media and network lab

KDD2021

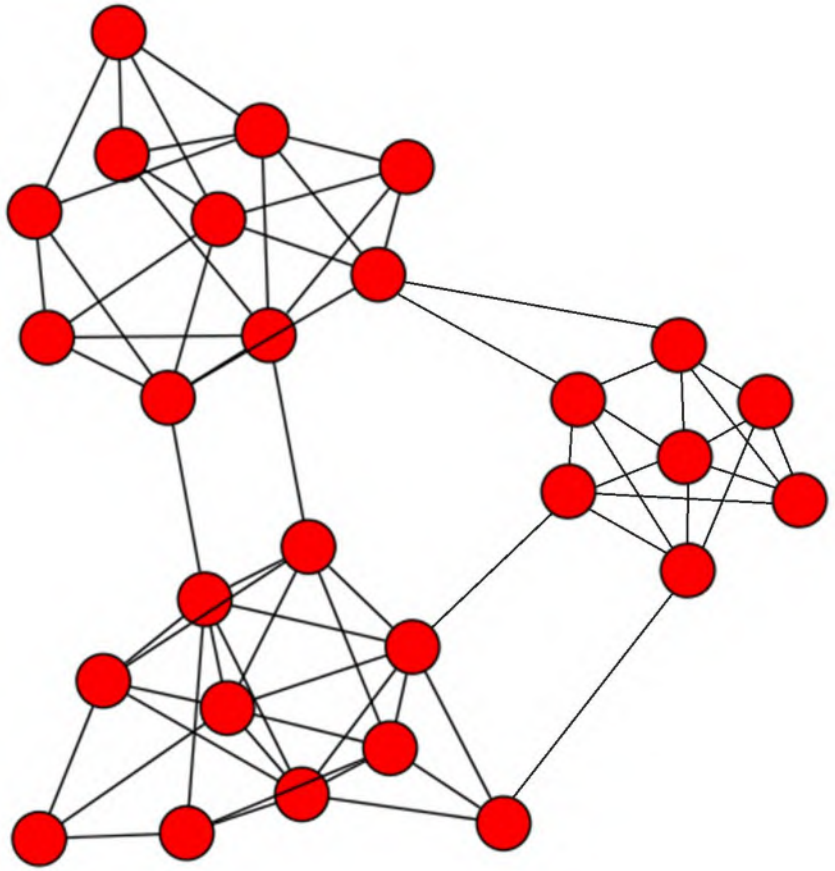
Automated Machine Learning on Graph

Xin Wang, Ziwei Zhang, Wenwu Zhu

Tsinghua University

Network (Graph)

The general description of data and their relations.

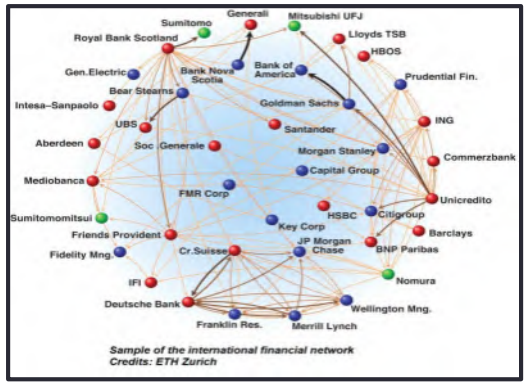


Many types of data are networks

Social Networks

Biology Networks

Finance Networks



Internet of Things

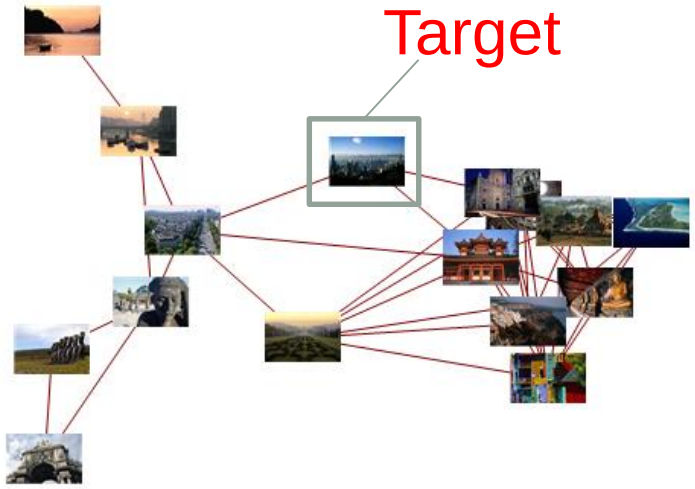
Information Networks

Logistic Networks

Why network is important?

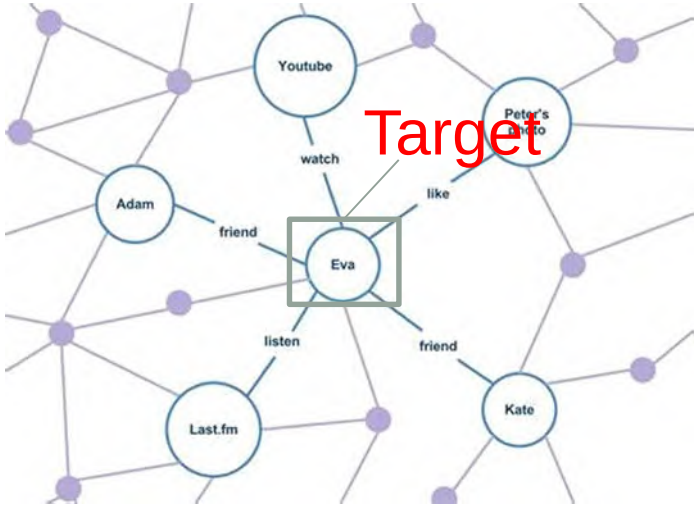
In few cases, you only care about a subject but not its relations with other subjects.

