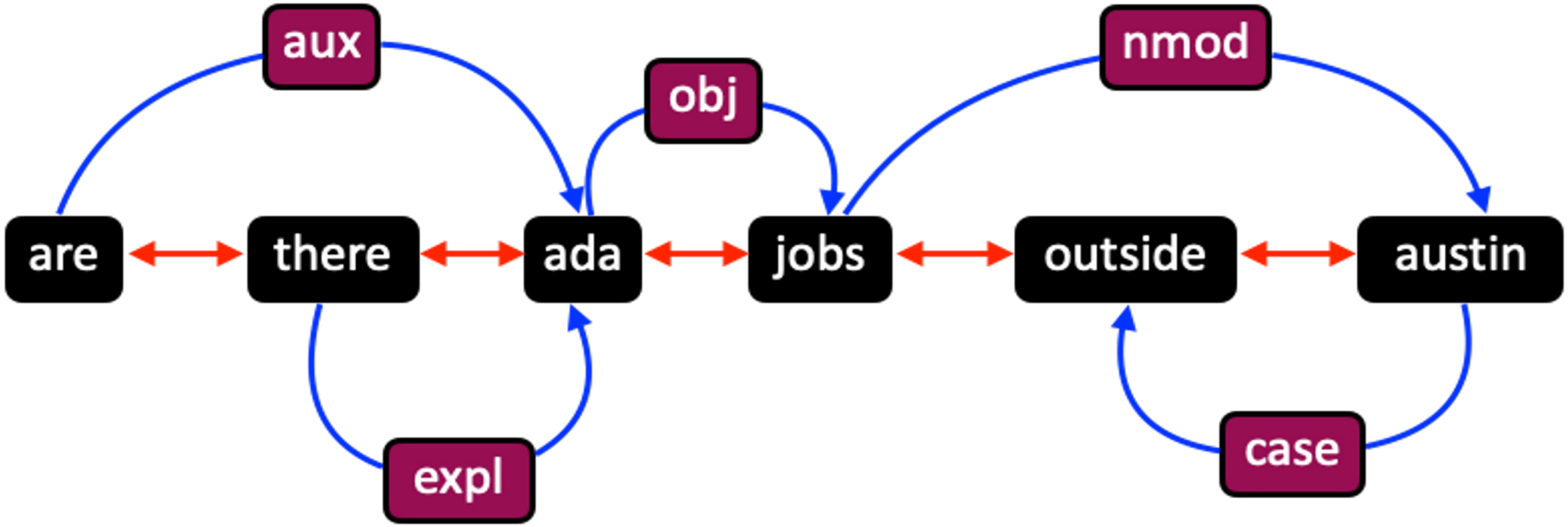
Static Graph Construction

- Problem setting:
  - Input: raw text (e.g., sentence, paragraph, document, corpus)
  - Output: graph
- Conducted during preprocessing by augmenting text with domain knowledge





Static Graph Construction: Dependency Graph

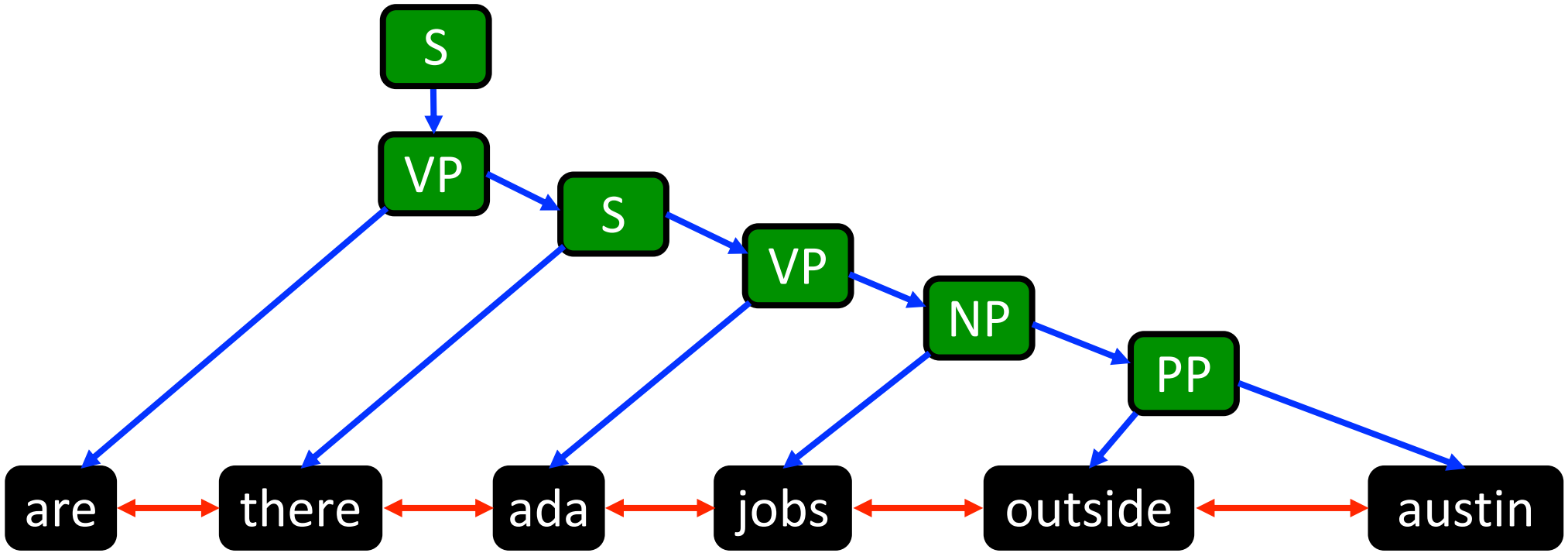


Dependency parsing

Text input: are there ada jobs outside austin

Add additional sequential edges to
1) reserve sequential information in raw text
2) connect multiple dependency graphs in a paragraph

Static Graph Construction: Constituency Graph

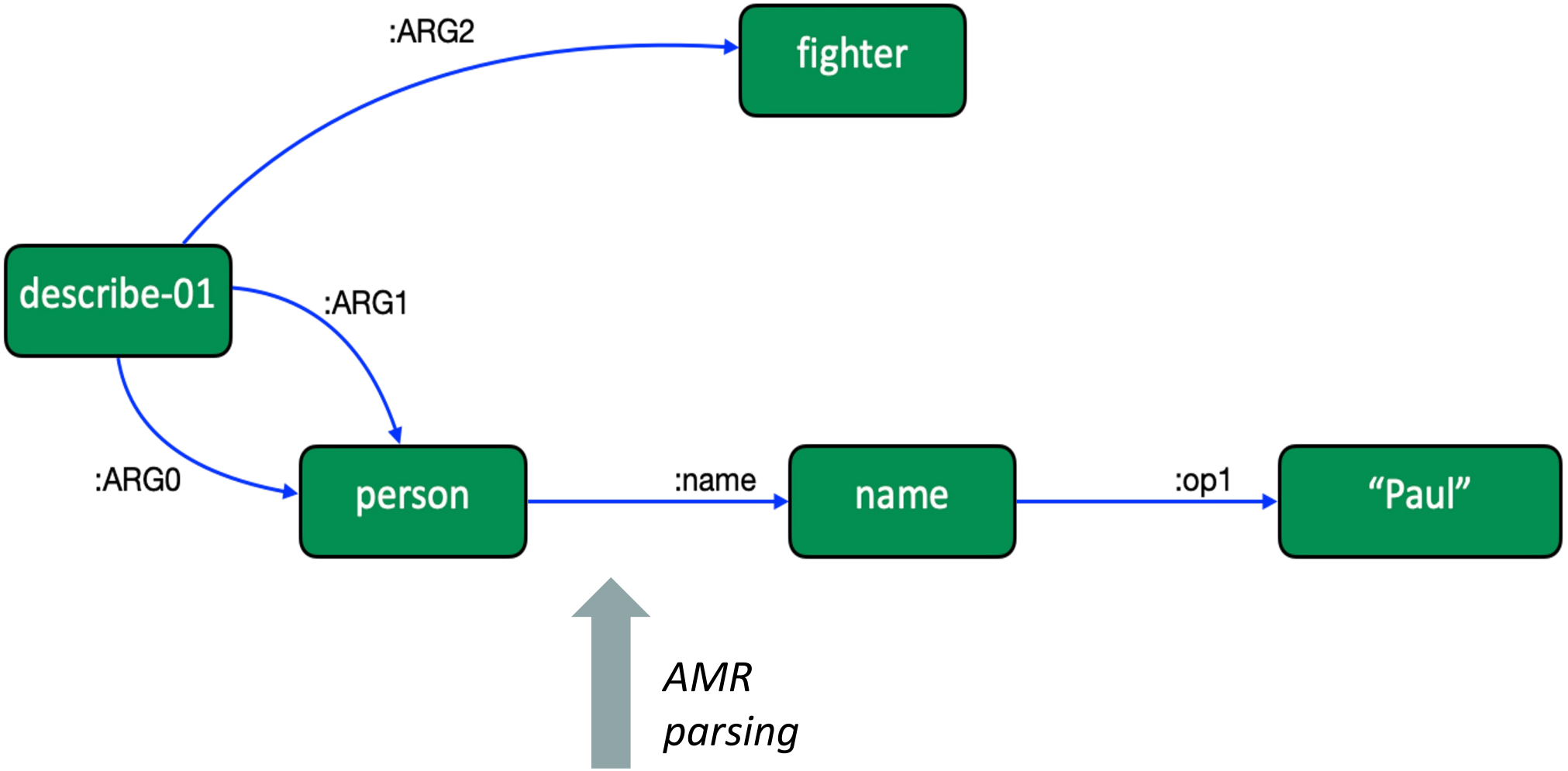


Constituency parsing

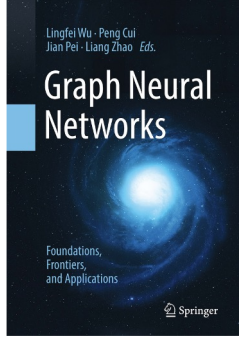
Again, add additional sequential edges

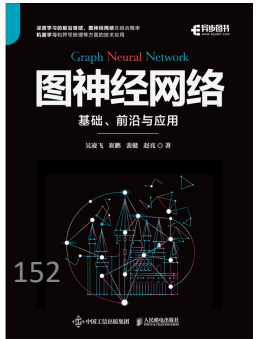
Text input: are there ada jobs outside austin

Static Graph Construction: AMR Graph

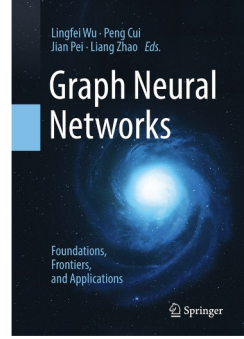


Text input: Paul's description of himself: a fighter





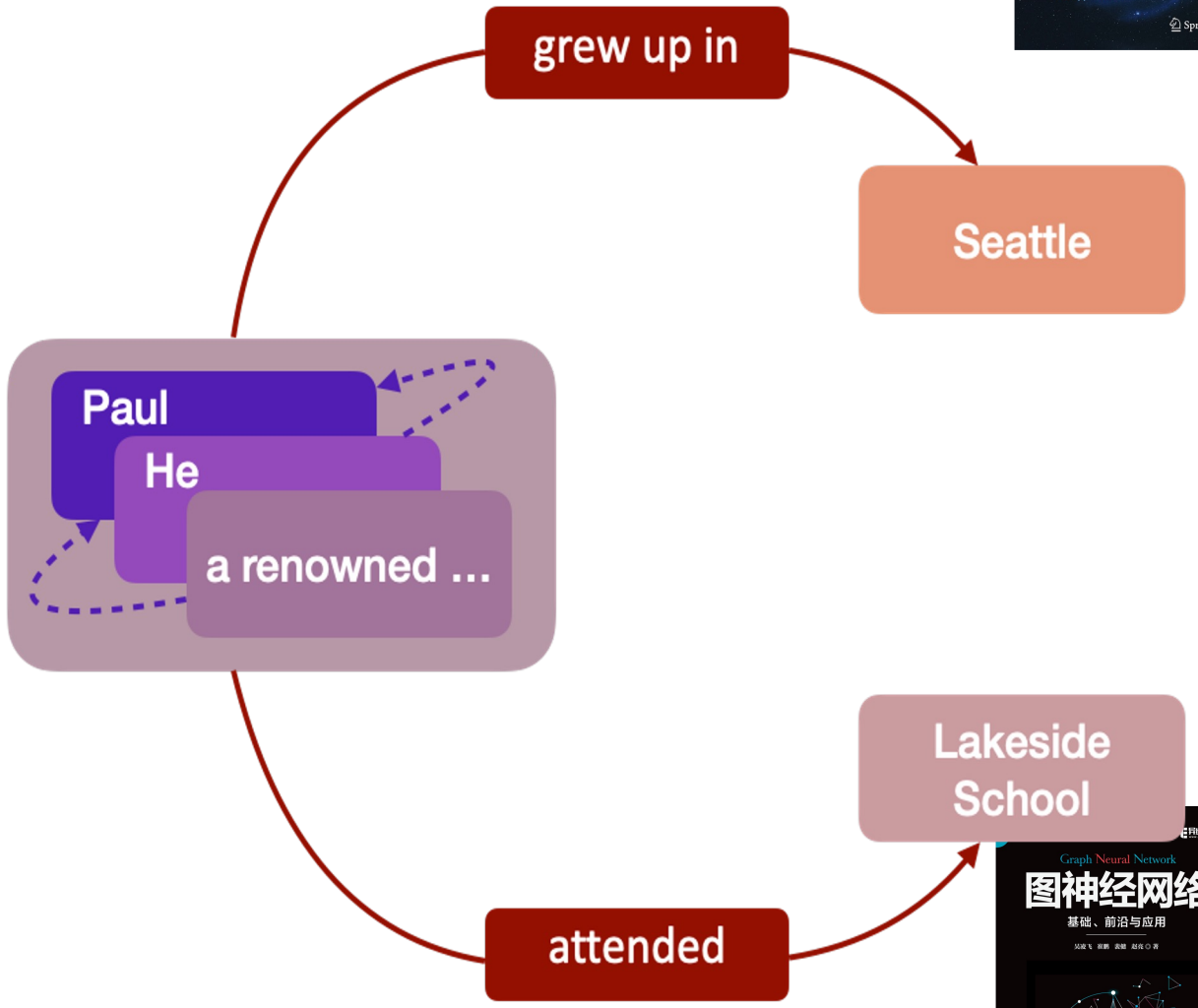
Static Graph Construction: IE Graph

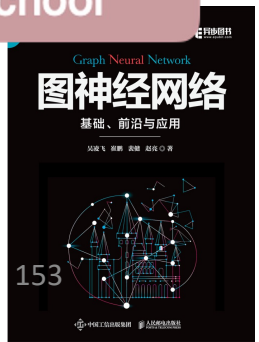


Text input: *Paul, a renowned computer scientist, grew up in Seattle. He attended Lakeside School.*

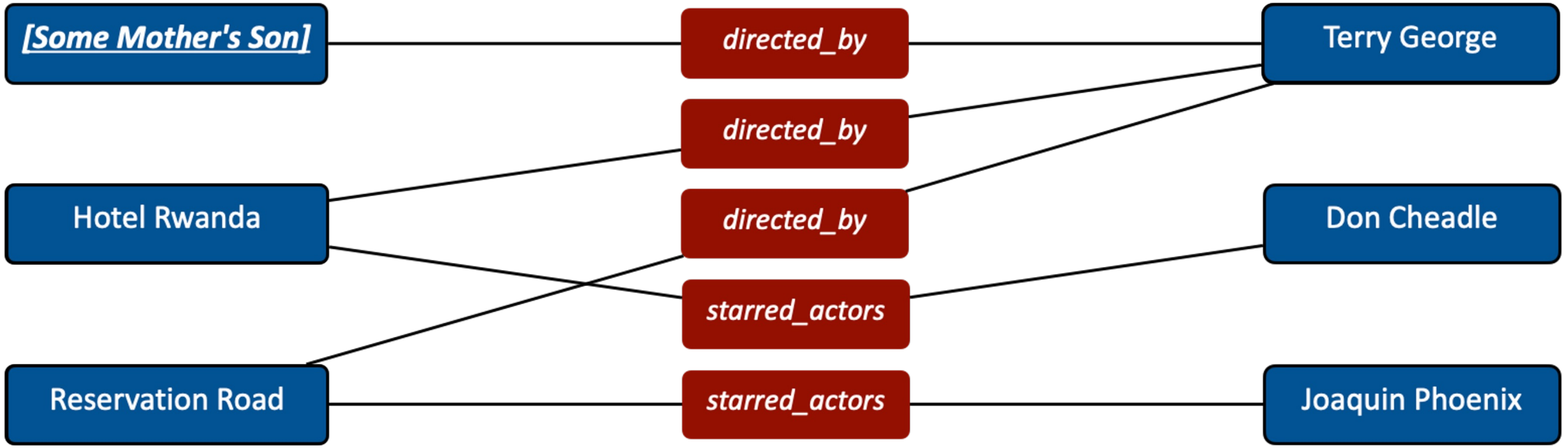
OpenIE

Coreference



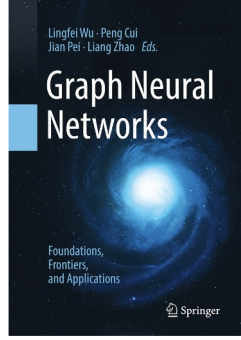


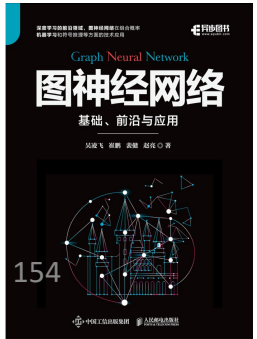
Static Graph Construction: Knowledge Graph



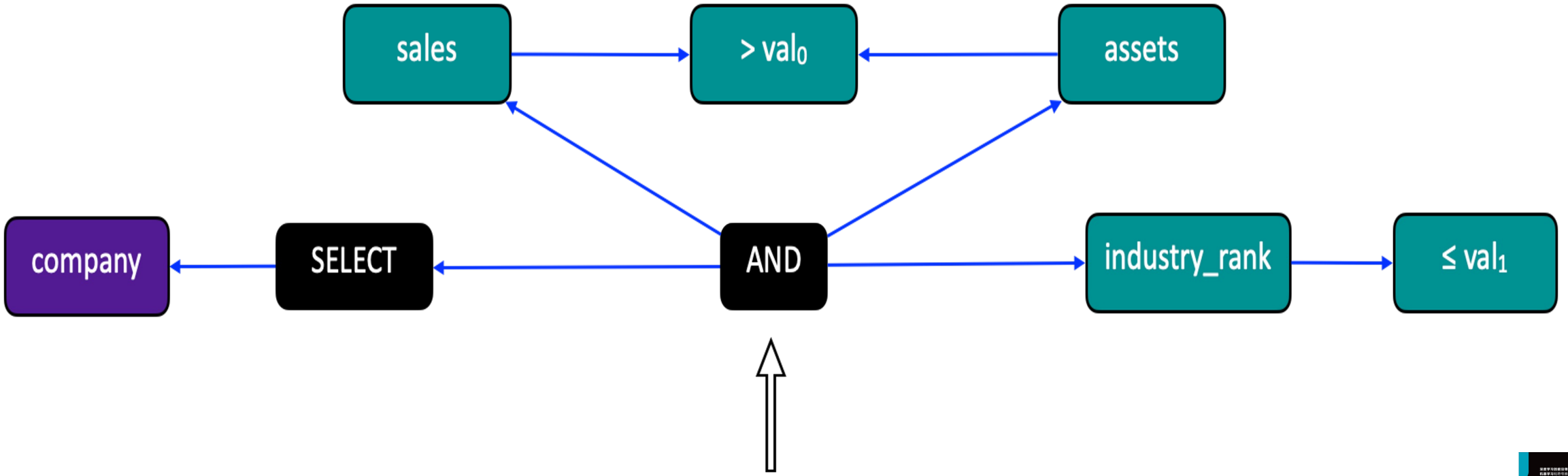
Get the concept sub-graph from KB

Question: *who acted in the movies directed by the director of* ***[Some Mother's Son]***
Answer: *Don Cheadle, Joaquin Phoenix*

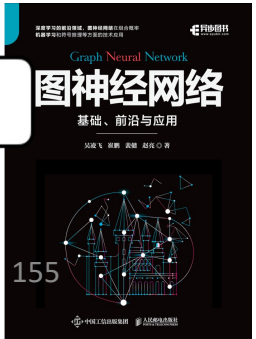




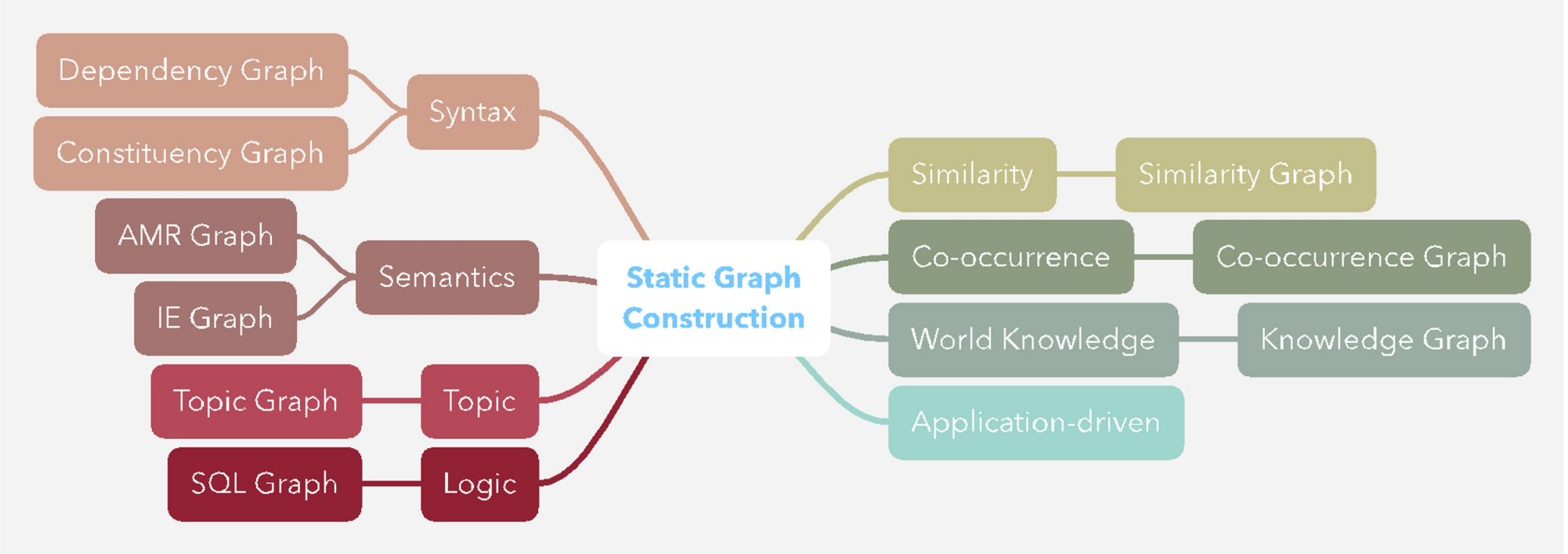
Static Graph Construction: SQL Graph



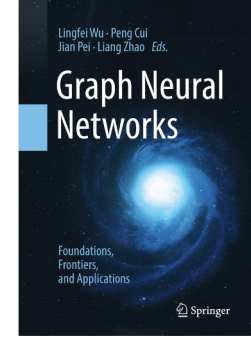
***SQL query input: SELECT** company **WHERE** assets > val_0 **AND** sales > val_0 **AND** industry_rank ≤ val_1*

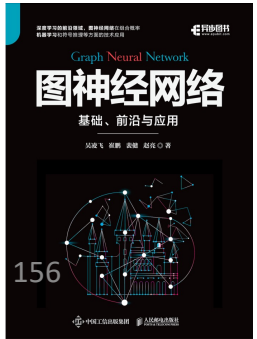


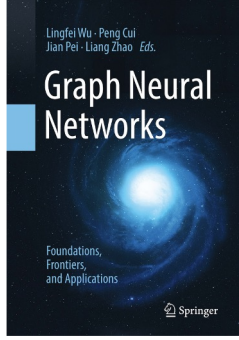
Static Graph Construction: Summary



Widely used in various NLP applications such as NLG, MRC, semantic parsing, etc.