Image Characterization



Reflected by relational subjects

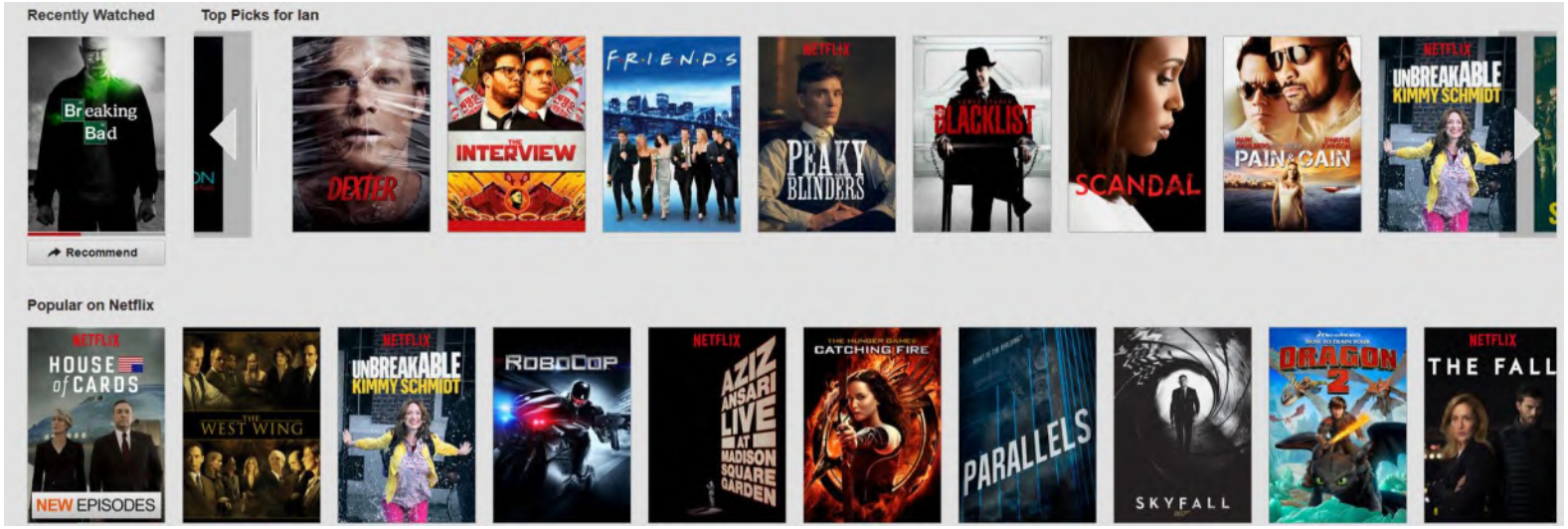
Social Capital



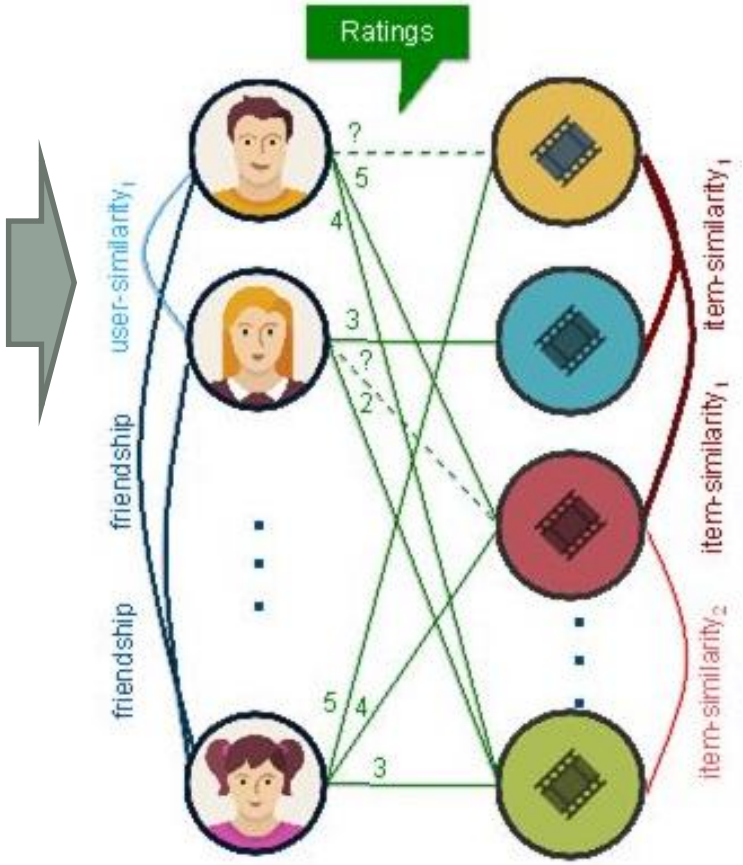
Decided by relational subjects

Many applications are intrinsically network problems

Recommendation Systems



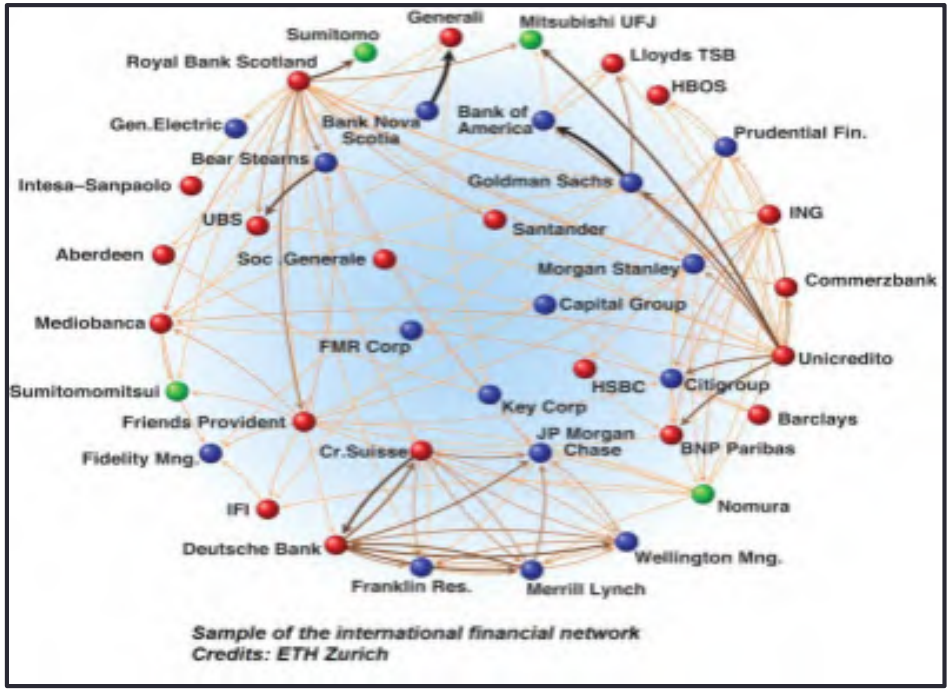
Link prediction in bipartite graphs



Many applications are intrinsically network problems

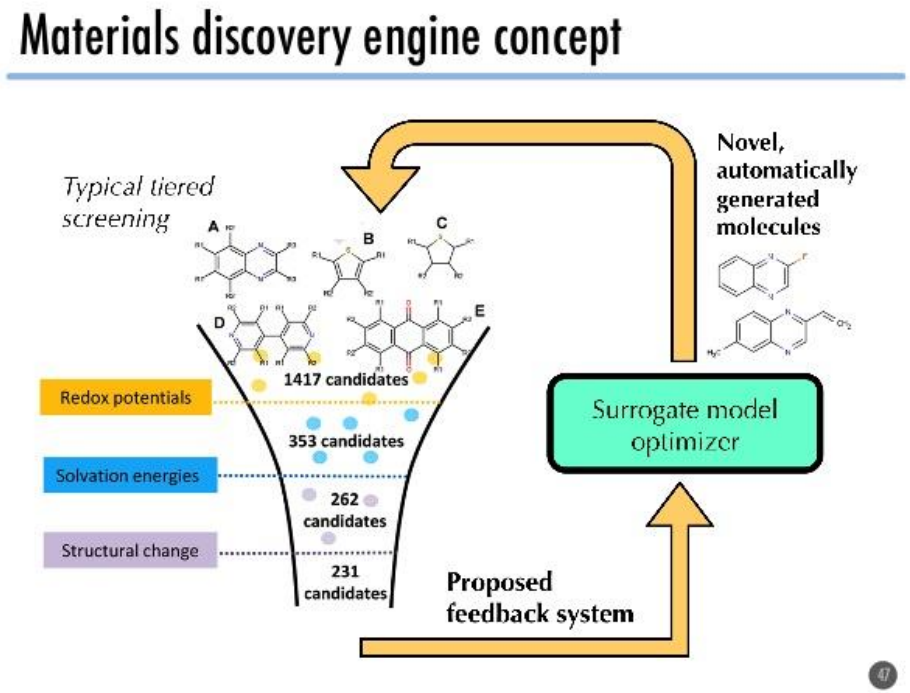
Financial credit & risk management

Node importance & classification

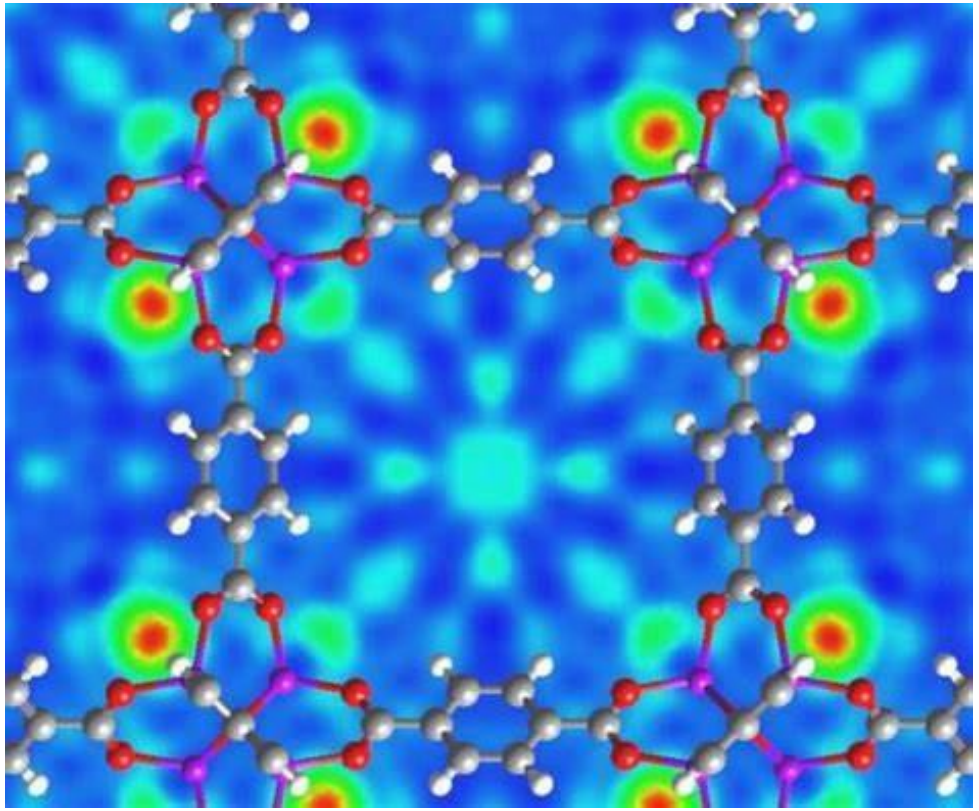