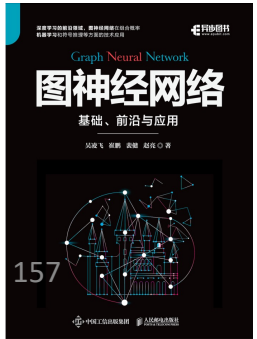


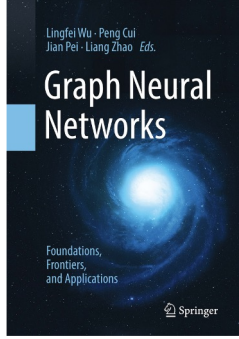




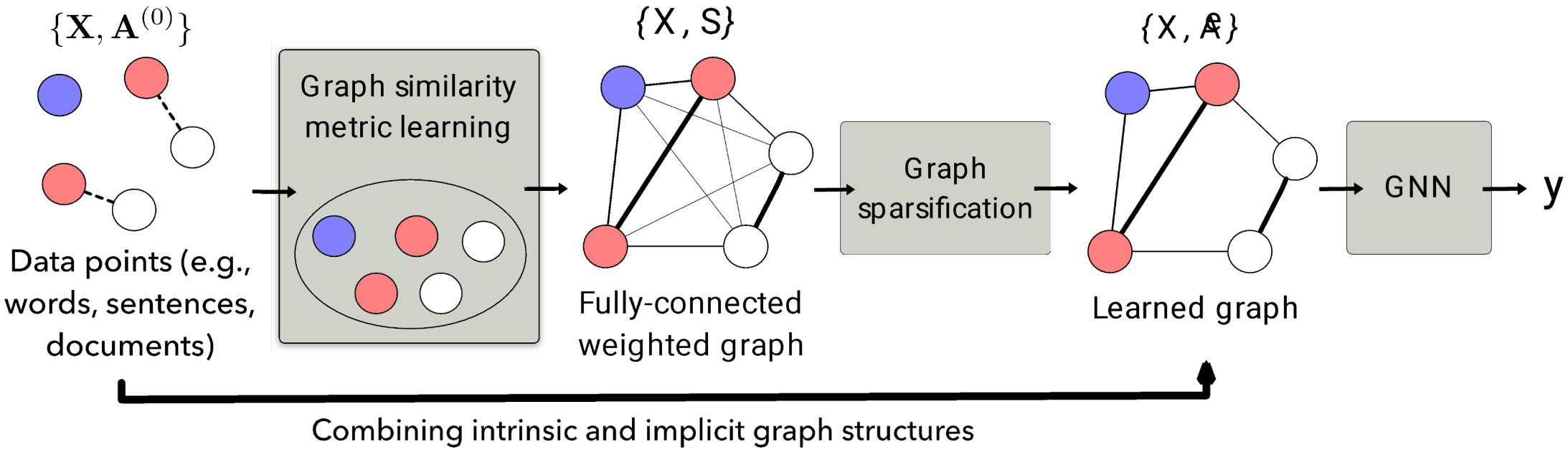
Dynamic Graph Construction

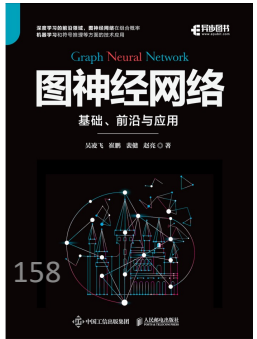
- Problem setting:
  - Input: raw text (e.g., sentence, paragraph, document, corpus)
  - Output: graph
- Graph structure (adjacency matrix) learning on the fly, joint with graph representation learning



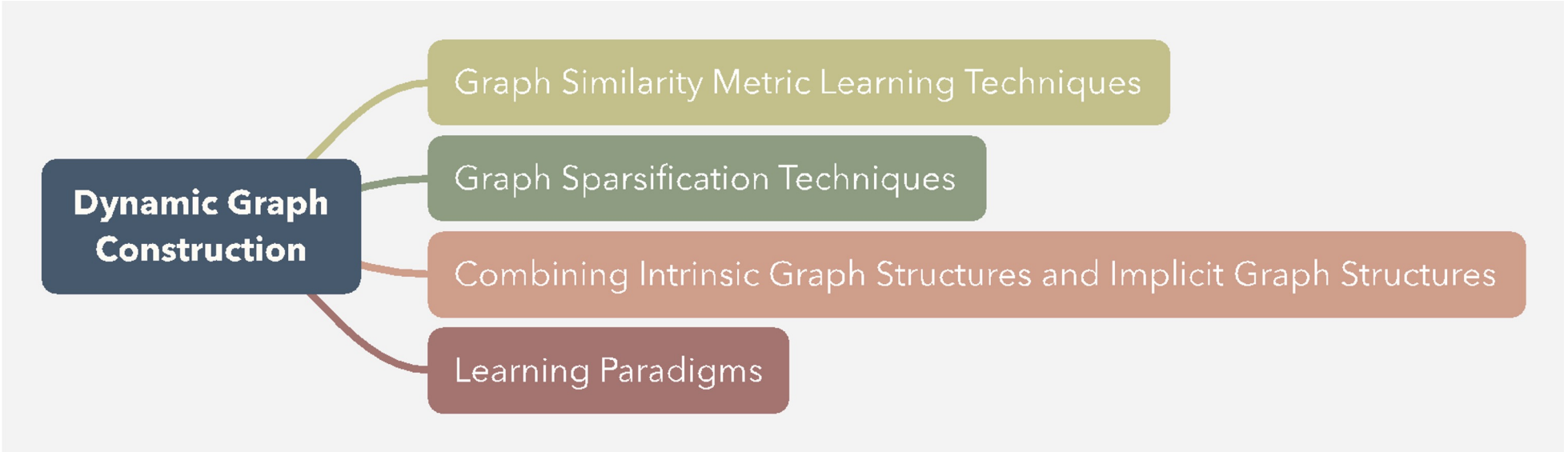


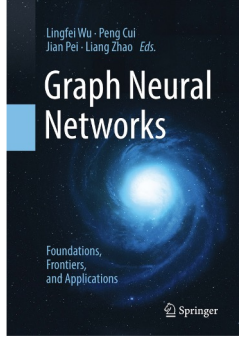
Dynamic Graph Construction: Overview

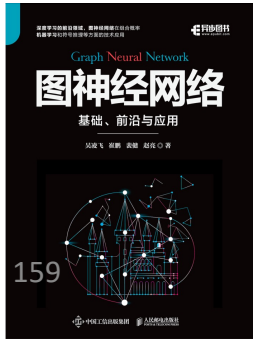


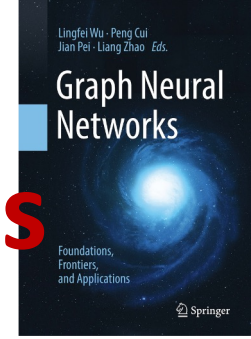


Dynamic Graph Construction Outline



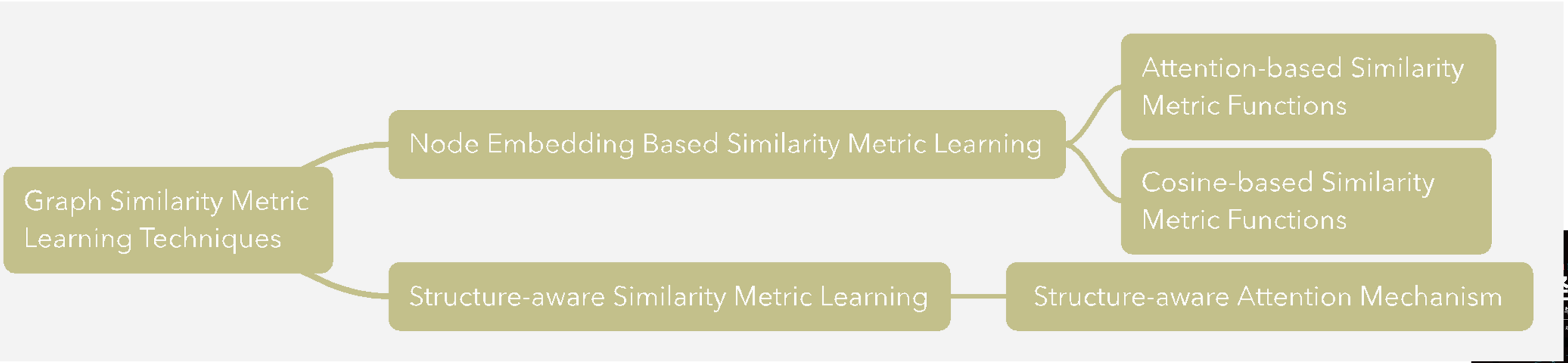




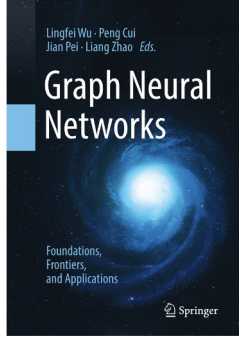


Graph Similarity Metric Learning Techniques

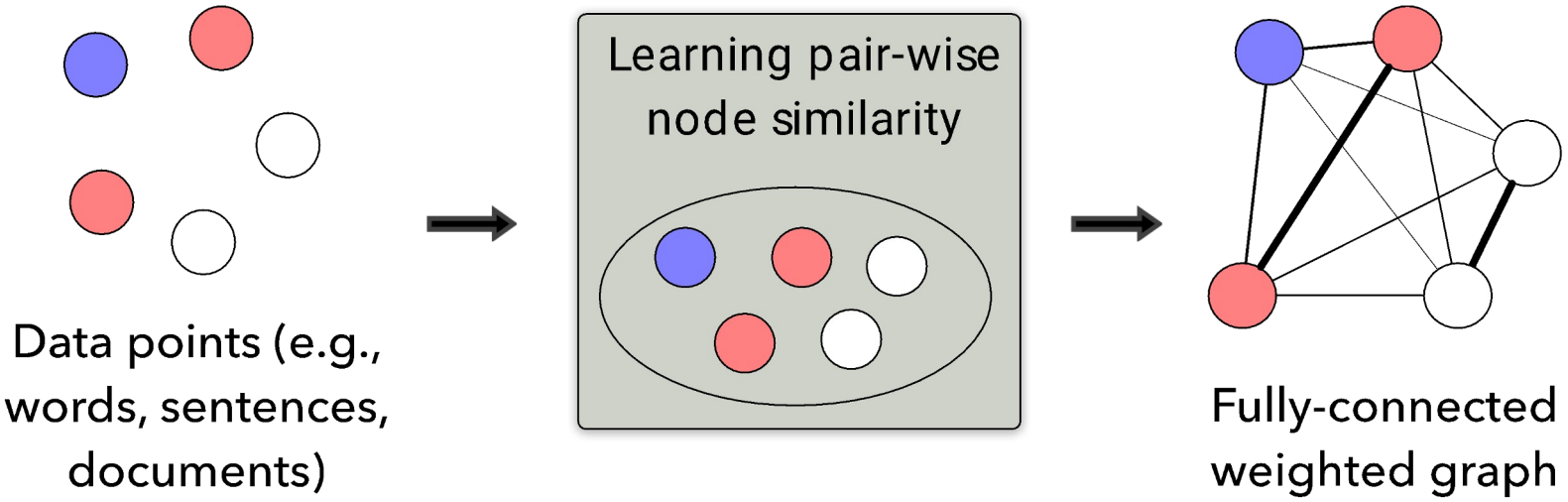
- Graph structure learning as similarity metric learning (in the node embedding space)
- Enabling inductive learning
- Various metric functions

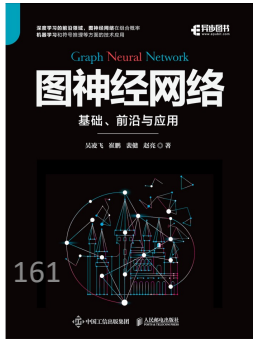


Node Embedding Based Similarity Metric Learning

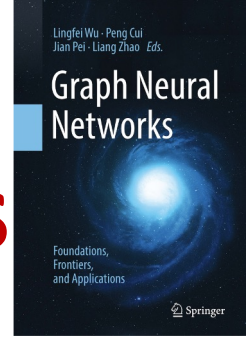


- Learning a weighted adjacency matrix by computing the pair-wise node similarity in the embedding space
- Common metrics functions
  - Attention-based similarity metric functions
  - Cosine-based similarity metric functions





Attention-based Similarity Metric Functions

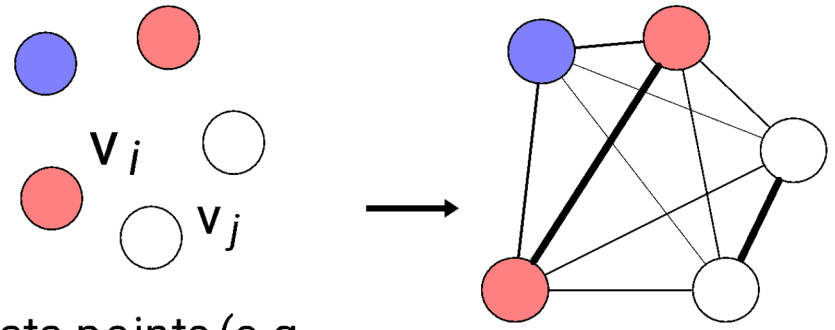


Variant 1)

$$S_{i,j} = (\mathbf{v}_i \odot \mathbf{u})^T \mathbf{v}_j$$

Node feature vector

Non-negative learnable weight vector



Data points (e.g., words, sentences, documents)

Fully-connected weighted graph

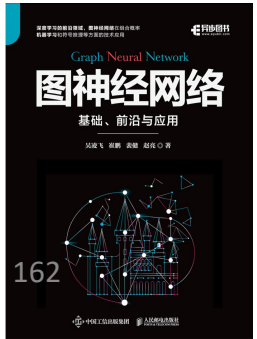
Variant 2)

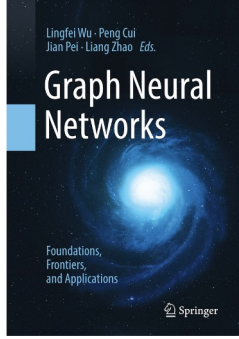
$$S_{i,j} = \mathrm{ReLU}(\mathbf{W}\mathbf{v}_i)^T \mathrm{ReLU}(\mathbf{W}\mathbf{v}_j)$$

Learnable weight matrix

Chen at al. "GraphFlow: Exploiting Conversation Flow with Graph Neural Networks for Conversational Machine Comprehension". IJCAI 2020.

Chen et al. "Reinforcement Learning Based Graph-to-Sequence Model for Natural Question Generation". ICLR 2020.





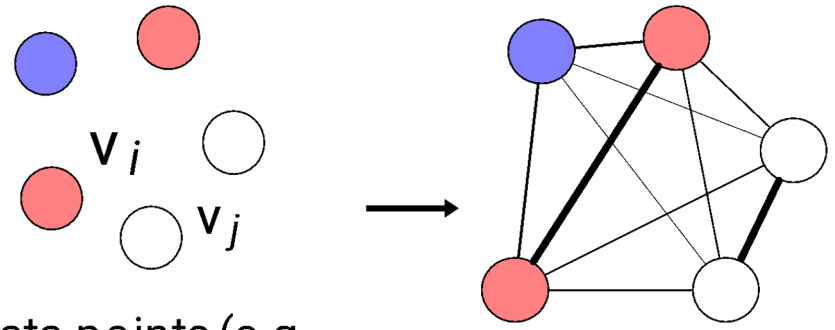
Cosine-based Similarity Metric Functions

$$S_{i,j}^p = \cos\left(\mathbf{w}_p \odot \mathbf{v}_i, \mathbf{w}_p \odot \mathbf{v}_j\right)$$

Learnable weight vector

$$S_{i,j} = \frac{1}{m}\sum_{p=1}^{m} S_{ij}^p$$

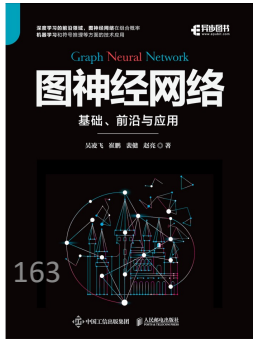
Multi-head similarity scores



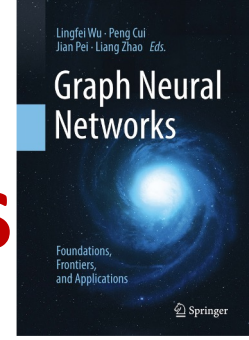
Data points (e.g., words, sentences, documents)

Fully-connected weighted graph

Chen et al. “Iterative Deep Graph Learning for Graph Neural Networks: Better and Robust Node Embeddings”. NeurIPS 2021.



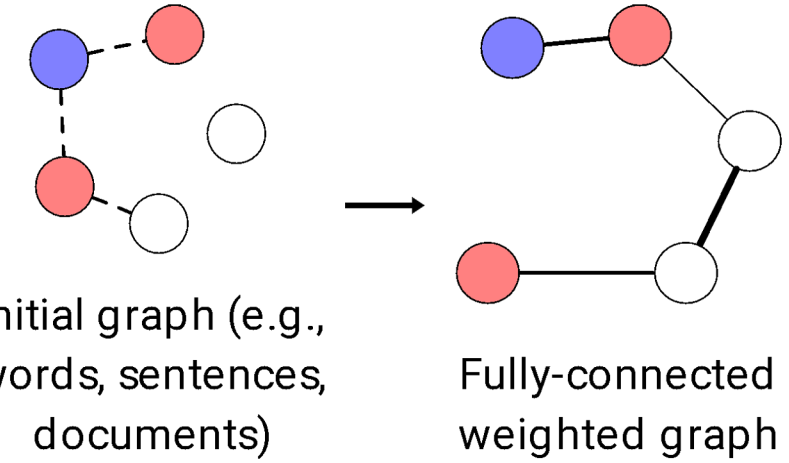
Attention-based Similarity Metric Functions



Variant 1)

$$S^l_{i,j} = \text{softmax}(\mathbf{u}^T \tanh(\mathbf{W}[\mathbf{h}^l_i, \mathbf{h}^l_j, \mathbf{v}_i, \mathbf{v}_j, \mathbf{e}_{i,j}]))$$

Edge embeddings

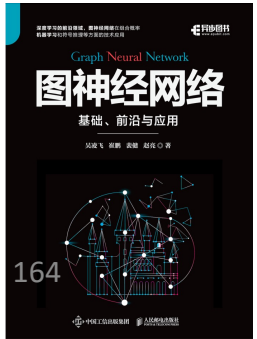


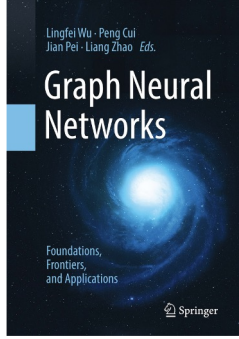
Variant 2)

$$S_{i,j} = \frac{\text{ReLU}(\mathbf{W}^Q \mathbf{v}_i)^T (\text{ReLU}(\mathbf{W}^K \mathbf{v}_i) + \text{ReLU}(\mathbf{W}^R \mathbf{e}_{i,j}))}{\sqrt{d}}$$

Liu et al. "Contextualized Non-local Neural Networks for Sequence Learning". AAAI 2019.