Many applications are intrinsically network problems

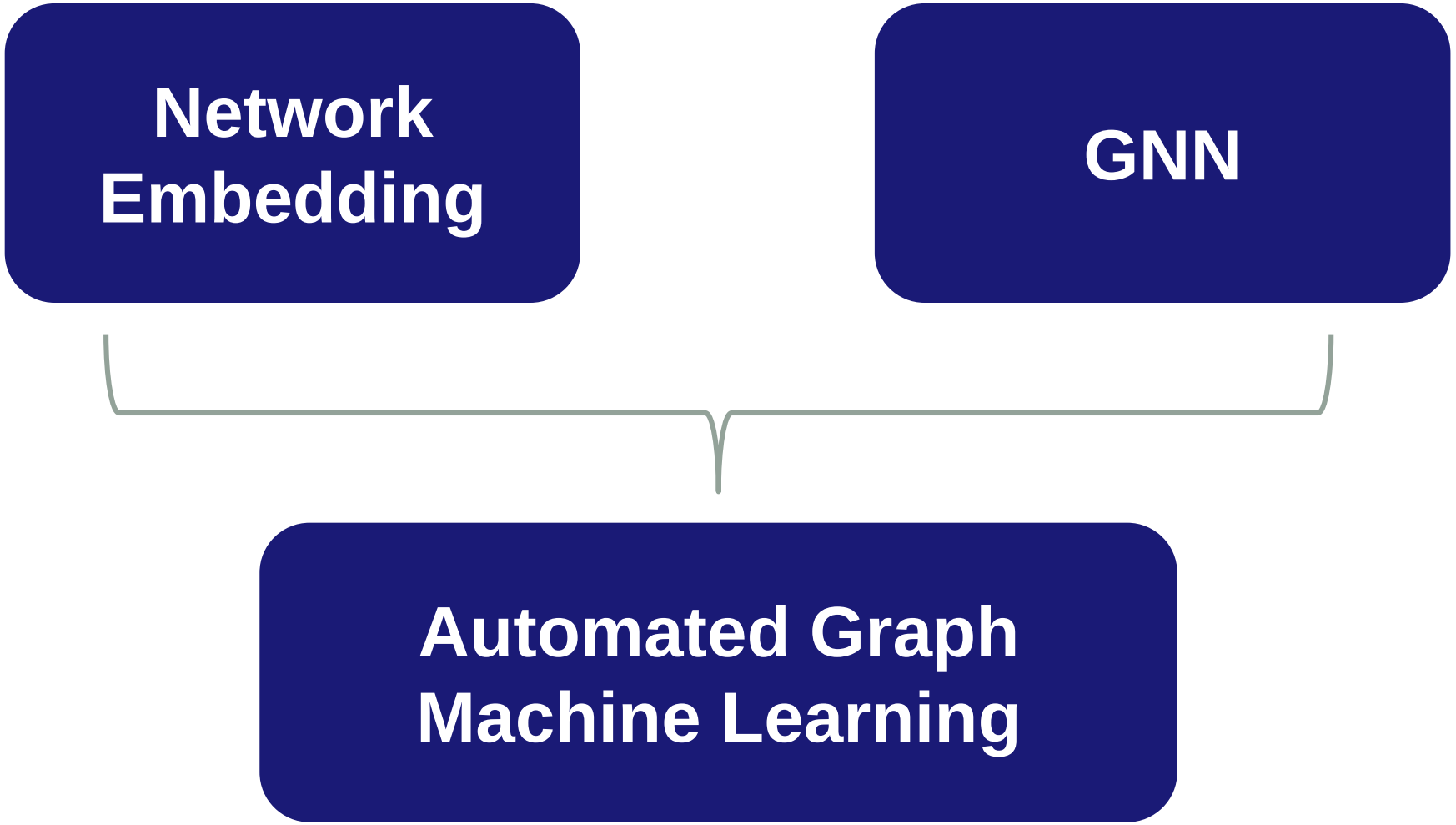
New material discovery



Subgraph pattern discovery



Learning from networks

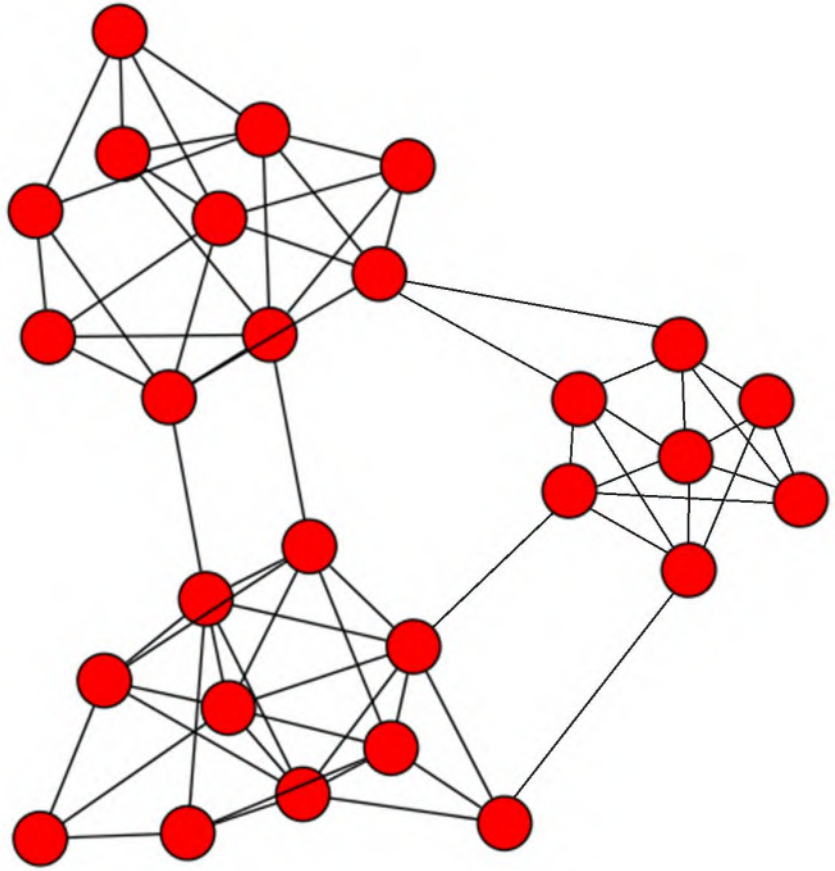


Learning from networks

Network Embedding

Network Embedding

G = (V, E)

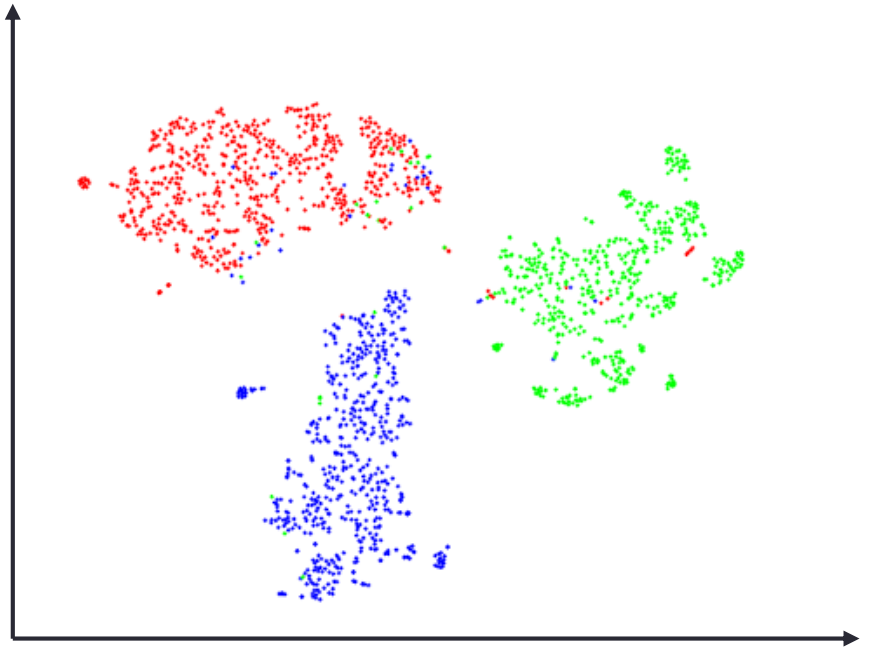


generate

embed

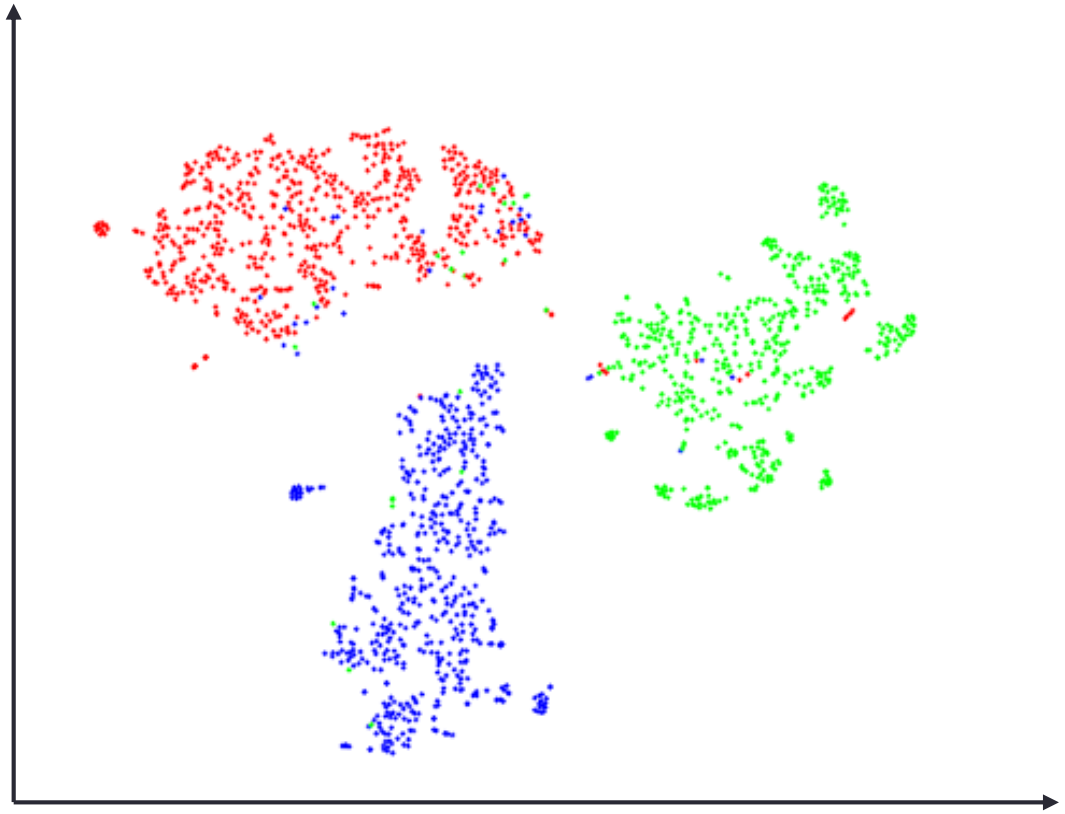
G = (V)

Vector Space



- Easy to parallel
- Can apply classical ML methods

The ultimate goal



Network Inference

- Node importance
- Community detection
- Network distance
- Link prediction
- Node classification
- Network evolution
- …

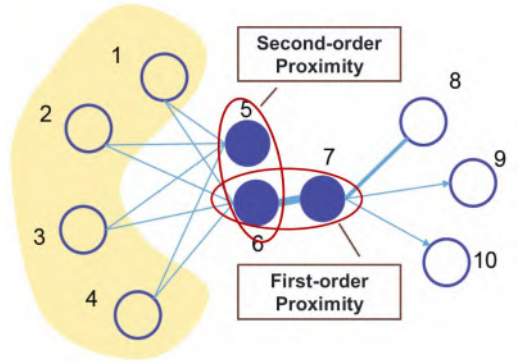
in *Vector Space*

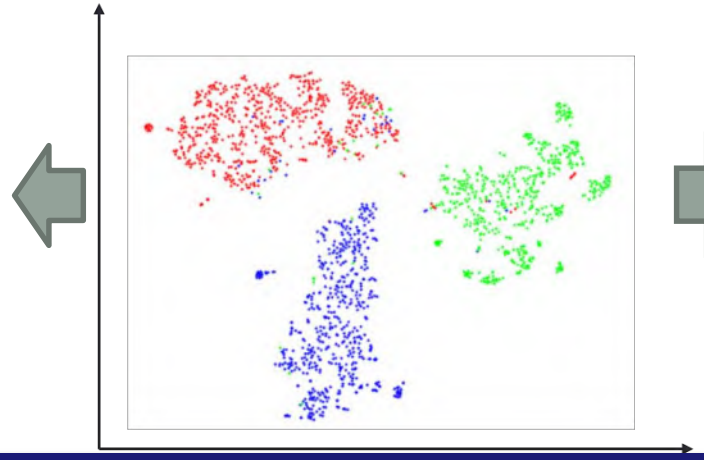
The goal of network embedding

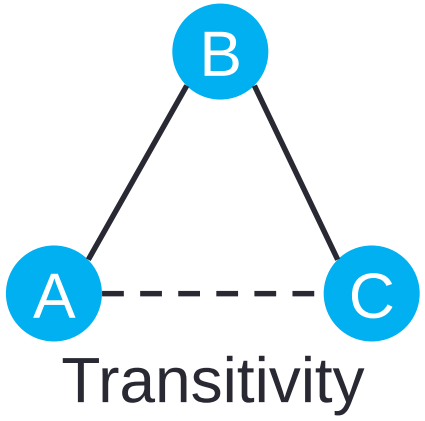
Goal Support network inference in vector space