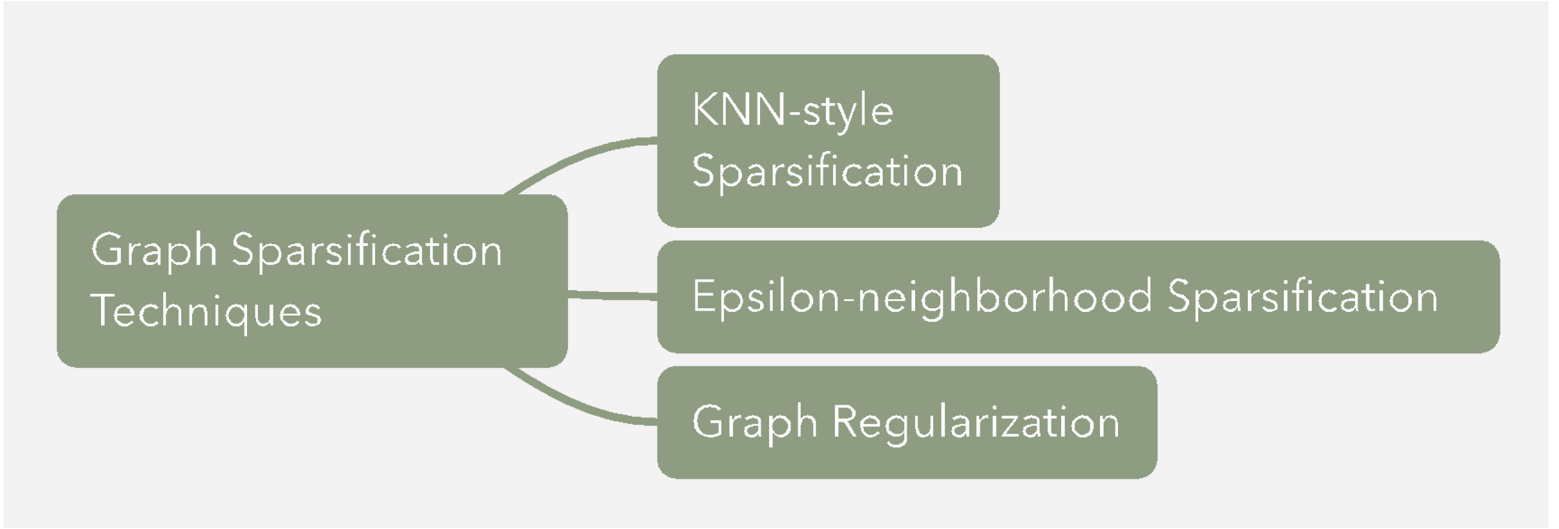
Liu et al. "Retrieval-Augmented Generation for Code Summarization via Hybrid GNN". ICLR 2021.

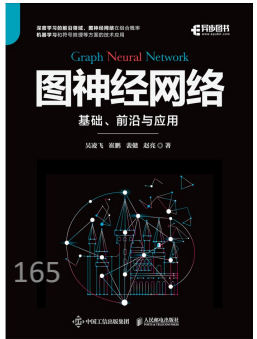


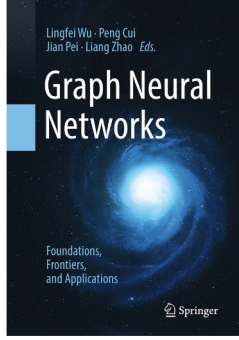


Graph Sparsification Techniques

- Similarity metric functions learn a fully-connected graph
- Fully-connected graph is computationally expensive and might introduce noise
- Enforcing sparsity to the learned graph structure
- Various techniques







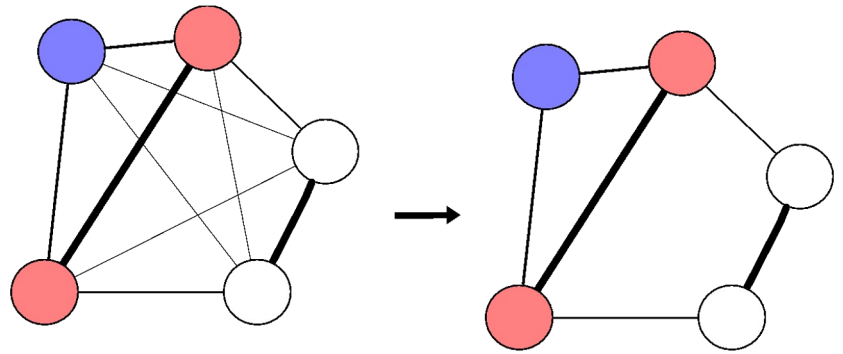
Common Graph Sparsification Options

Option 1) KNN-style Sparsification

$$\mathbf{A}_{i,:} = \text{topk}(\mathbf{S}_{i,:})$$

Option 2) epsilon-neighborhood Sparsification

$$A_{i,j} = \begin{cases} S_{i,j} & S_{i,j} > \varepsilon \\ 0 & \text{otherwise} \end{cases}$$

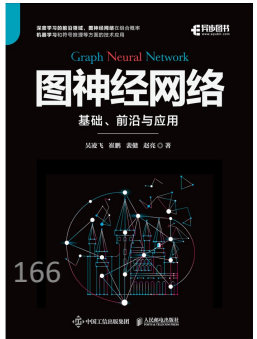


Fully-connected weighted graph

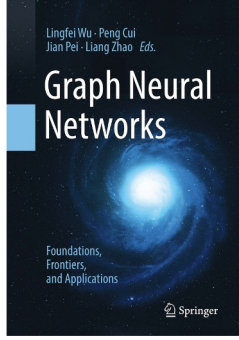
Sparsified graph

Option 3) graph Regularization

$$\frac{1}{n^2}||A||_F^2$$



Combining Intrinsic and Implicit Graph Structures



- Intrinsic graph typically still carries rich and useful information
- Learned implicit graph is potentially a "shift" (e.g., substructures) from the intrinsic graph structure

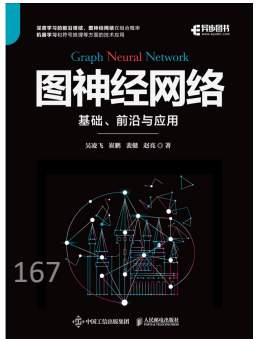
$$\widetilde{A} = \lambda L^{(0)} + (1-\lambda)\mathrm{f}(A)$$

Normalized graph Laplacian

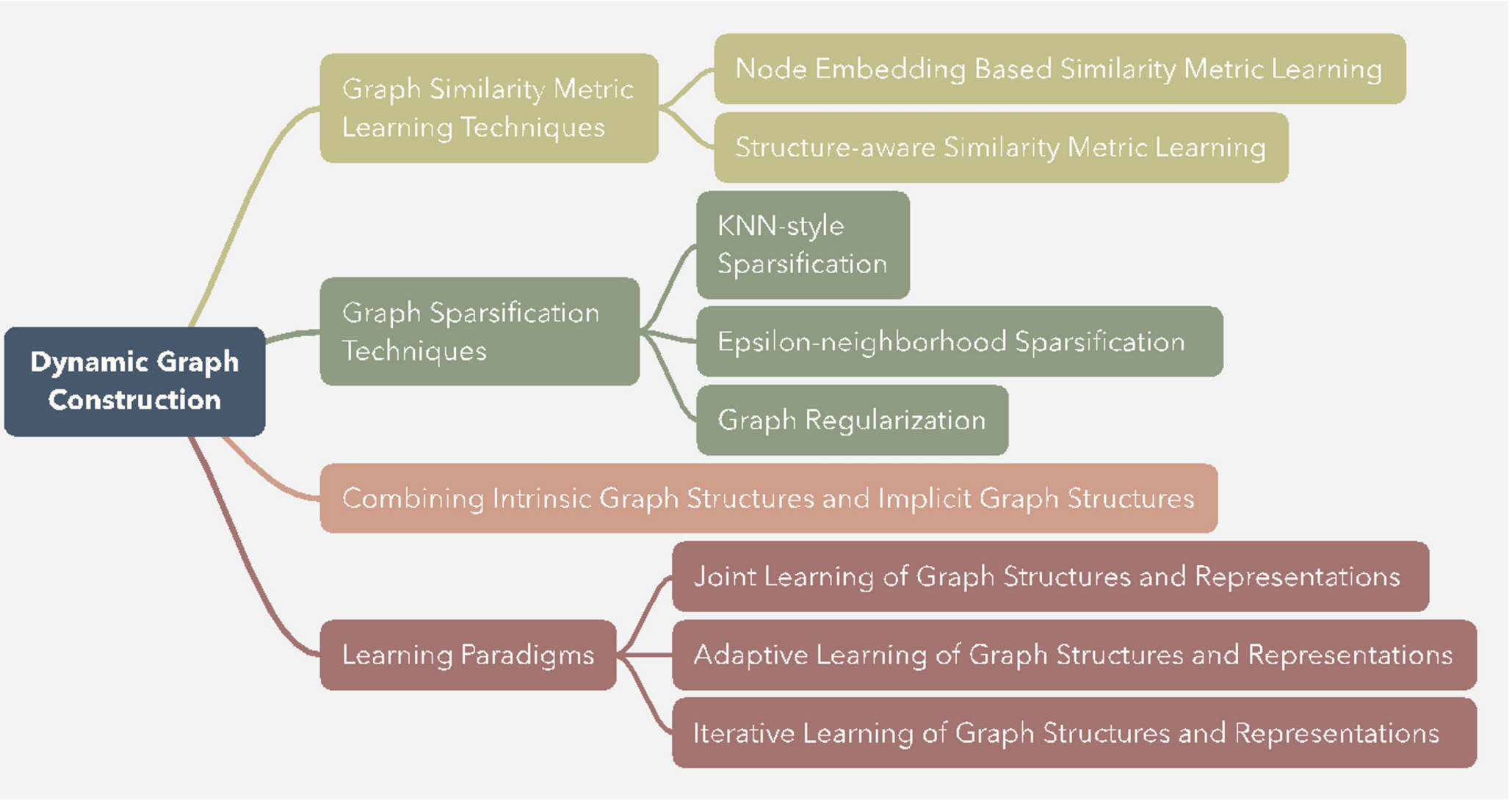
f(A) can be arbitrary operation, e.g., graph Laplacian, row-normalization

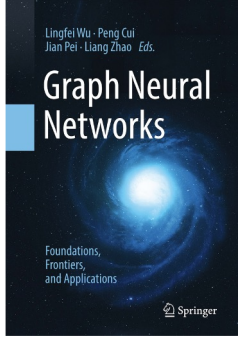
Li et al. "Adaptive Graph Convolutional Neural Networks". AAAI 2018.

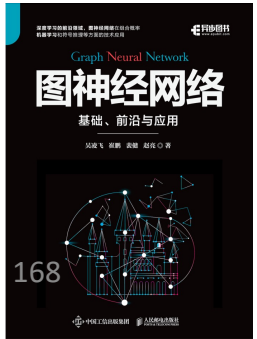
Chen et al. "Iterative Deep Graph Learning for Graph Neural Networks: Better and Robust Node Embeddings". NeurIPS 2021.



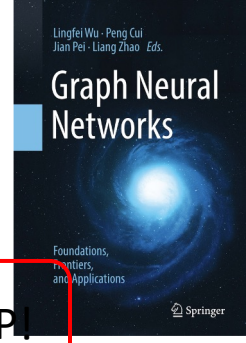
Dynamic Graph Construction Summary





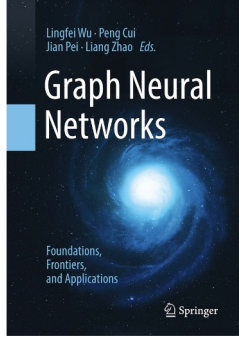


Static vs. Dynamic Graph Construction

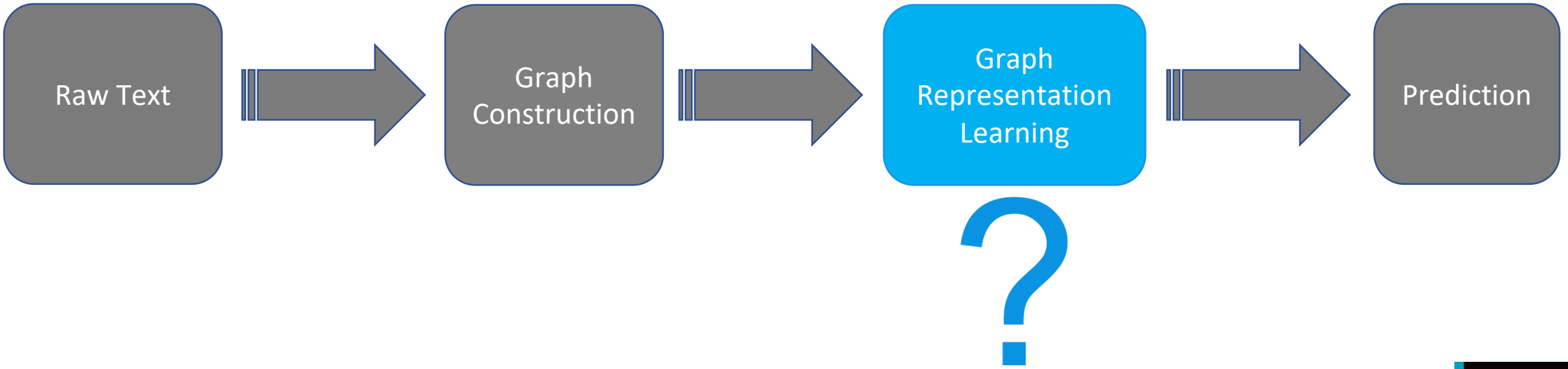


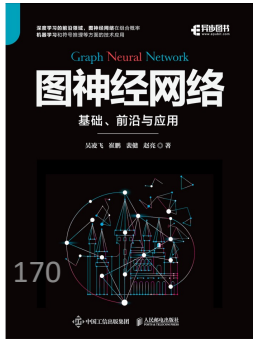
New topic in DLG4NLP!

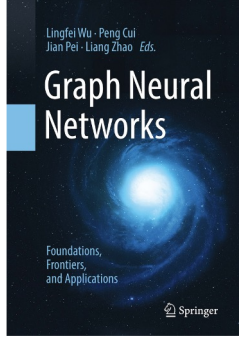
Static graph construction	Dynamic graph construction
Pros	Pros
prior knowledge	no domain expertise
	joint graph structure & representation learning
Cons	Cons
extensive domain expertise	scalability
• error-prone (e.g., noisy, incomplete) • sub-optimal	explainability
• disjoint graph structure & representation learning • error accumulation	



GNNs for Natural Language Processing

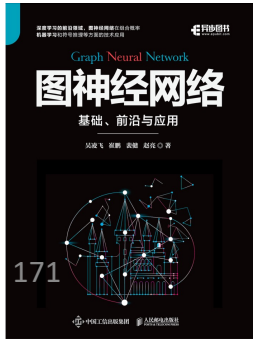




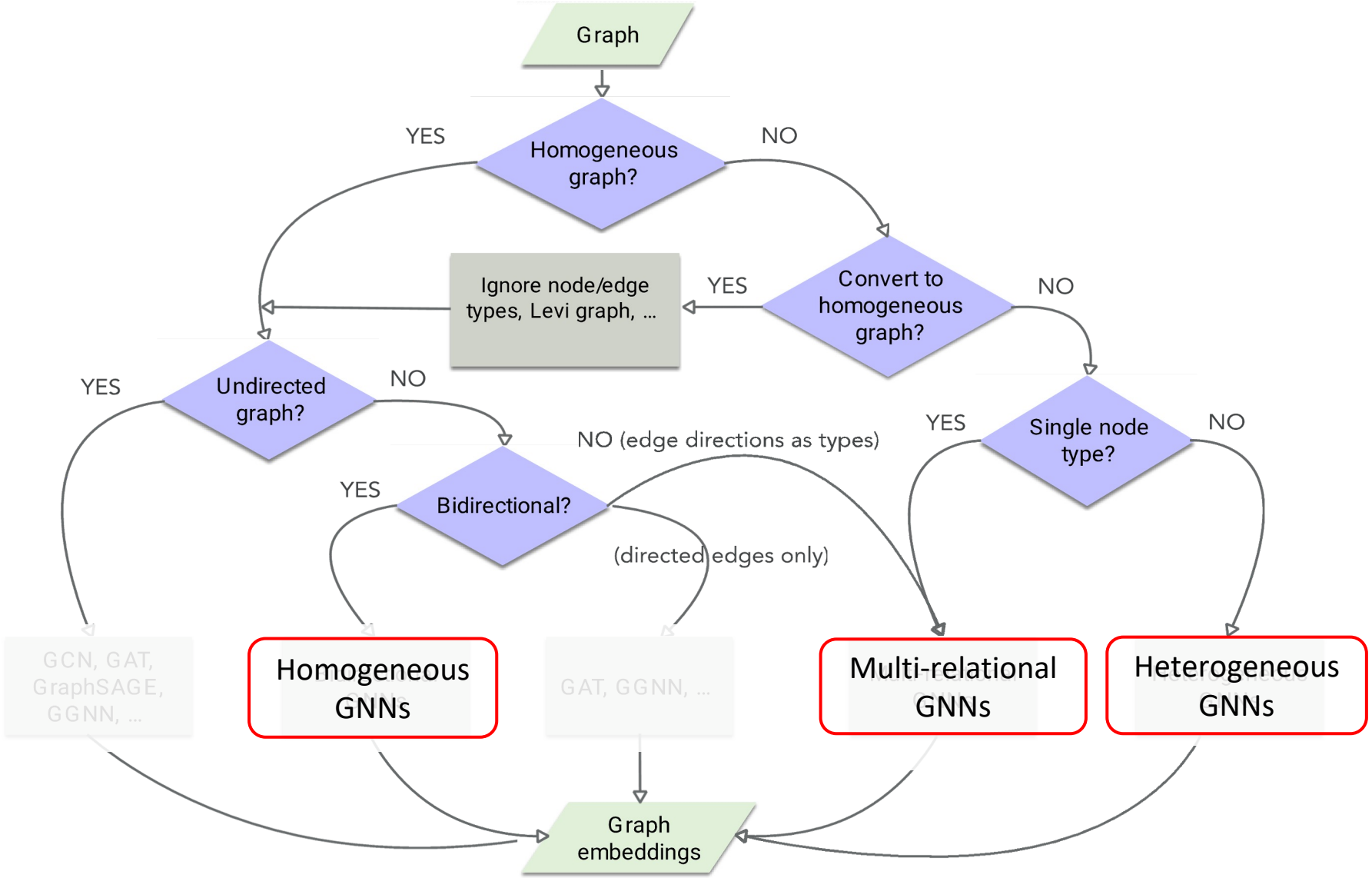


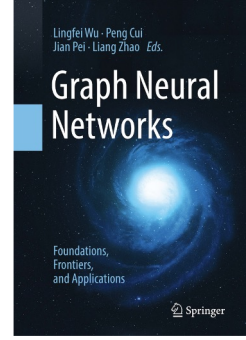
Homogeneous vs Multi-relational vs Heterogeneous Graphs

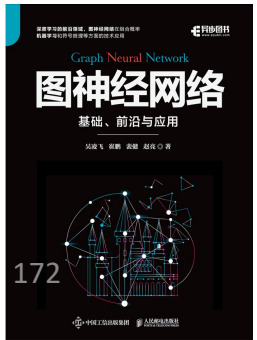
Graph types	Homogeneous	Multi-relational	Heterogeneous
# of node types	1	1	> 1
# of edge types	1	> 1	>= 1



Which GNNs to Use Given a Graph?