Reflect network structure

Maintain network properties

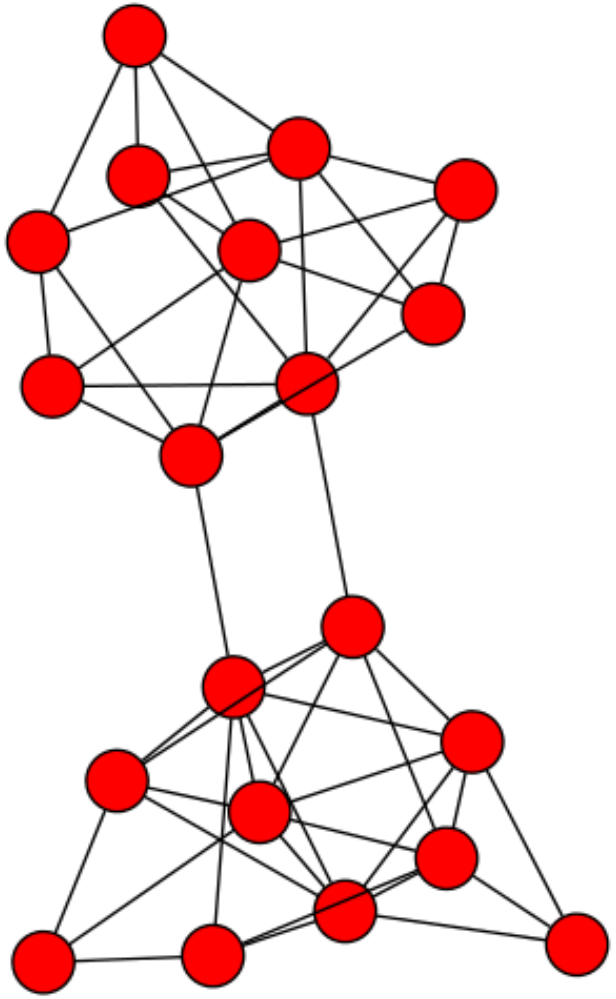






Transform network nodes into vectors that are fit for off-the-shelf machine learning models

Network Structures



Nodes & Links

Pair-wise Proximity

Community Structures

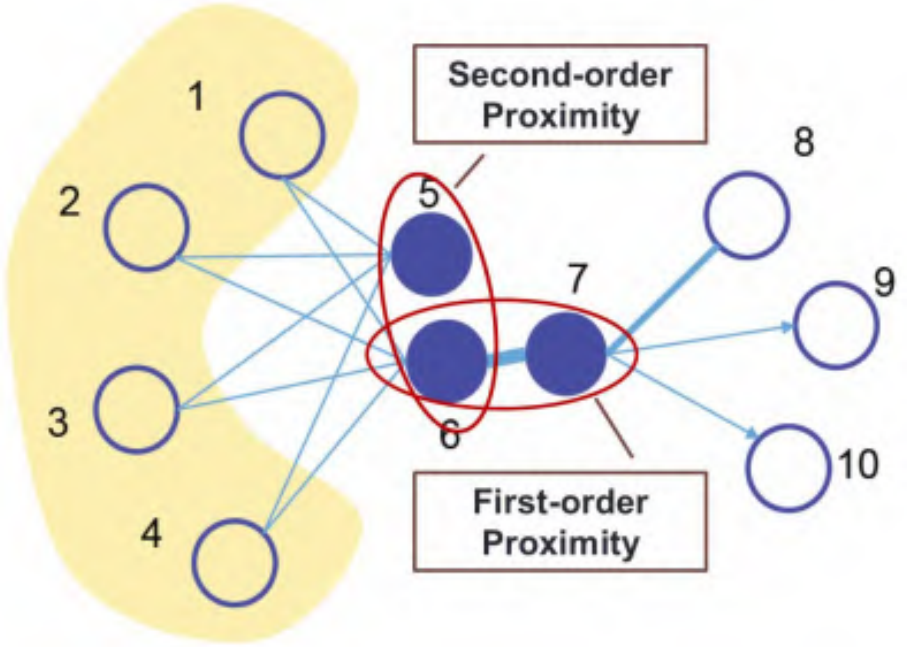
Hyper Edges

Global Structure

Dedicated Network Embedding Tutorial at KDD 2019:
http://cuip.thumedialab.com/papers/KDD19%20Tutorial%20on%20NE_Peng.pdf