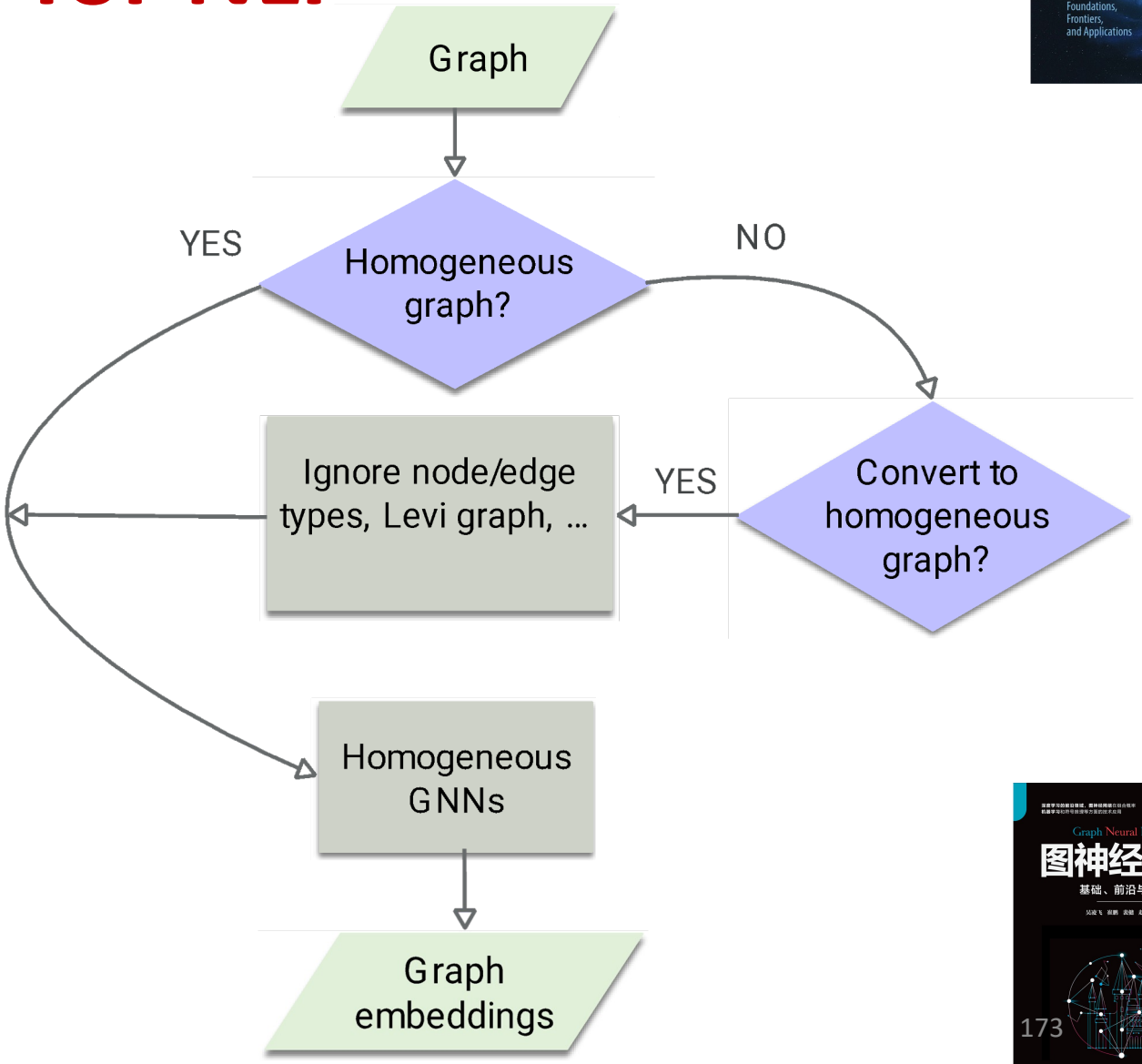


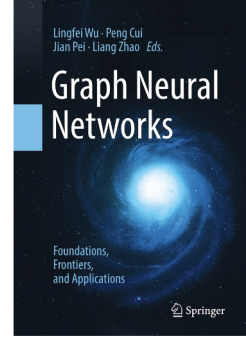


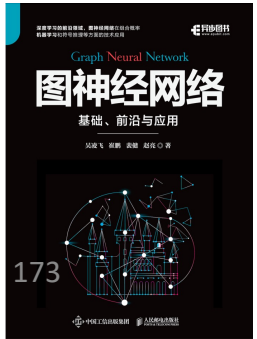


Homogeneous GNNs for NLP

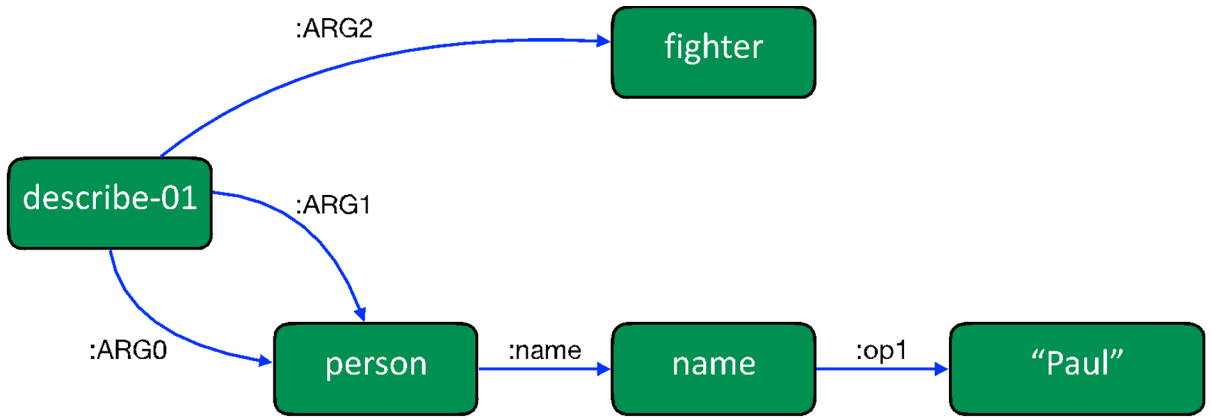
- When to use homogeneous GNNs?
- Homogeneous GNNs
  - GCN
  - GAT
  - GraphSAGE
  - GGNN
  - …



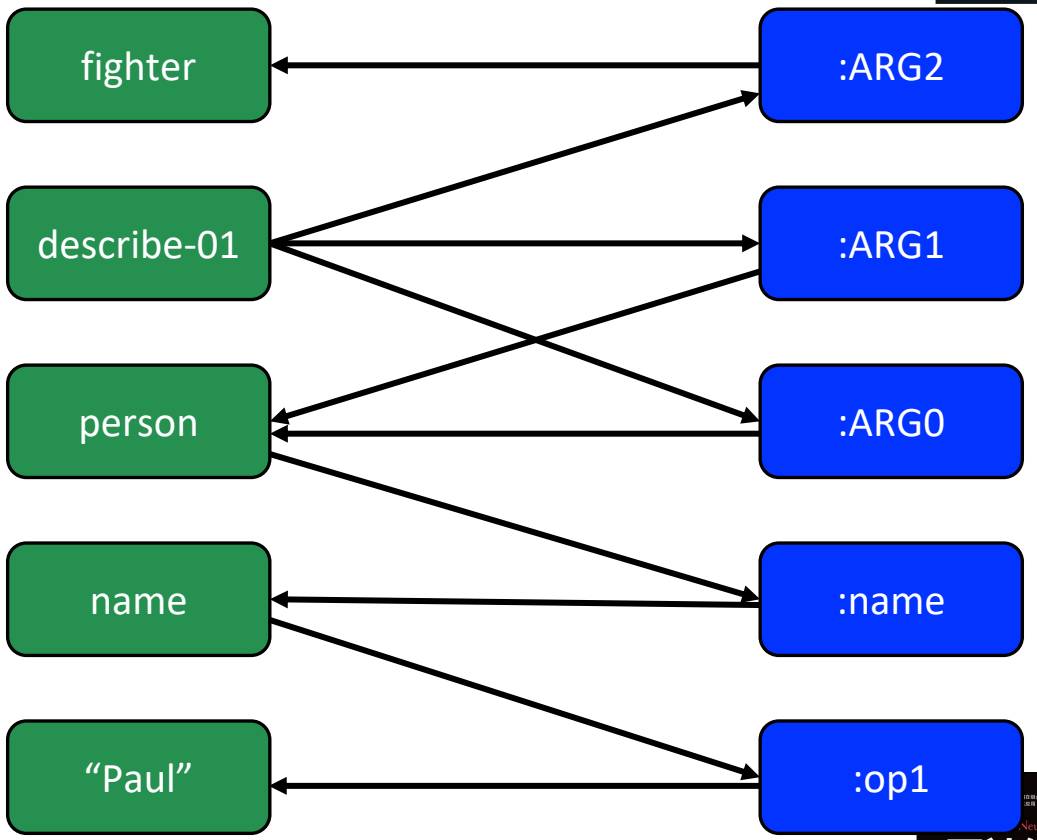




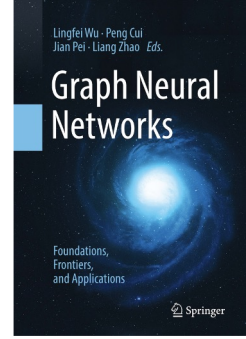
Non-homogeneous to Homogeneous Conversion via Levi Graph

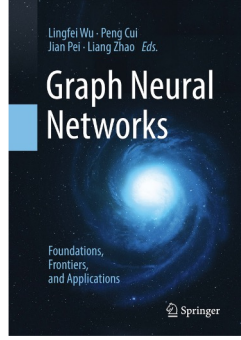


Levi graph conversion



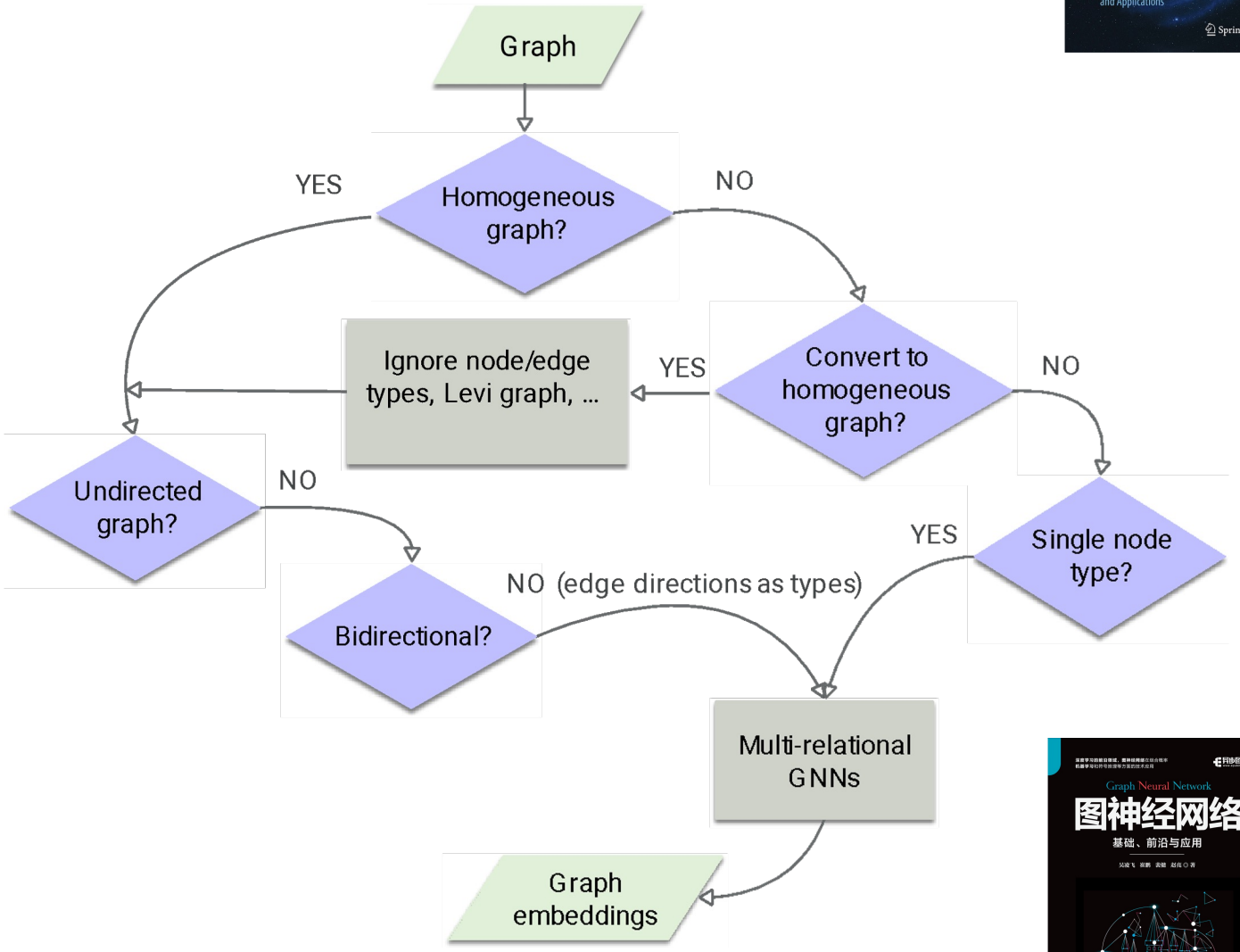
Levi graph: edges as new nodes





Multi-relational GNNs for NLP

- When to use multi-relational GNNs?
- Multi-relational GNNs
  a) Including relation-specific transformation parameters in GNN
  b) Including edge embeddings in GNN
  c) Multi-relational Graph Transformers



R-GNN: Overview

$$\mathbf{h}_i^k = \sigma(\mathbf{h}_i^{k-1}, \sum_{v_j \in \mathcal{N}(v_i)} AGG(\mathbf{h}_j^{k-1}, \theta^k))$$

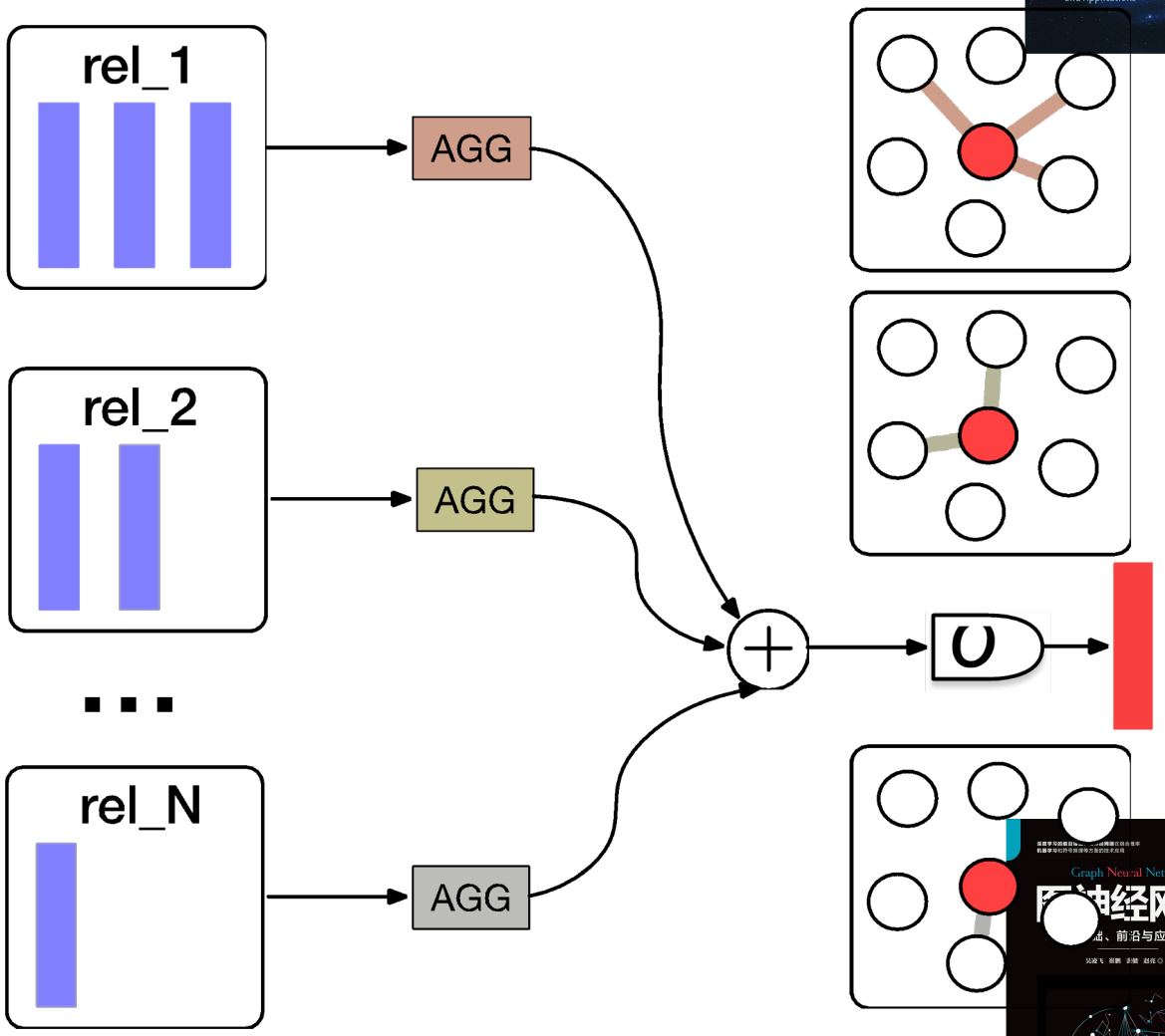
GNN

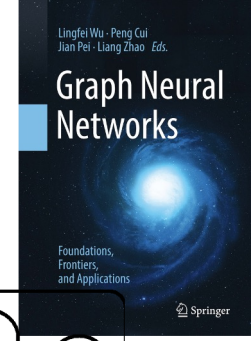
R-GNN

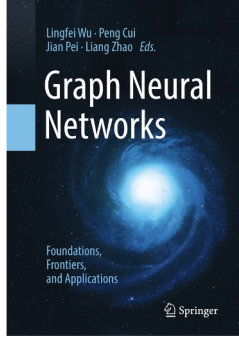
1) relation-specific transformation, e.g., node feature transformation, attention weight …

$$\mathbf{h}_i^k = \sigma(\mathbf{h}_i^{k-1}, \sum_{r \in \mathcal{E}} \sum_{v_j \in \mathcal{N}_r(v_i)} AGG(\mathbf{h}_j^{k-1}, \theta_r^k))$$

2) aggregation per relation-specific subgraph







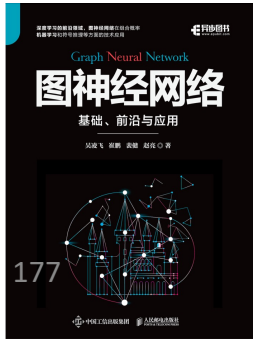
R-GNN Variant: R-GCN

- Relation-specific node feature transformation during neighborhood aggregation

$$\mathbf{h}_i^k = \sigma\Big(\sum_{r\in\mathcal{E}}\sum_{v_j\in\mathcal{N}_r(v_i)}\frac{1}{c_{i,r}}\mathbf{W}_r^k\mathbf{h}_j^{k-1} + \mathbf{W}_0^k\mathbf{h}_i^{k-1}\Big), \quad c_{i,r} = |\mathcal{N}_r(v_i)$$

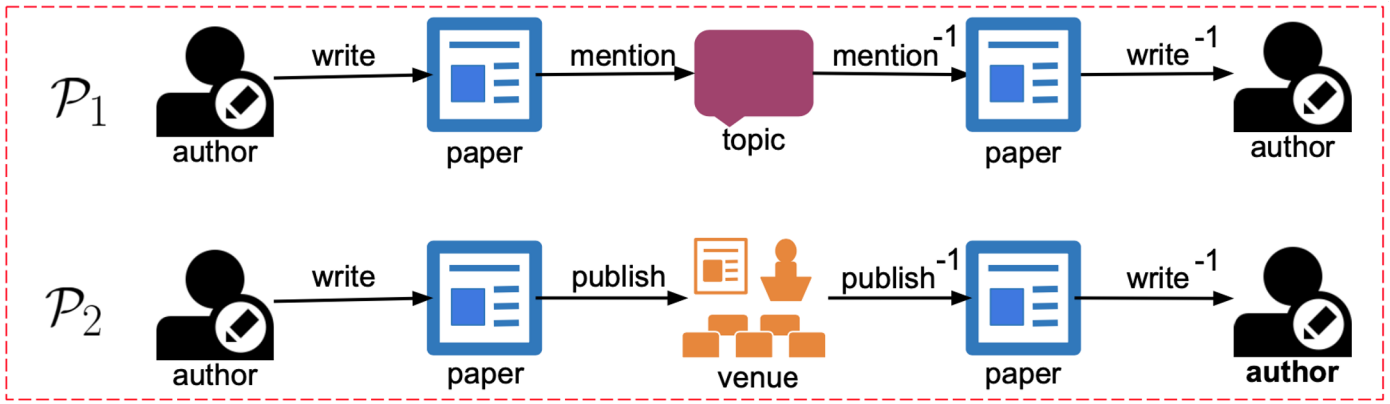
Relation-specific d x d learnable weight matrix

Schlichtkrull et al. "Modeling Relational Data with Graph Convolutional Networks". 2017.

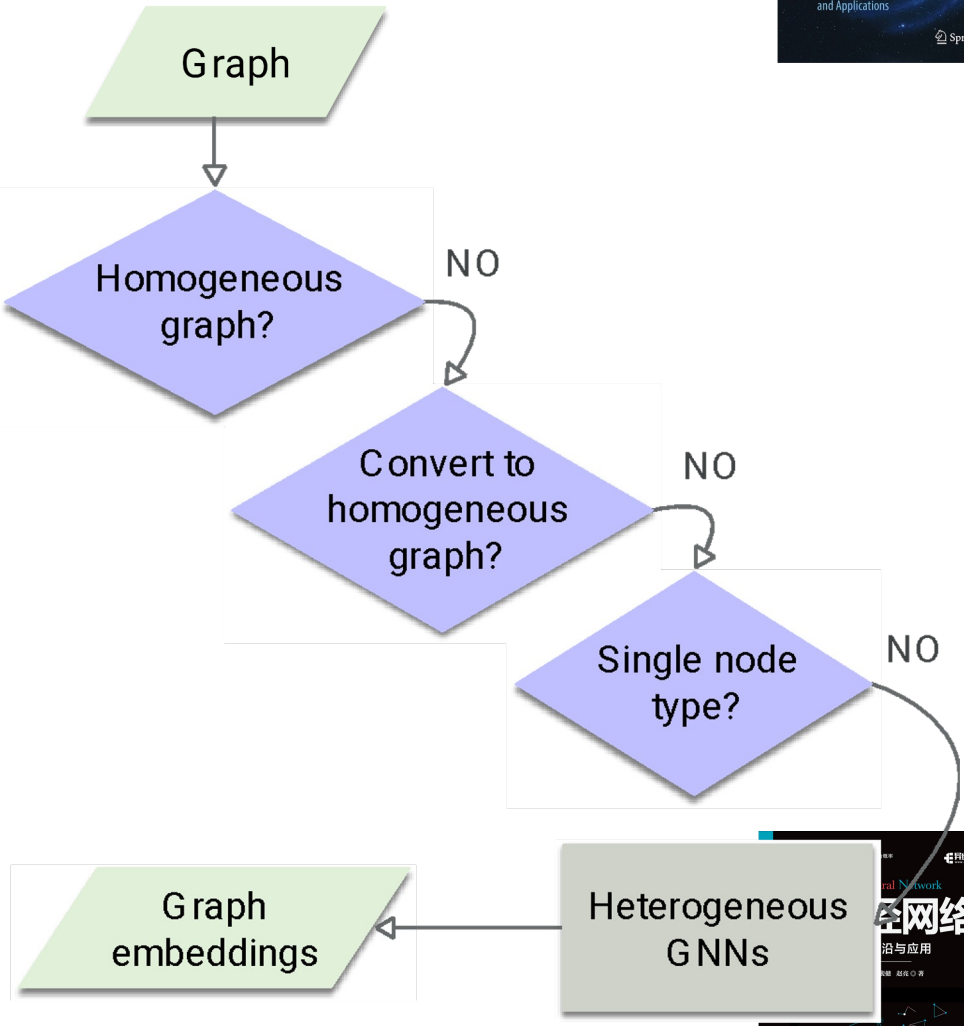


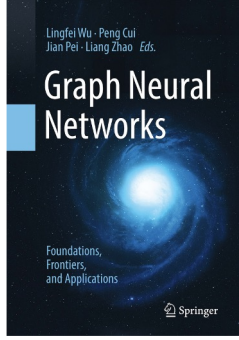
Heterogeneous GNNs

- When to use Heterogeneous GNNs?
- Heterogeneous GNNs
  a) Meta-path based Heterogeneous GNNs



Meta paths among author nodes





Meta-path based Heterogeneous GNN example: HAN

Step 1) type-specific node feature transformation

$$\mathbf{h}_i = \mathbf{W}_{\tau(v_i)}\mathbf{v}_i$$

Node-type specific learnable weight matrix

Step 2) node-level aggregation along each meta path

$$\mathbf{z}_{i,\Phi_k} = \sigma\Big(\sum_{v_j \in \mathcal{N}_{\Phi_k}(v_i)} \alpha_{i,j}^{\Phi_k}\mathbf{h}_j\Big)$$

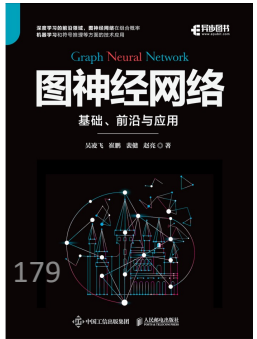
Aggregate over neighboring nodes in k-length meta path

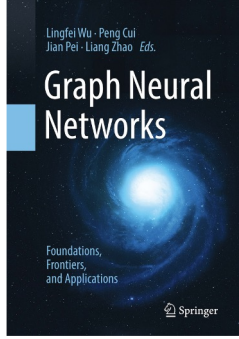
Step 3) meta-path level aggregation

$$\mathbf{z}_i = \sum_{k=1}^{p} \beta_{\Phi_k}\mathbf{z}_{i,\Phi_k}$$

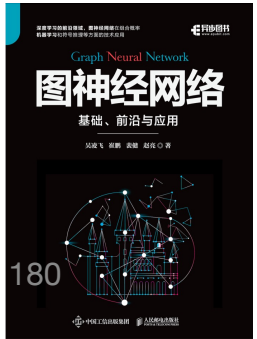
Attention weights over meta paths

Wang et al. "Heterogeneous Graph Neural Networks for Extractive Document Summarization". ACL 2020.



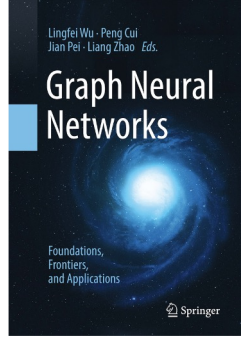


Graph4NLP: A Library for Deep Learning on Graphs for NLP



Graph4NLP

Graph4NLP: A Brief History and Future



Year June/2021

Graph4NLP v0.4.1 Released, first library for promoting easy use of GNN for NLP

Year September/2021

Our **DLG4NLP website** launched: survey, library, tutorial, workshop and many more.

Year September/2021

Graph4NLP v0.5.1 released, Lint the codes, support testing with user data, and fix many reported bugs.

Year 01/20/2022

Graph4NLP v0.5.5 released, Support model.predict API by introducing wrapper functions, separate graph Topology and graph Embedding and many more…

Year 07/31/2022 (planning)

- Graph4NLP v0.6.1 will be released, new configuratio system, relational GNN su

Year 2022

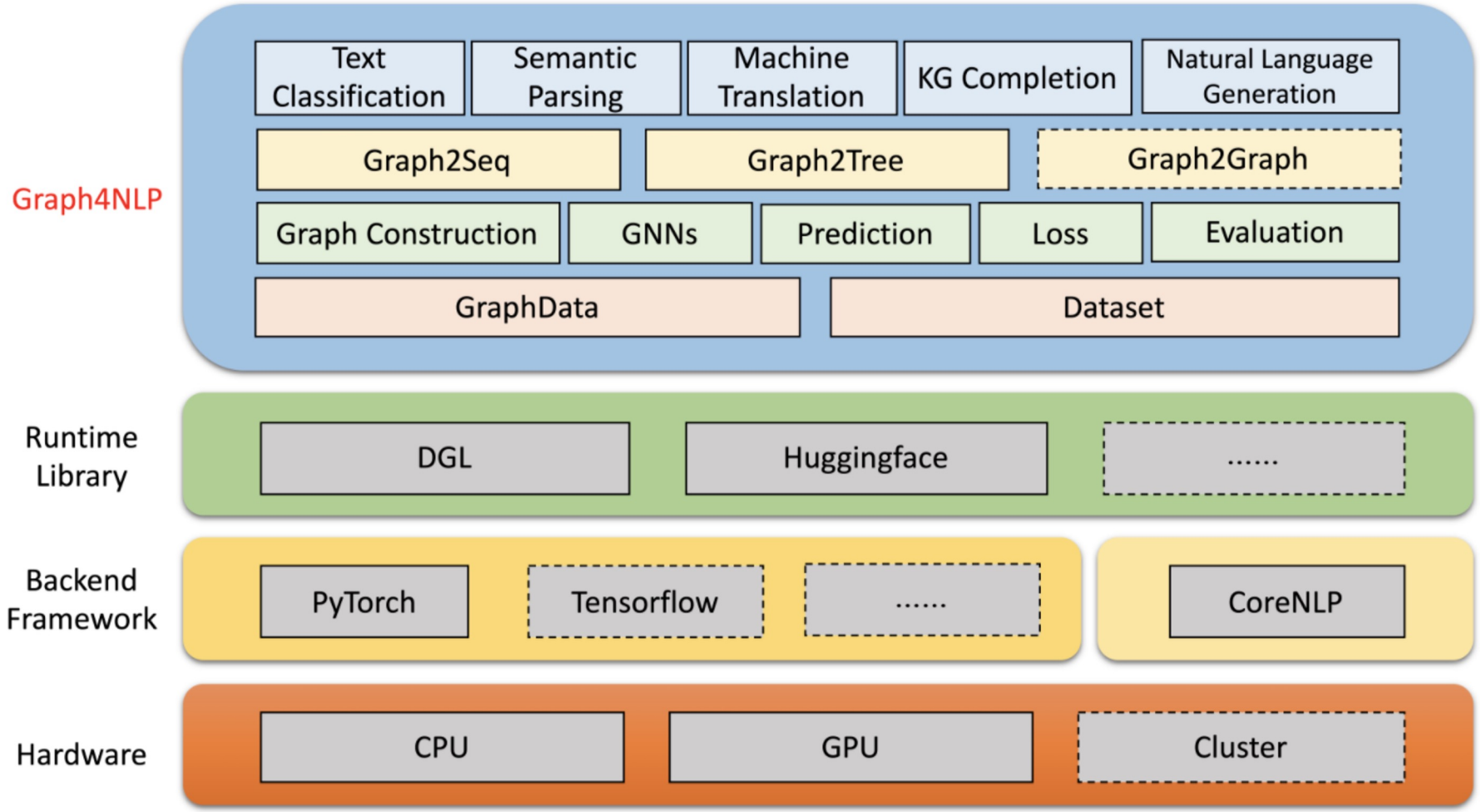
- Graph4NLP v0.6.1 will be released,

- More tutorials, and workshops

- Release a New Graph4NLP book by Cambridge Press

Graph4NLP

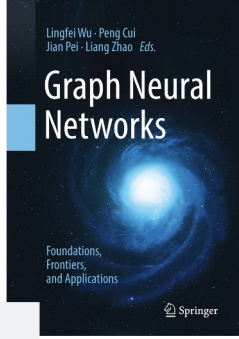
Overall Architecture of Graph4NLP Library



DGL: https://github.com/dmlc/dgl, DIG: https://github.com/divelab/DIG, Huggingface: https://github.com/huggingface/transf

Graph4NLP

Key Features and Future Releases



Easy-to-use and Flexible

Provides both full implementations of state-of-the-art models and also flexible interfaces to build customized models with whole-pipeline support

Rich Set of Learning Resources

Provide a variety of learning materials including code demos, code documentations, research tutorials and videos, and paper survey

High Running Efficiency and Extensibility

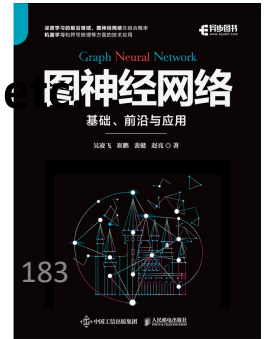
Build upon highly-optimized runtime libraries including DGL and provide highly modulization blocks

Comprehensive Code Examples

Provide a comprehensive collection of NLP applications and the corresponding code examples for quick-start

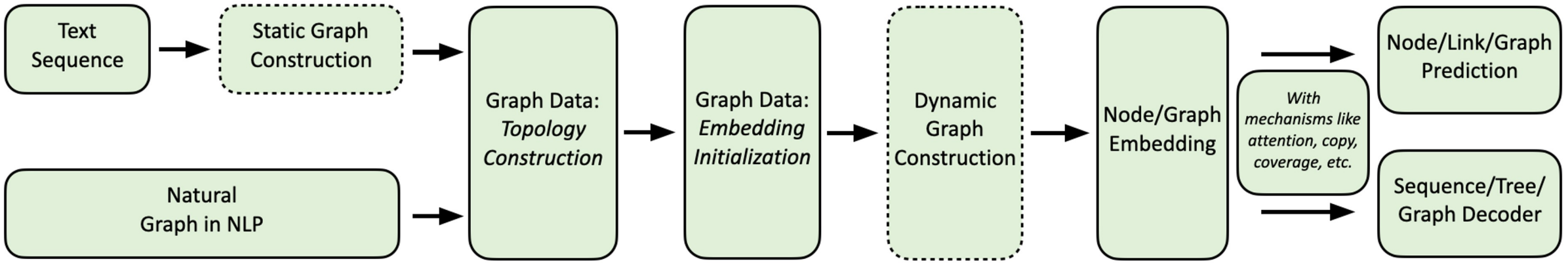
Future Releases

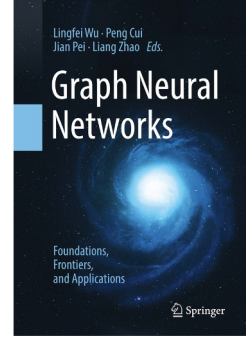
- **V0.6 new features**: new configuration system, relational GNN support, etc.
- **Future todo**: Native multi-GPU support, better support for customized graph construction, e

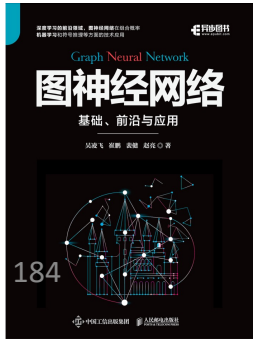


Graph4NLP

Computing Flow of Graph4NLP

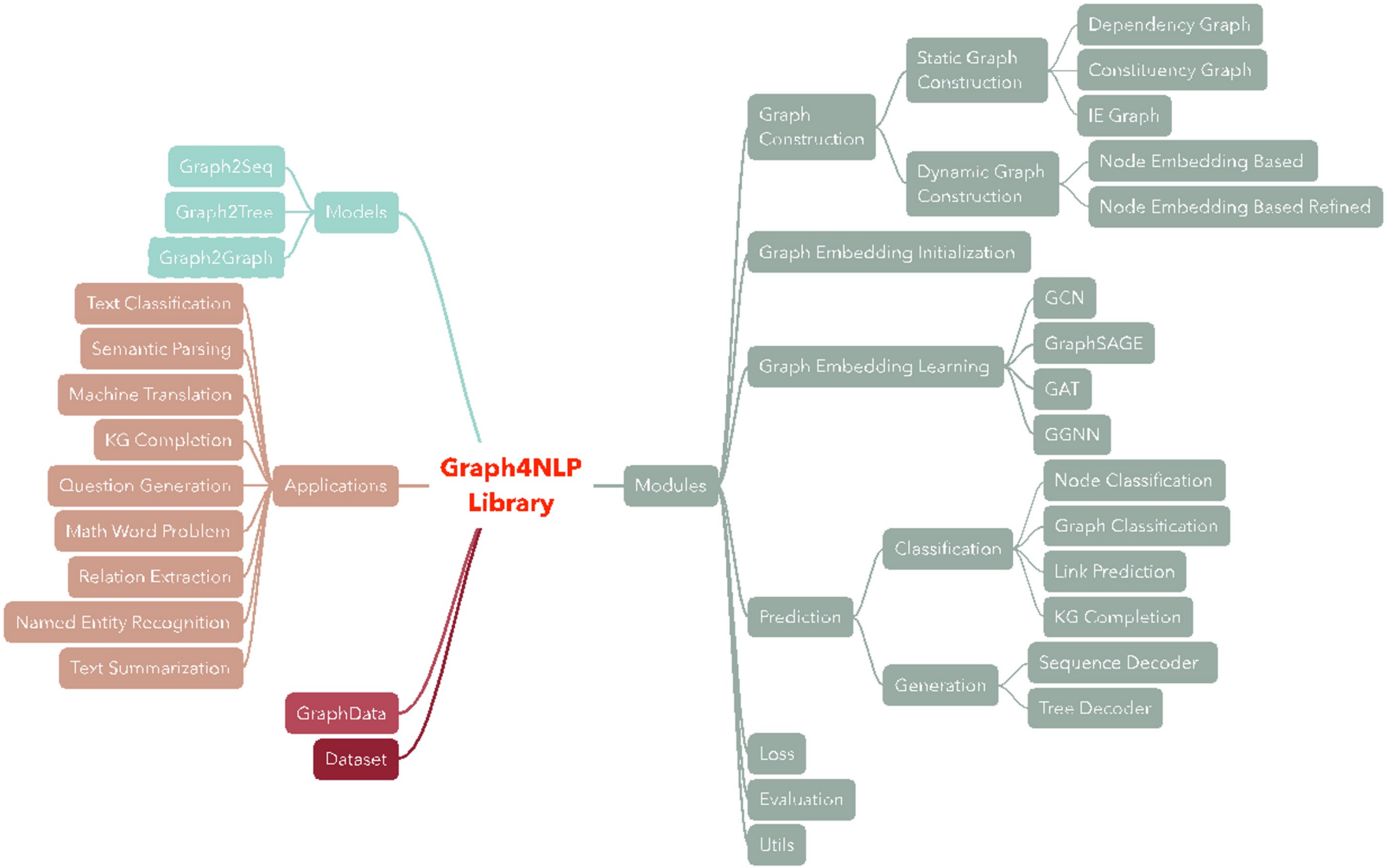


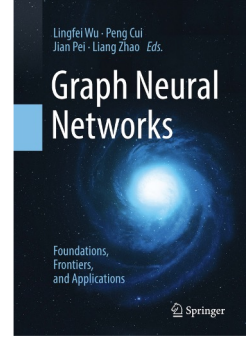


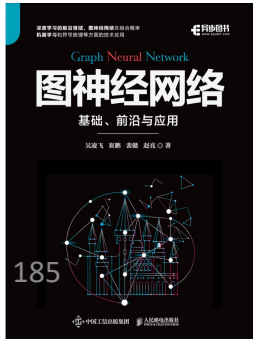


Graph4NLP

Dive Into Graph4NLP Library



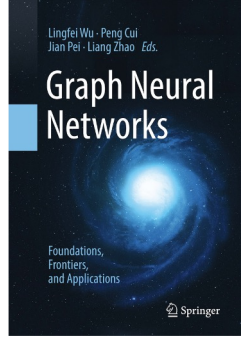


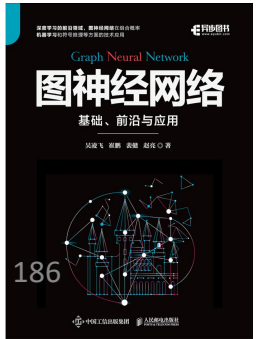


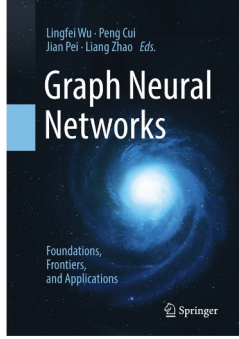
Graph4NLP

Conclusions

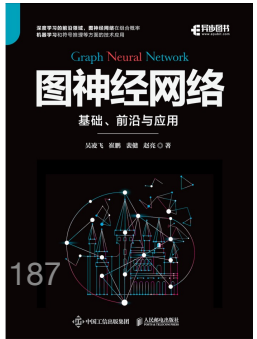
- Deep Learning on Graphs for NLP is a fast-growing area today!
- Since graph can naturally encode complex information, it could bridge a gap by combining both empirical domain knowledges and the power of deep learning.
- For a NLP task,
  - how to convert text sequence into the best graph (directed, multi-relation, heterogeneous)
  - how to determine proper graph representation learning technique?
- Our Graph4NLP library aims to make easy use of GNNs for NLP:
  - Website: https://dlg4nlp.github.io/index.html
  - Survey: https://arxiv.org/abs/2106.06090
  - Library: https://github.com/graph4ai/graph4nlp
  - Demo: https://github.com/graph4ai/graph4nlp_demo



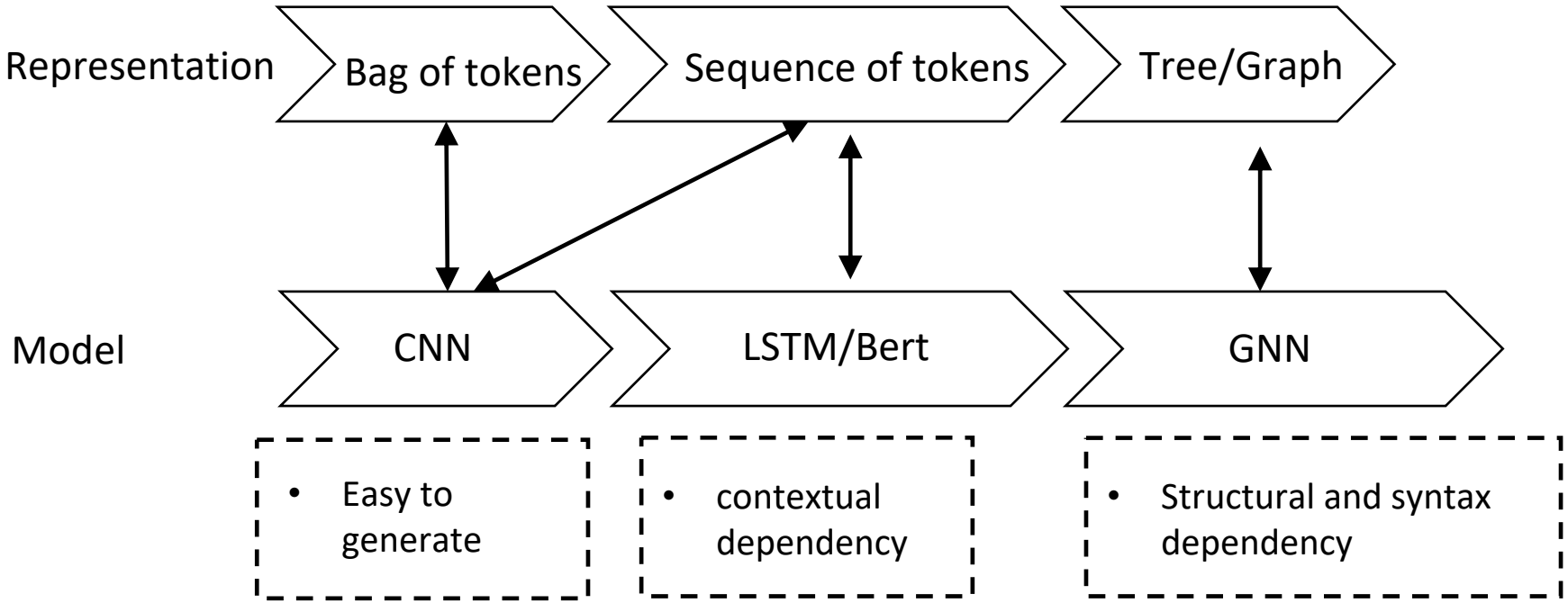


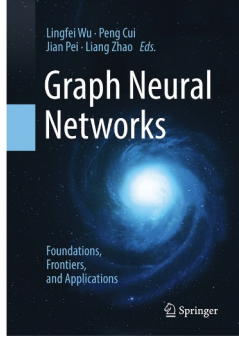


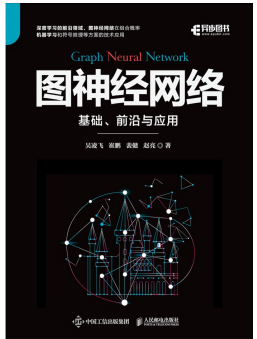
GNNs in Program Analysis (Chapter 22)