High-Order Proximity

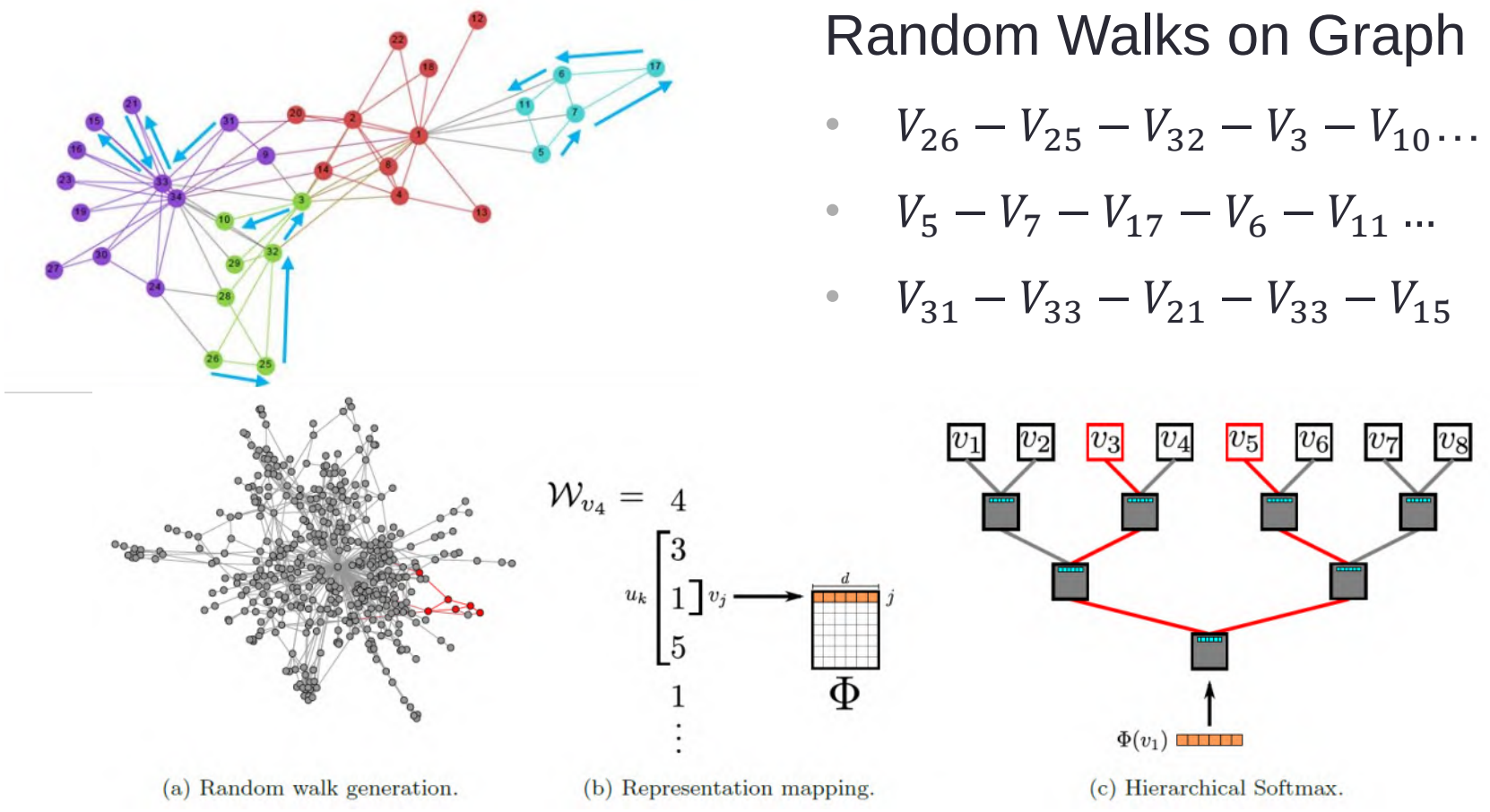
- Capturing the underlying structure of networks



- Advantages:
  - Solve the sparsity problem of network connections
  - Measure indirect relationship between nodes

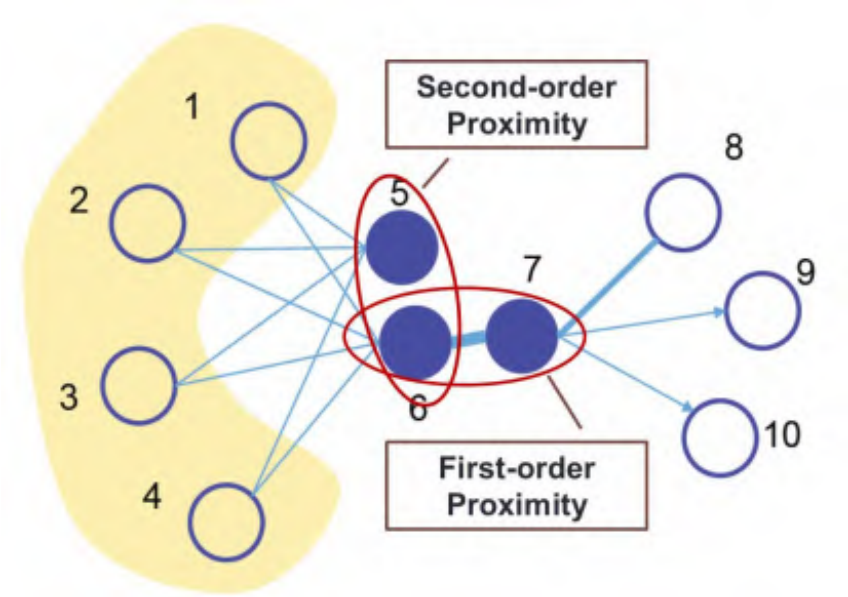
DeepWalk

- Exploit truncated random walks to define neighborhoods of a node.