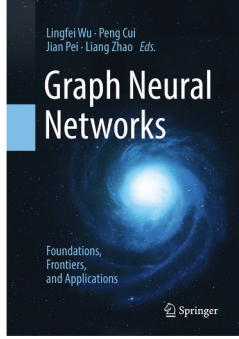


Representation of Programs in DL



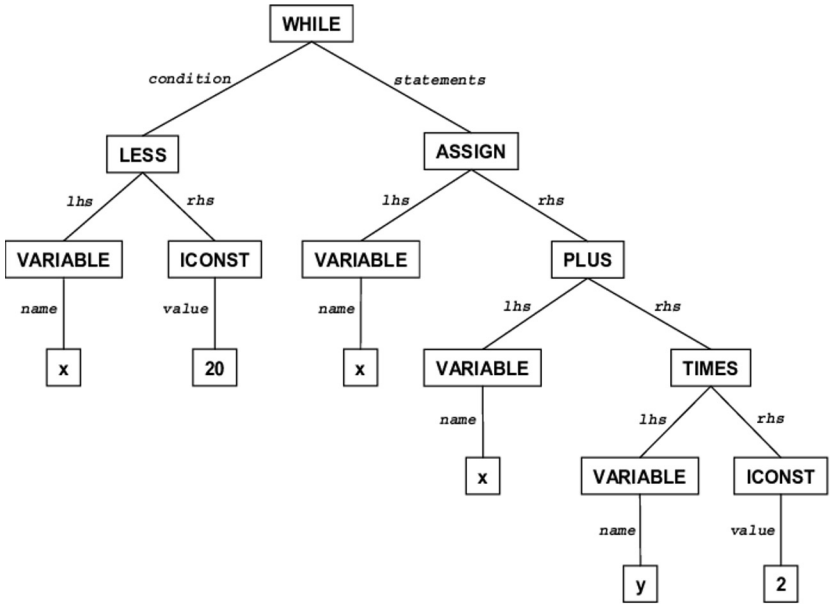




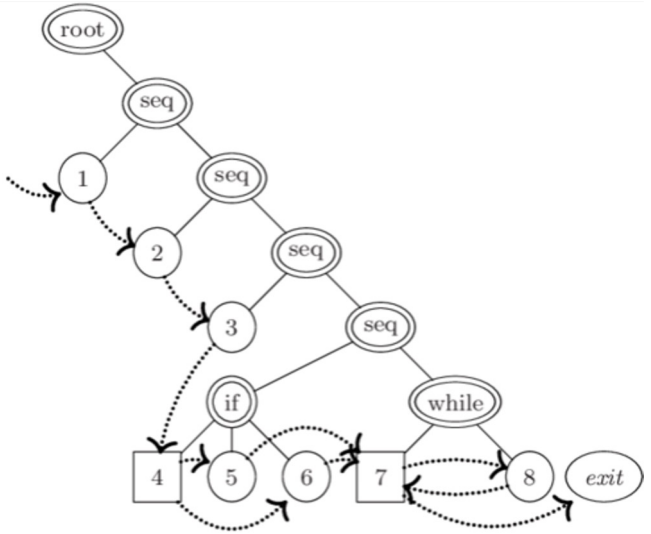


A Graph Representation of Programs

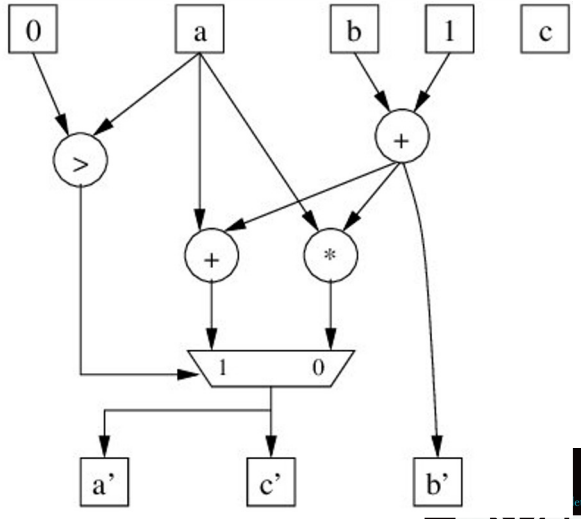
Trade-offs between expressing various program properties, the size of the graph representation, and the effort required to generate them.



Syntax Tree
(Fritzson P, et al 2009)

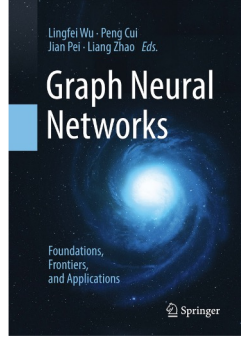


Control Flow Graph
(Va E , et al 2007)
n

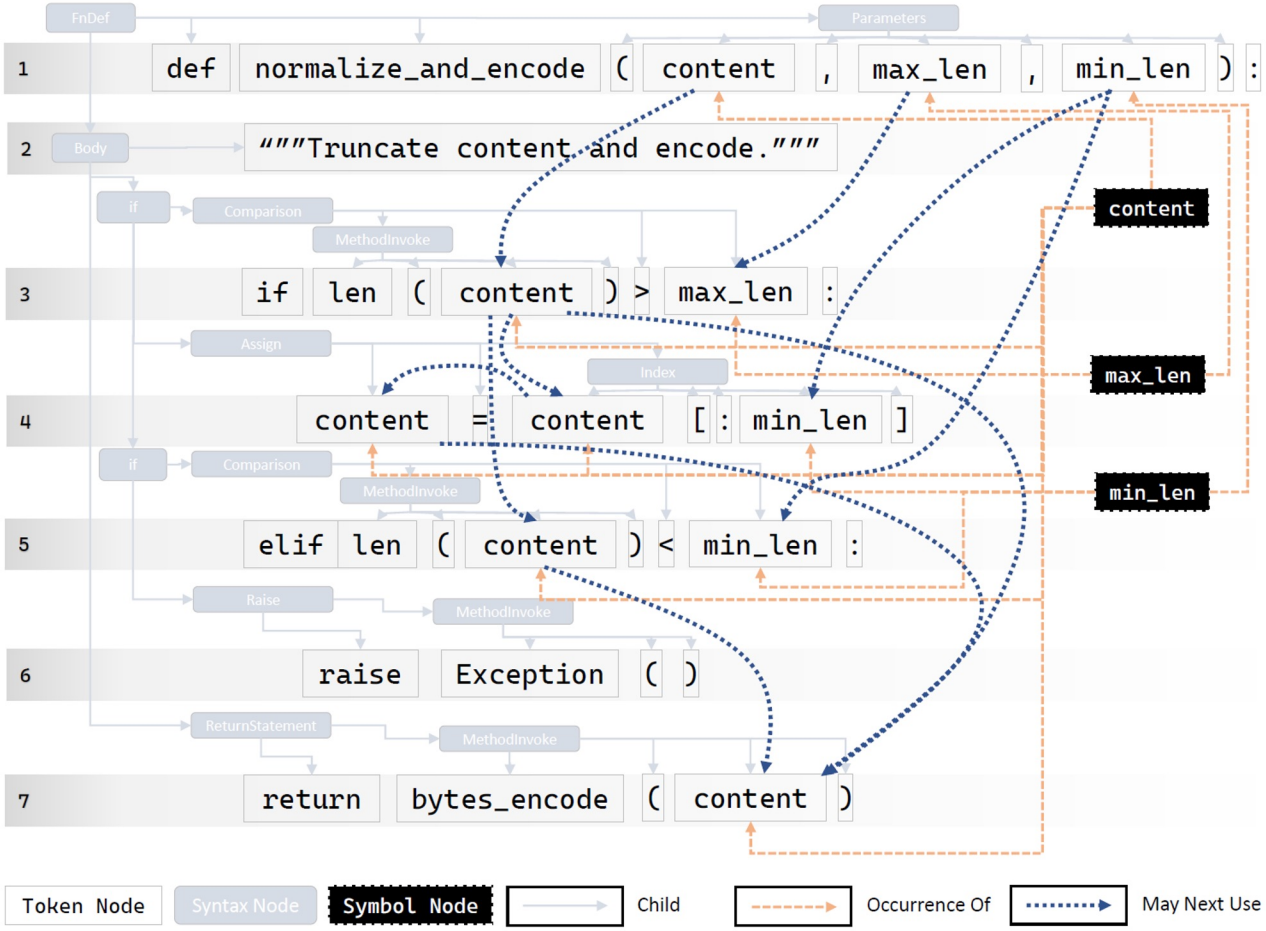


Data Flow Gra
(A. Koelbl, et al 20

A Graph Representation of Programs



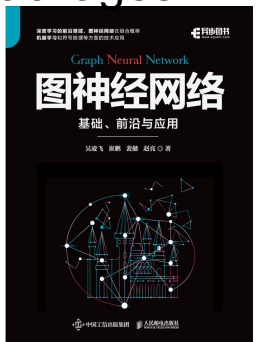
Model each source code file as a single graph (Allamanis et al, 2018b)



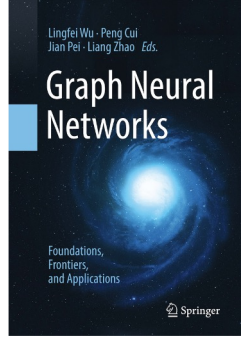
Token node: Program language can be tokenized into a sequence of tokens. (NextToken edge)

Syntax node: The sequence of tokens is parsed into a syntax tree. **(**Child edge**)**

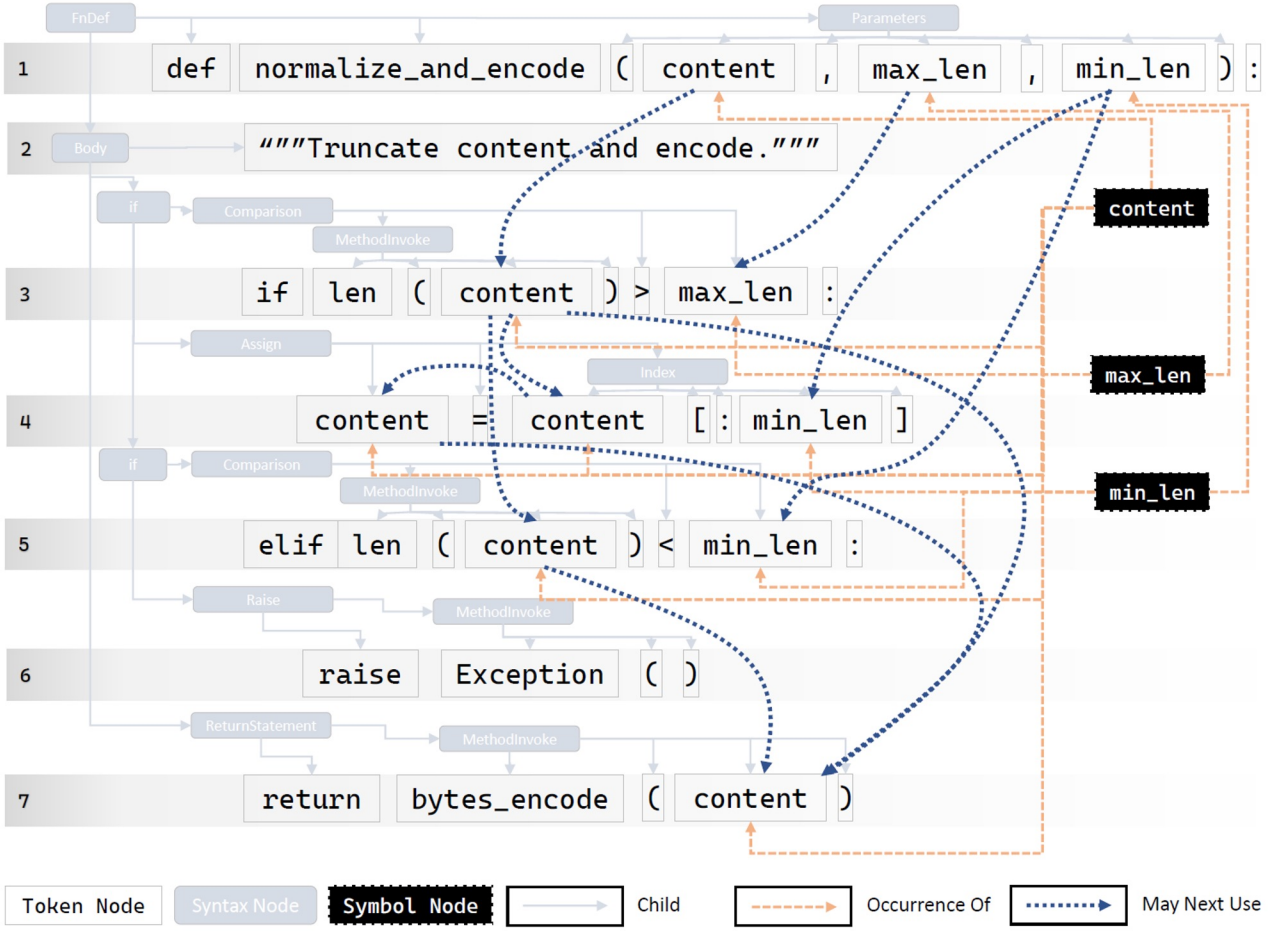
Symbol node: variables, functions, packages (Occurrence edge)



A Graph Representation of Programs



Model each source code file as a single graph (Allamanis et al, 2018b)

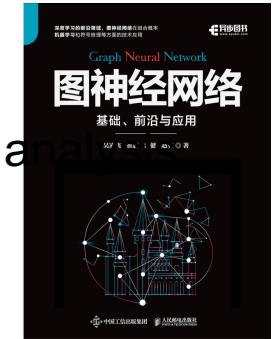


Data Flow: all the valid paths that data may flow through the program.

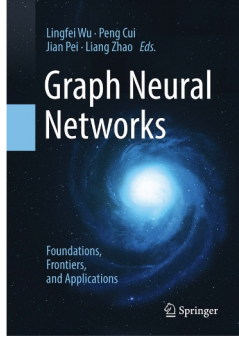
(MayNextUse)

Program Dependence Graph

- Less model capacity for learning deterministic facts.
- Inductive biases help on program analysis tasks.



Case Study 1: Detecting Variable Misuse Bugs



Variable misuse: the incorrect use of one variable instead of another already in the scope

Significance: 12% of the bugs in a large set of Java codebases are variable misuses, 6% of Java build errors in the Google engineering systems are variable misuses.

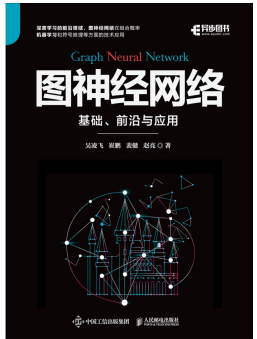
Goal: (1) localize the bug (2) suggest a repair.

$$\{\mathbf{h}_{v_i}\} = \text{GNN}\left(\mathscr{G}', \{\mathbf{n}_{v_i}\}\right)$$

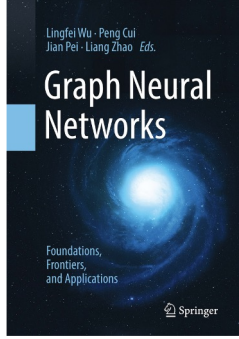
GNN for directed heterogeneous graphs.

$$p_{loc}(v_i) = \underset{v_j \in \mathscr{V}_{vu} \cup \{\emptyset\}}{\text{softmax}}\left(\mathbf{u}^\top \mathbf{h}_{v_i}\right)$$

Pointer network (Vinyalset al, 2015)



Case Study 1: Detecting Variable Misuse Bugs



Variable misuse: the incorrect use of one variable instead of another already in the scope

Significance: 12% of the bugs in a large set of Java codebases are variable misuses, 6% of Java build errors in the Google engineering systems are variable misuses.

Goal: (1) localize the bug (2) suggest a repair.

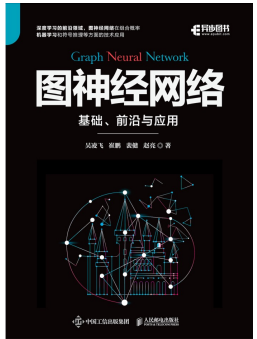
$$\{\mathbf{h}_{v_i}\} = \text{GNN}\left(\mathcal{G}', \{\mathbf{n}_{v_i}\}\right)$$

GNN for directed heterogeneous graphs.

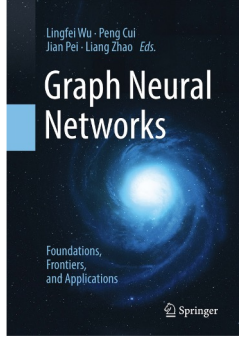
$$p_{rep}(s_i) = \underset{s_j \in \mathcal{V}_{s@v_{bug}}}{\text{softmax}}\left(\mathbf{w}^\top [\mathbf{h}_{v_{bug}}, \mathbf{h}_{s_i}]\right),$$

Embedding of correct candidates

Embedding of misuse nodes



Case Study 1: Detecting Variable Misuse Bugs



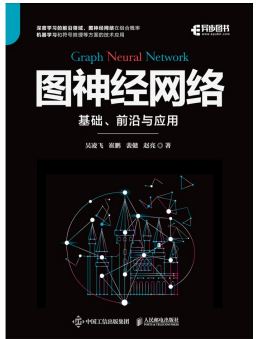
Variable misuse: the incorrect use of one variable instead of another already in the scope

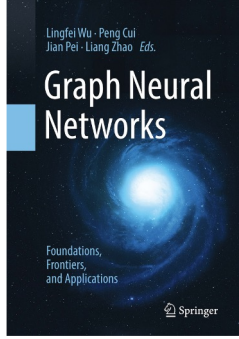
Significance: 12% of the bugs in a large set of Java codebases are variable misuses, 6% of Java build errors in the Google engineering systems are variable misuses.

Goal: (1) localize the bug (2) suggest a repair.

```
1 def describe_identity_pool(self, identity_pool_id):
2   identity_pool = self.identity_pools.get(identity_pool_id, None)
3
4   if not identity_pool:
5 -     raise ResourceNotFoundError(identity_pool)
6 +     raise ResourceNotFoundError(identity_pool_id)
7 ...
```

A diff snippet of code with a real-life variable misuse error caught by a GNN-based model.





Case Study 2: Predicting Types in Dynamically Typed Languages

Type check: Guarantee that the values of variables will only take the values of the annotated type (e.g., int, float, str)

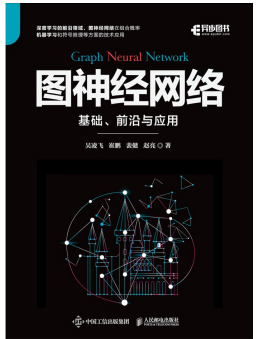
Issues: many programming languages either forgo the guarantees provided by types or require their users to explicitly provide type annotations.

Probabilistic type inference: Node classification tasks

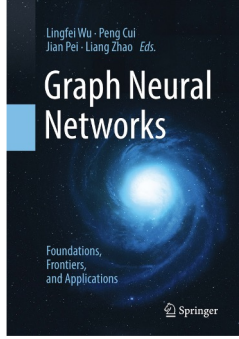
$$p(s_j : \tau) = \underset{\tau' \in Z}{\text{softmax}} \left(E_\tau{}^\top \mathbf{h}_{v_{s_j}} + b_\tau \right),$$

fixed vocabulary of type annotations

Open-source project and datasets: https://github.com/microsoft/dpu-utils



Case Study 2: Predicting Types in Dynamically Typed Languages



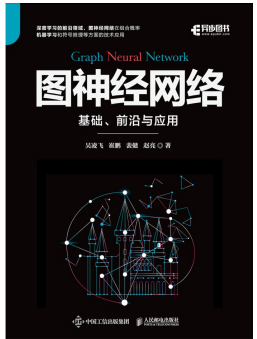
Current Limitations:

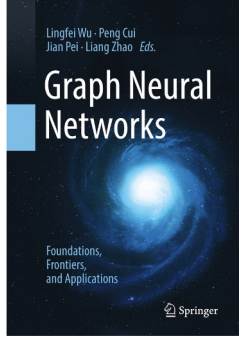
- Type annotations are highly structured and sparse.
  e.g., *Dict[Tuple[int, str], List[bool]]* appear infrequently
- Dynamic/open set recognition environment
  e.g., new user-defined types (classes) will also appear at test time.

```
def __init__(
  self,
- embedding_dim: float = 768,
- ffn_embedding_dim: float = 3072,
- num_attention_heads: float = 8,
+ embedding_dim: int = 768,
+ ffn_embedding_dim: int = 3072,
+ num_attention_heads: int = 8,
  dropout: float = 0.1,
  attention_dropout: float = 0.1,
```

Typilus (Allamanis et al, 2020).

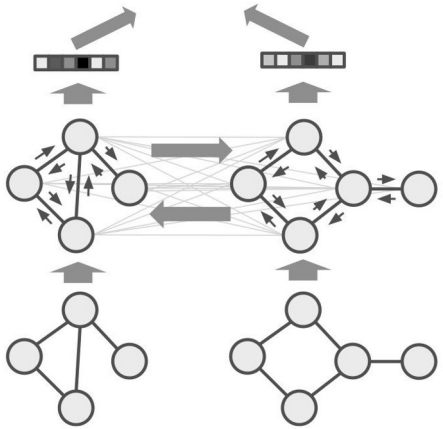
The variables cannot contain floats but instead should contain integers.



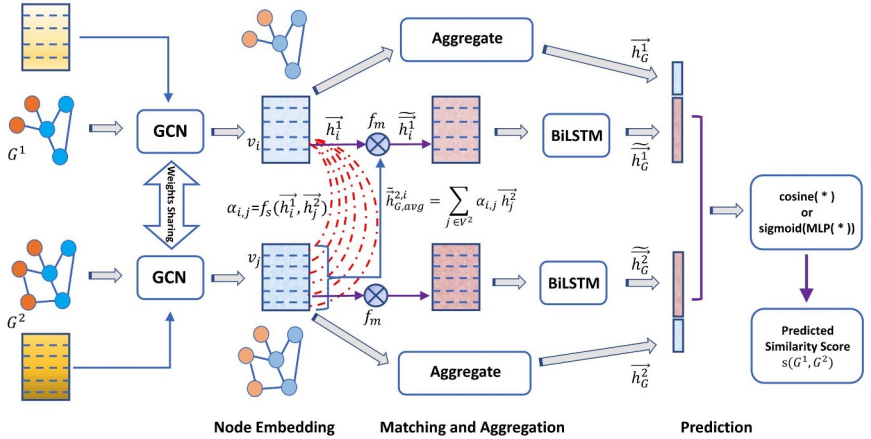


Case Study 3: code similarity analysis

Goal: Compare and predict whether two codes are solving the same problem.
Graph Matching problem: finding a similarity between graphs



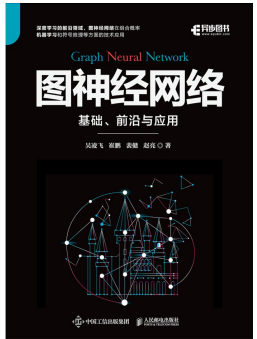
Graph Matching Networks w/ SPT [1]



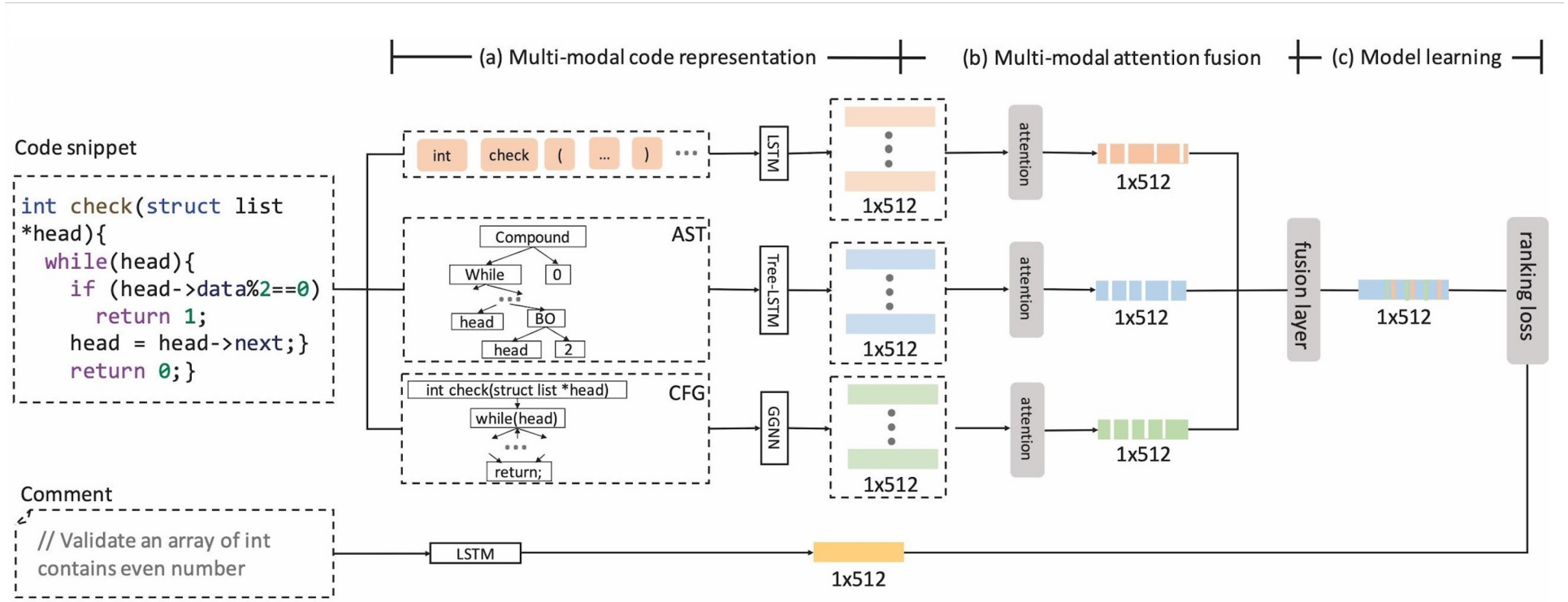
Multilevel Graph Matching Networks [2]

1. Abstract syntax tree (AST): more semantic information and capture the “problem” semantics
2. Both **Node-graph-interaction**, node-node-interaction and graph-graph interaction

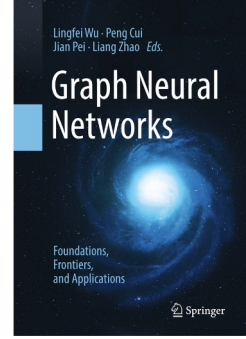
[1] Li Y, et al. Graph matching networks for learning the similarity of graph structured objects. InInternational conference on machine learning 2019 May 24 (pp. 3835-3845). PMLR
[2] Ling X, et al. Multilevel Graph Matching Networks for Deep Graph Similarity Learning. IEEE Transactions on Neural Networks and Learning Systems. 2021 Aug 18

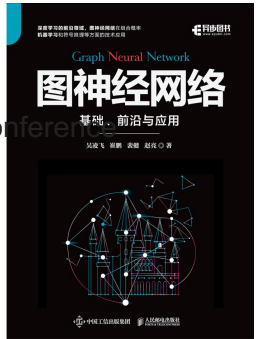


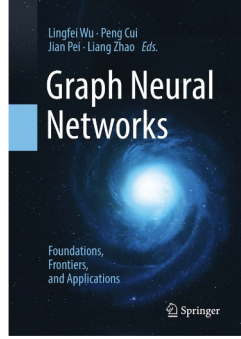
Case Study 3: code similarity analysis



Wan Y, Shu J, Sui Y, Xu G, Zhao Z, Wu J, Yu P. Multi-modal attention network learning for semantic source code retrieval. In2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE) 2019 Nov 11 (pp. 13-25). IEEE.

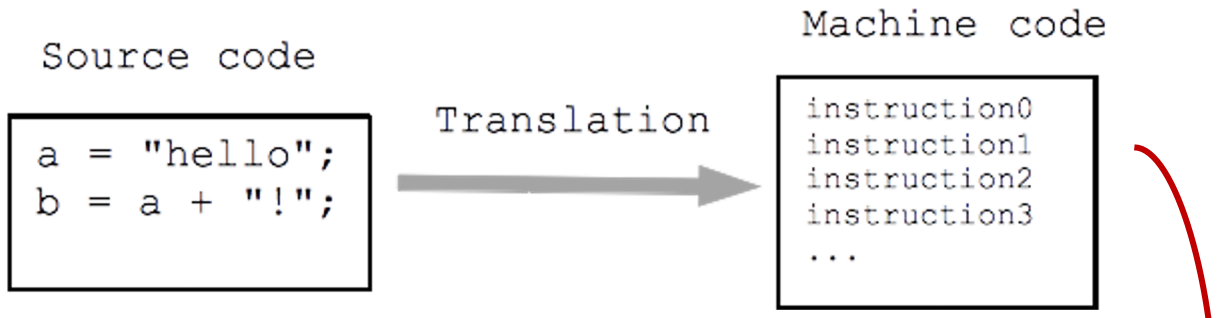




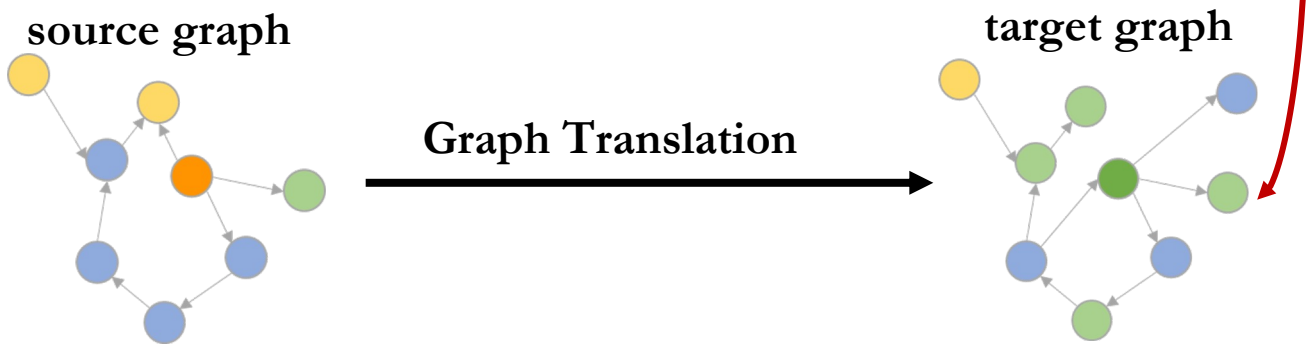


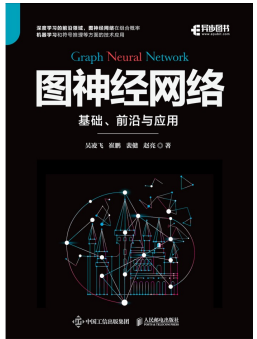
Case Study 4: Code Translation

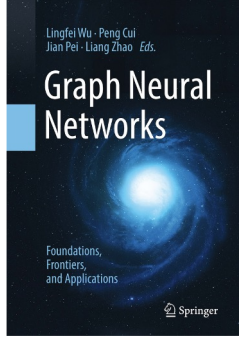
Code Translation: Translating source code from one language to another language.
e.g., program modernization



Graph-to-graph translation: Learning the mapping from graph in the source domain to the graph in the target domain.

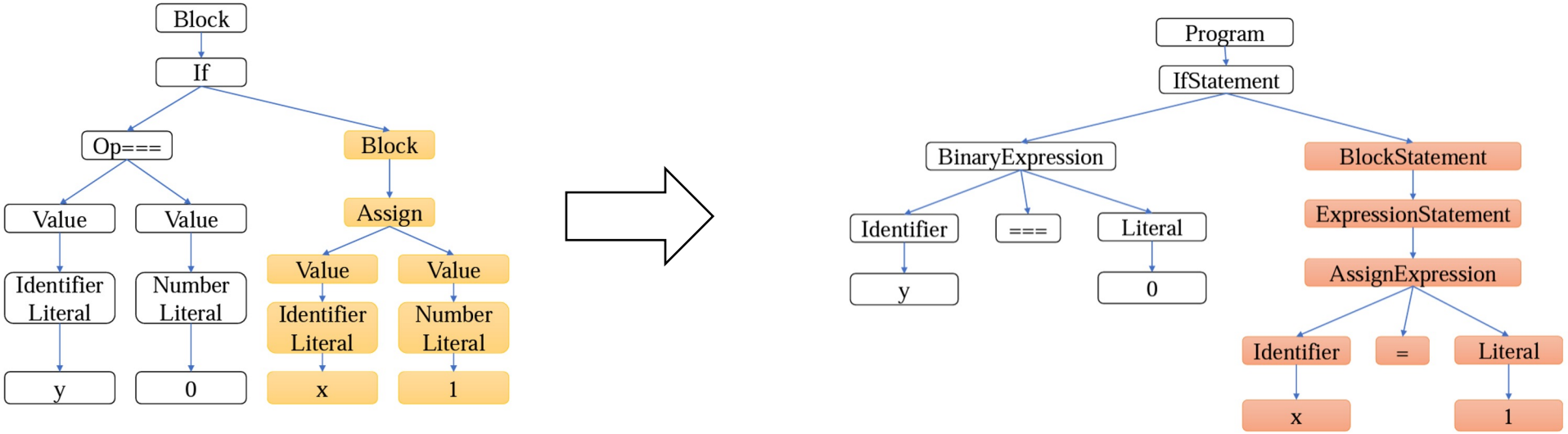




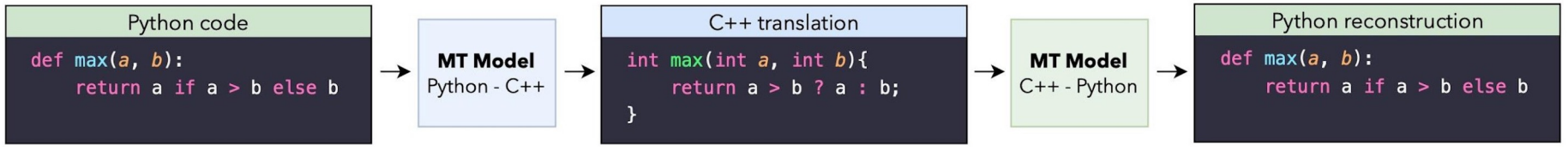


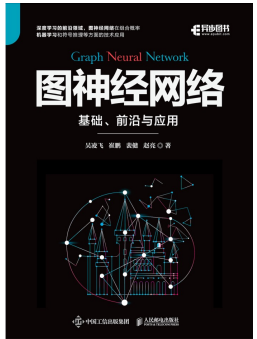
Case Study 4: Code Translation

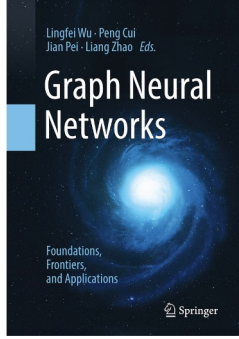
Supervised setting: Tree-to-tree neural networks for program translation (Chen X et al,. 2018)



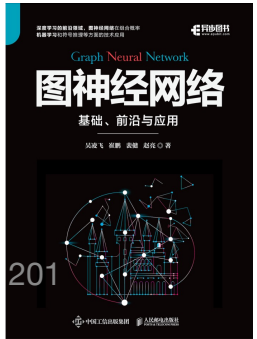
Unsupervised setting: Unsupervised Circle Translation (Roziere B et al,. 2020)

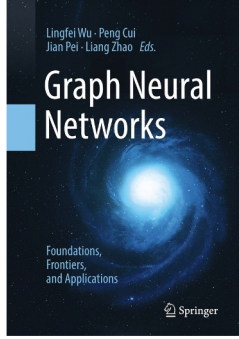






GNNs in Protein Modeling (Chapter 25)

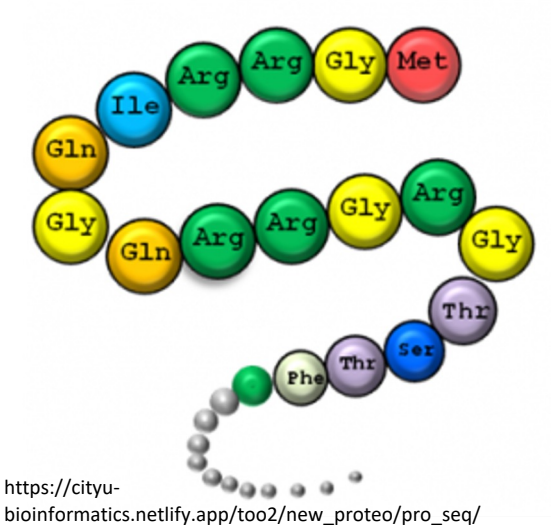




From Protein Interactions to Function

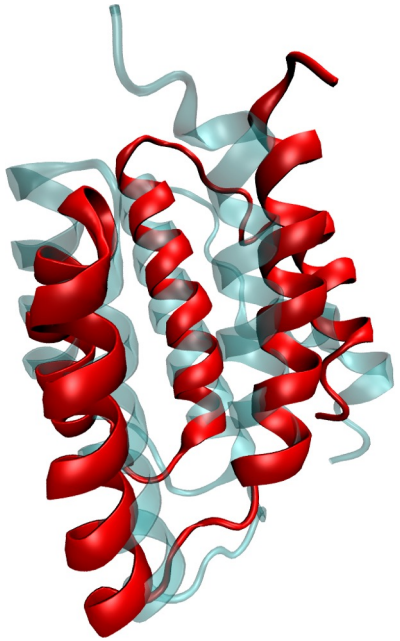
- Millions of protein products in database to be explored

  -----What function a protein molecule performs.

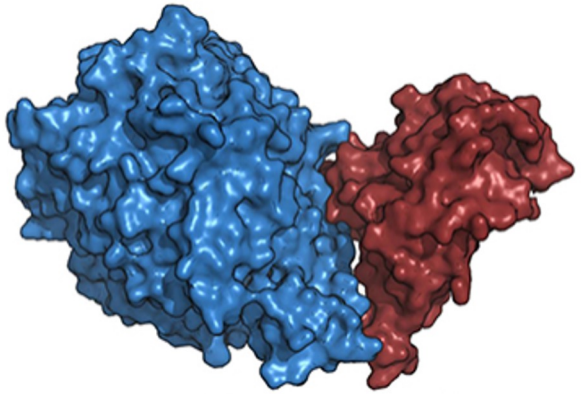


https://cityu-bioinformatics.netlify.app/too2/new_proteo/pro_seq/

Biological sequence

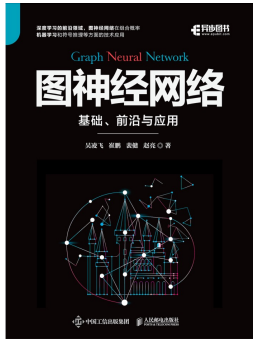


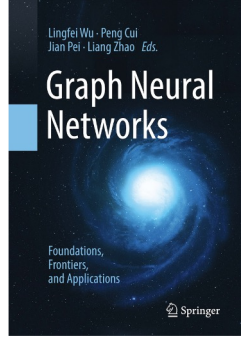
Protein structure



(Anderson Brito et al,. 2017)

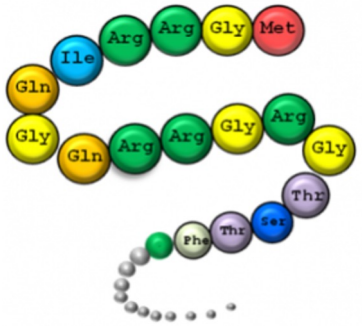
Protein Interaction



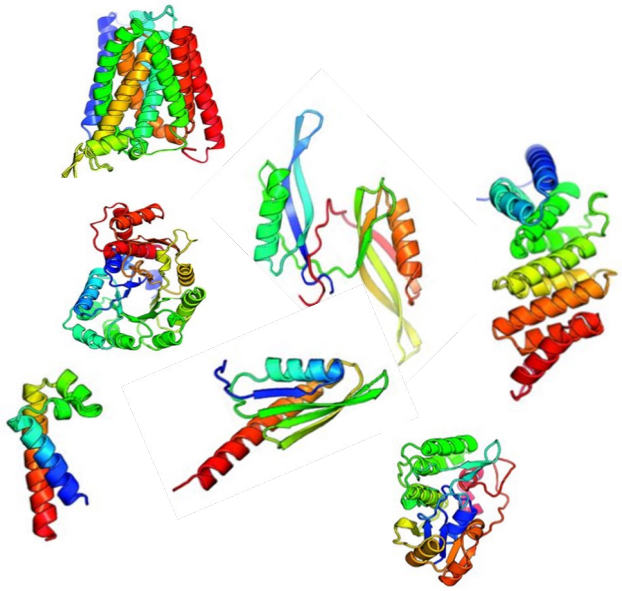


Case Study: De-novo protein generation

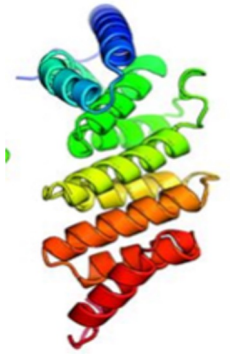
- **Protein structure prediction (PSP)** seeks to determine one or more biologically-active/native forms/structures of a protein molecule from the sequence of amino acids chain.



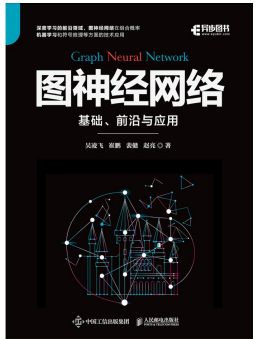
Amino acids chain

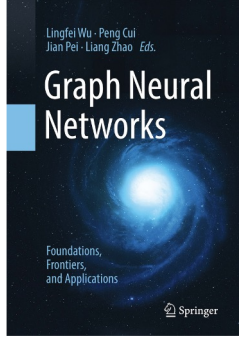


2^N complexity



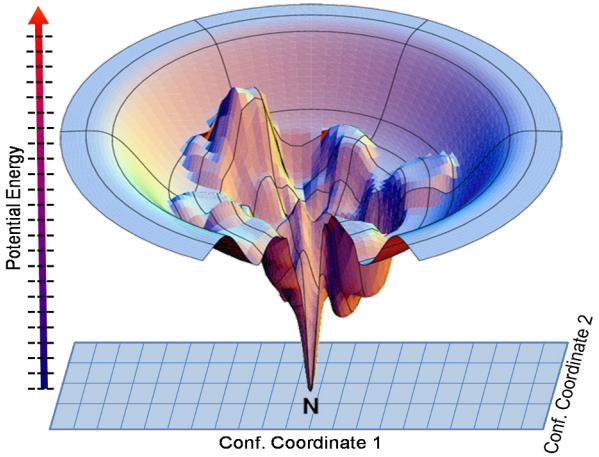
Protein Tertiary Structure



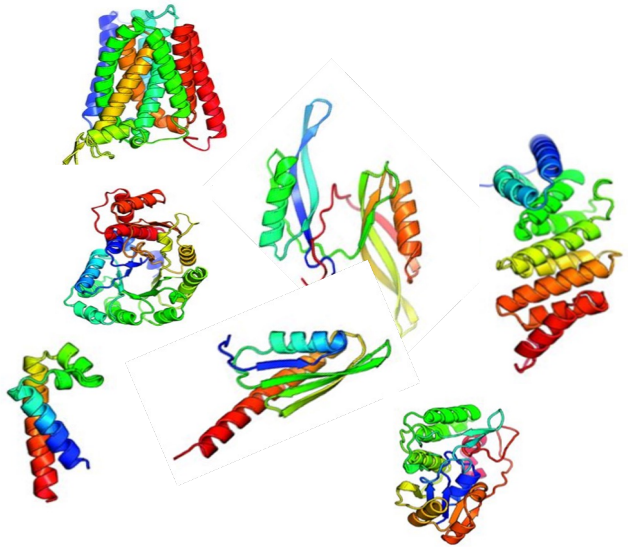


Case Study: De-novo protein generation

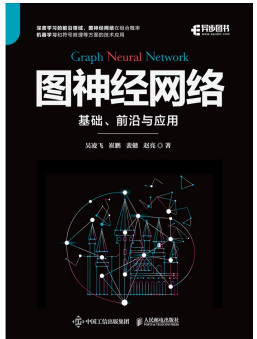
Given tertiary structures, can the model generate protein-like (physically realistic) tertiary structures without any particular amino acid sequence in mind?



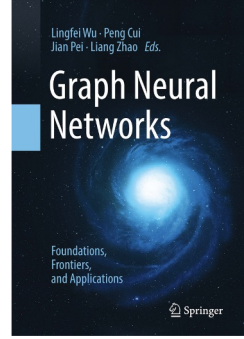
Protein structure space



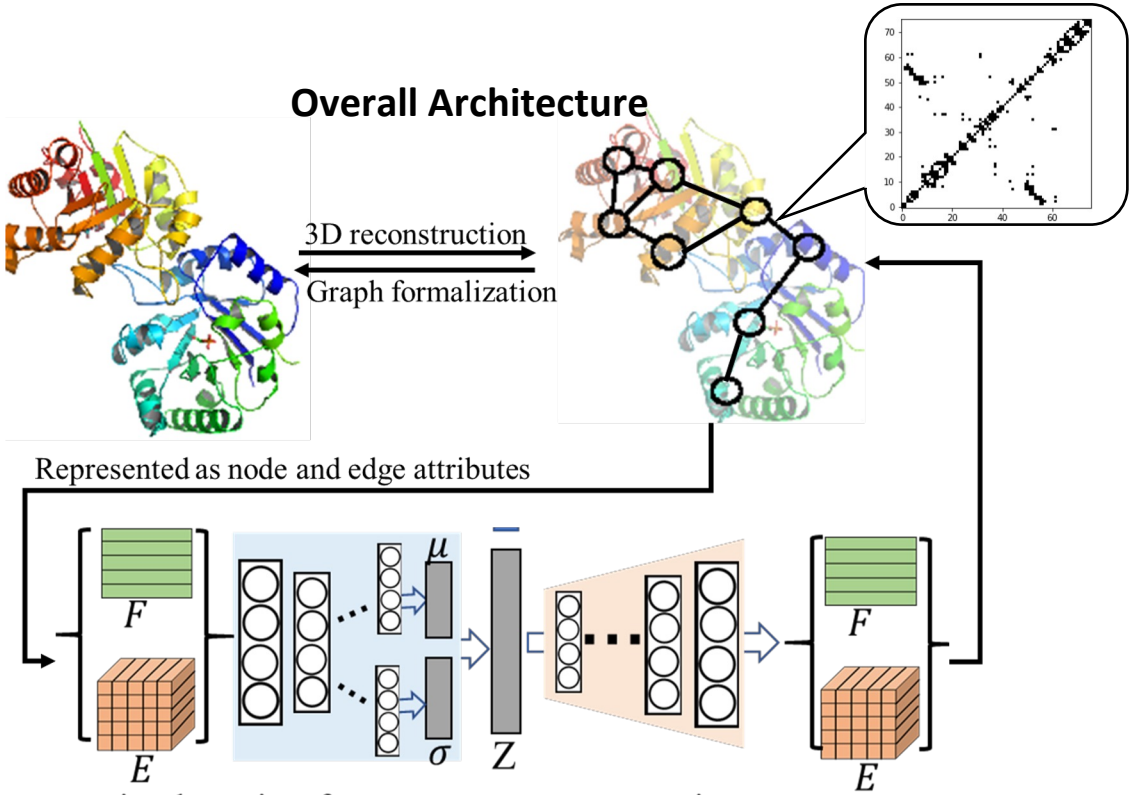
More various structures



Case Study: De-novo protein generation



Learning the protein structure distribution via Variational autoencoder



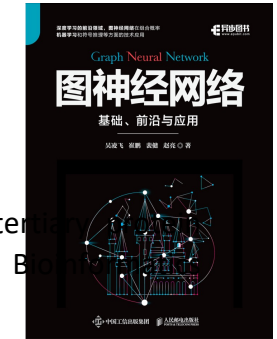
Protein contact graph: $G = (E, F)$

Objective function:

$$\ell = \mathbb{E}_{q(Z|F,E)}[\log p(E, F|Z)] - KL[q(Z|F, E)||p(Z)]$$

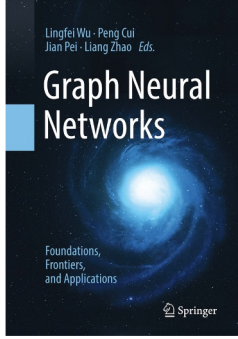
decoder encoder

$$q(Z|F, E) = \prod_{i=1}^{N} q(z_i|F, E), \text{ where } q(z_i|F, E) = \mathcal{N}(z_i|\mu_i, \sigma_i^2),$$

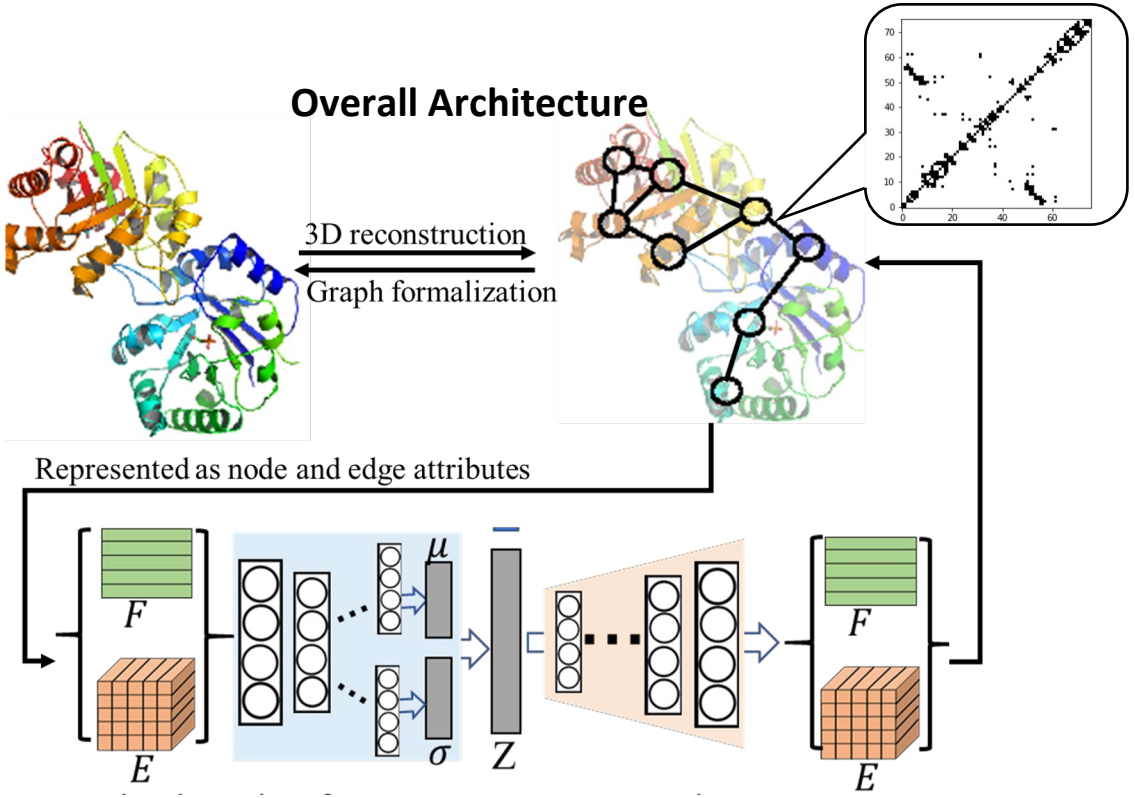


Guo X, Du Y, Tadepalli S, Zhao L, Shehu A. Generating tertiary protein structures via interpretable graph variational autoencoders. Bio-inspired Advances. 2021;1(1):vbab036.

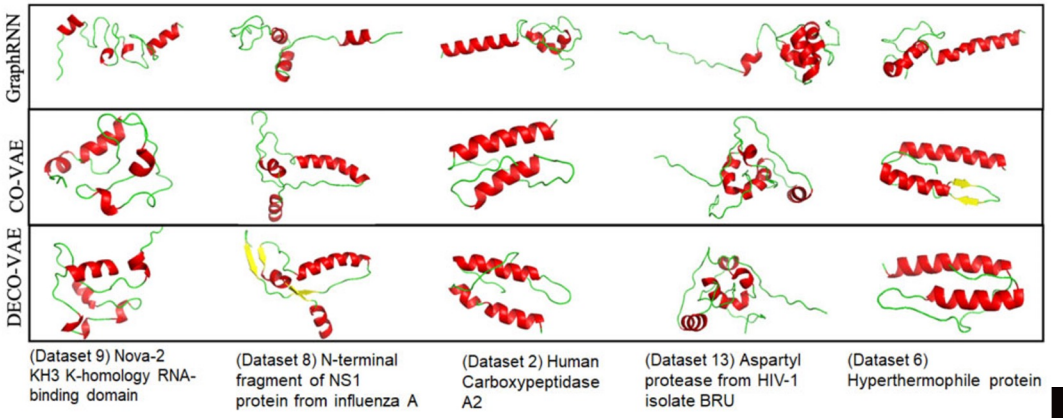
Case Study: De-novo protein generation



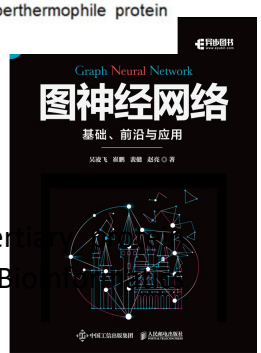
Learning the protein structure distribution via Variational autoencoder

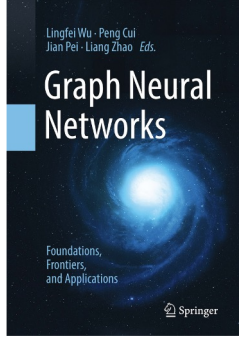


Generated protein structures



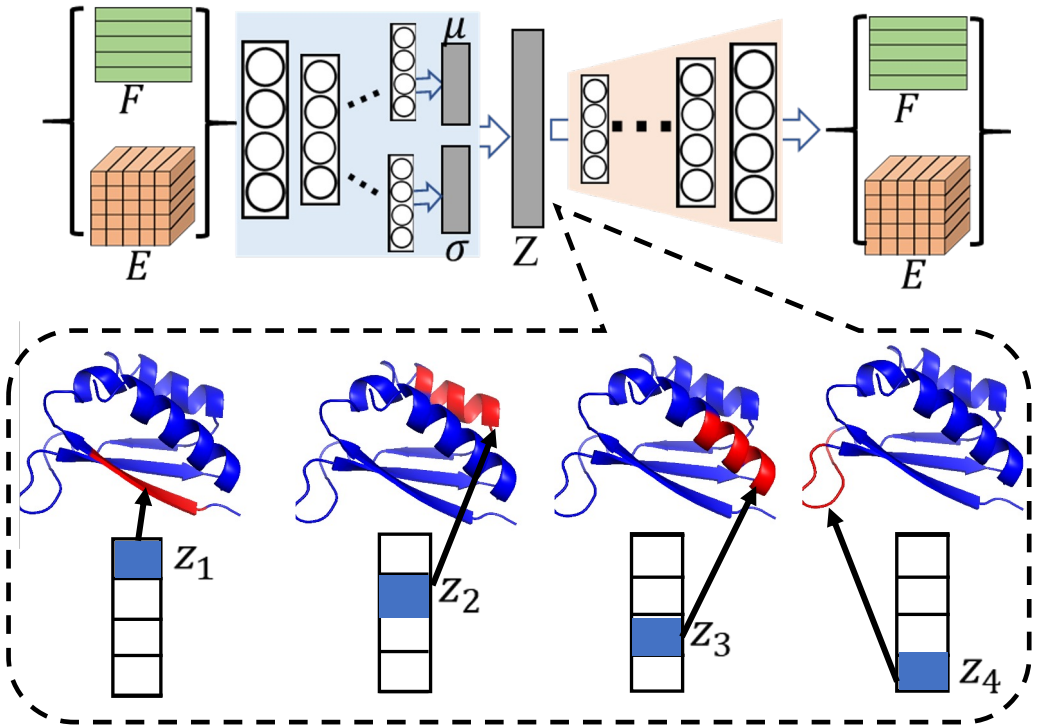
Guo X, Du Y, Tadepalli S, Zhao L, Shehu A. Generating tertiary protein structures via interpretable graph variational autoencoders. Bio-inspired Advances. 2021;1(1):vbab036.





Case Study: De-novo protein generation

Interpretability Enhancement



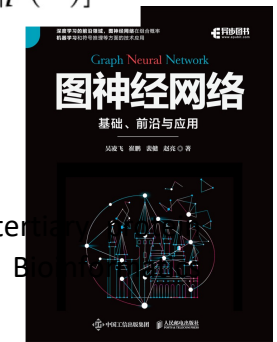
Protein contact graph: $G = (E, F)$

Objective function:

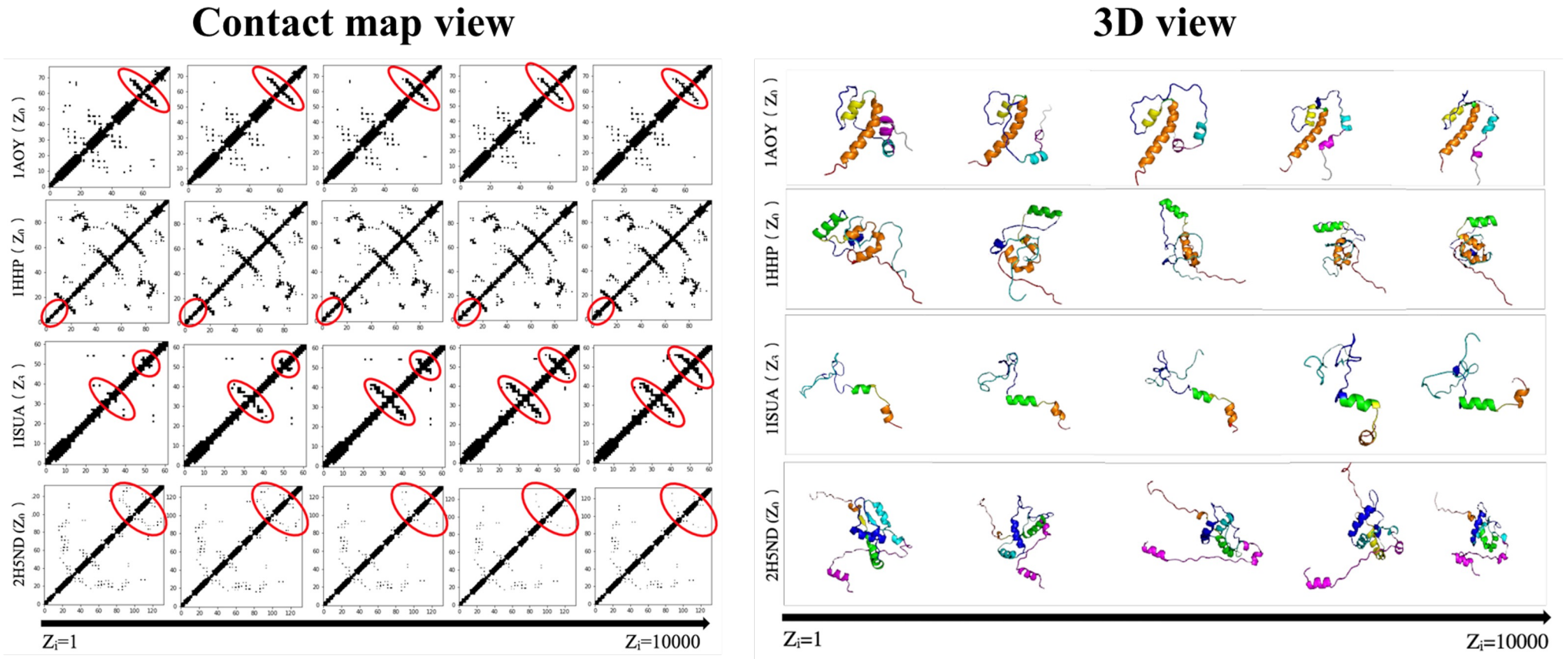
$$\ell = \mathbb{E}_{q(Z|F,E)}[\log p(E, F|Z)] - KL[q(Z|F, E)||p(Z)]$$

$$\ell = \mathbb{E}_{q(Z|F,E)}[\log p(E, F|Z)] - \beta KL[q(Z|F, E)||p(Z)].$$

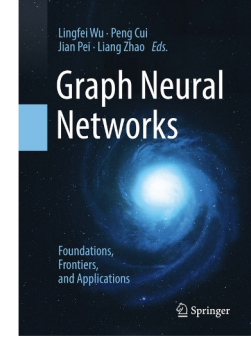
Guo X, Du Y, Tadepalli S, Zhao L, Shehu A. Generating tertiary protein structures via interpretable graph variational autoencoders. Bio-inspired Advances. 2021;1(1):vbab036.

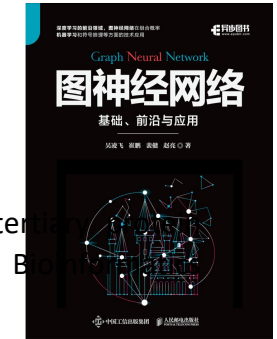


Case Study: De-novo protein generation

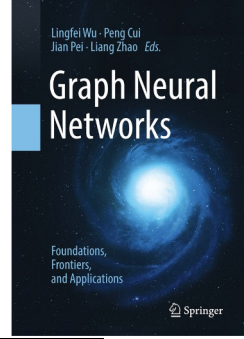


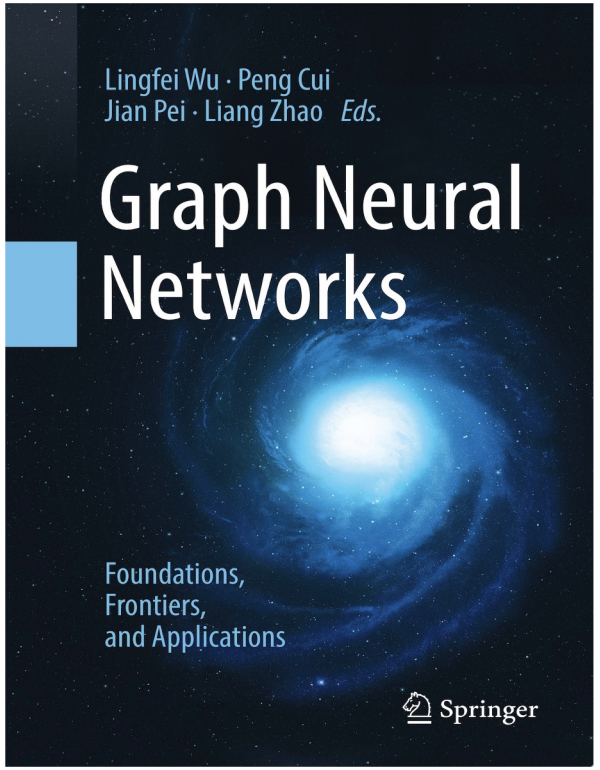
Guo X, Du Y, Tadepalli S, Zhao L, Shehu A. Generating tertiary protein structures via interpretable graph variational autoencoders. Bio-inspired Advances. 2021;1(1):vbab036.





Graph Neural Networks Book





Free download: https://graph-neural-networks.github.io/index.html

The English version of the book is available on Amazon!

The Chinese version of the book (图神经网络中文城堡书) is available on JD.com!