B. Perozzi et al. Deepwalk: Online learning of social representations. *KDD 2014.*

LINE



LINE with First-order Proximity: local pairwise

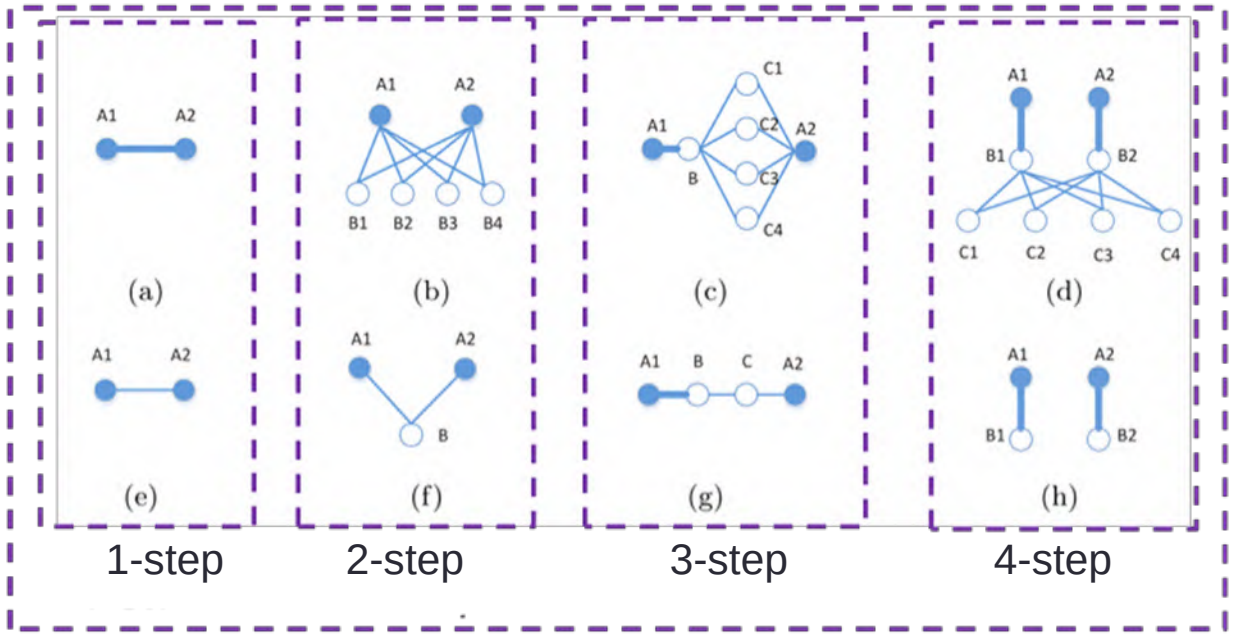
$$O_1 = -\sum_{(i,j)\in E} w_{ij} \log p_1(v_i, v_j)$$

LINE with Second-order Proximity: neighborhood structures

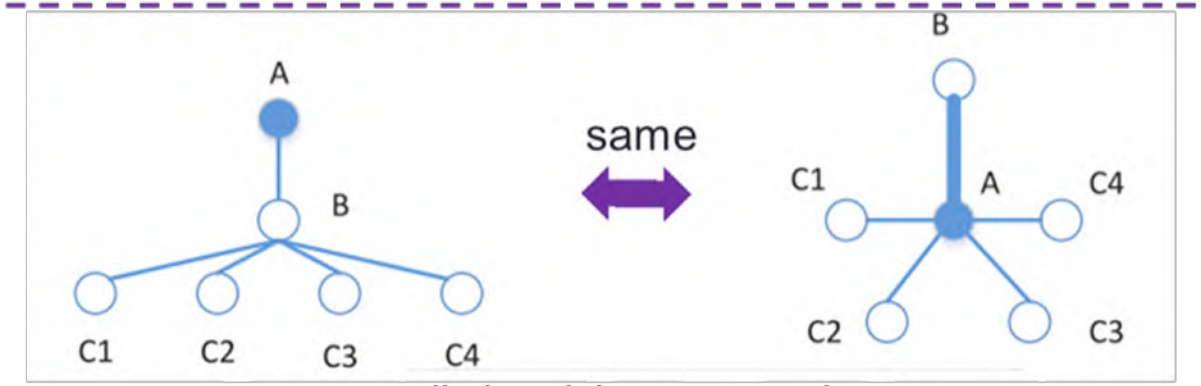
$$O_2 = \sum_{i\in V} \lambda_i d(\hat{p}_2(\cdot|v_i), p_2(\cdot|v_i))$$

Jian Tang et al. LINE: Large-scale Information Network Embedding. *WWW 2015*.

GraRep



capturing different k-step information



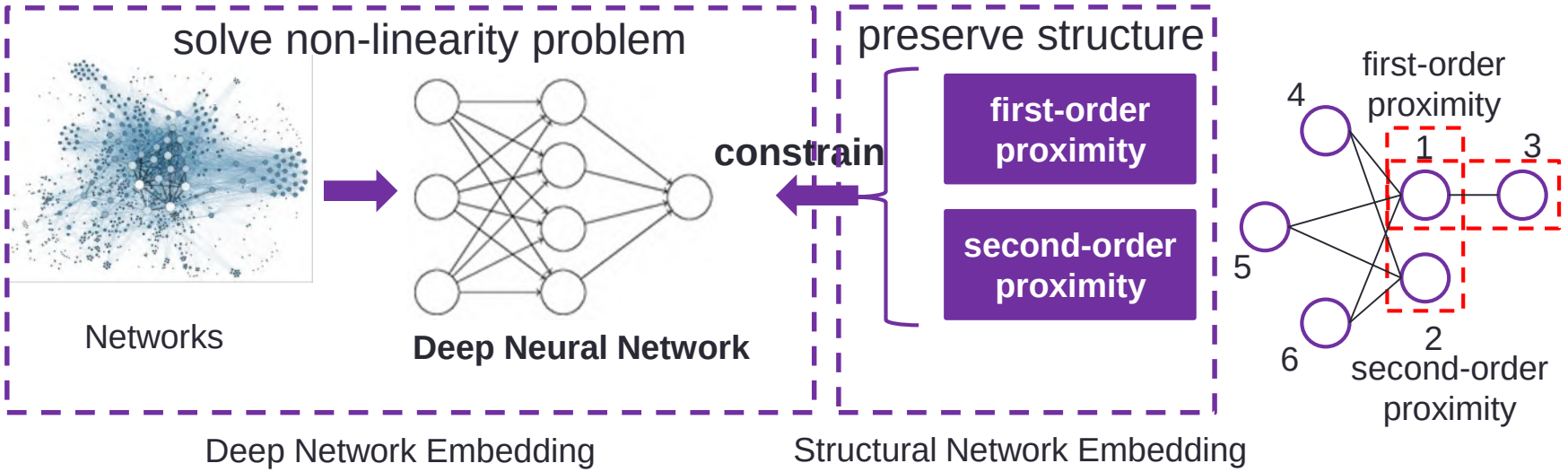
Do not distinguish 1-step and 2-step.

maintaining different k-step information separately

Shaosheng Cao et al. GraRep: Learning Graph Representations with Global Structural Information. *CIKM 2015*.

Structural Deep Network Embedding

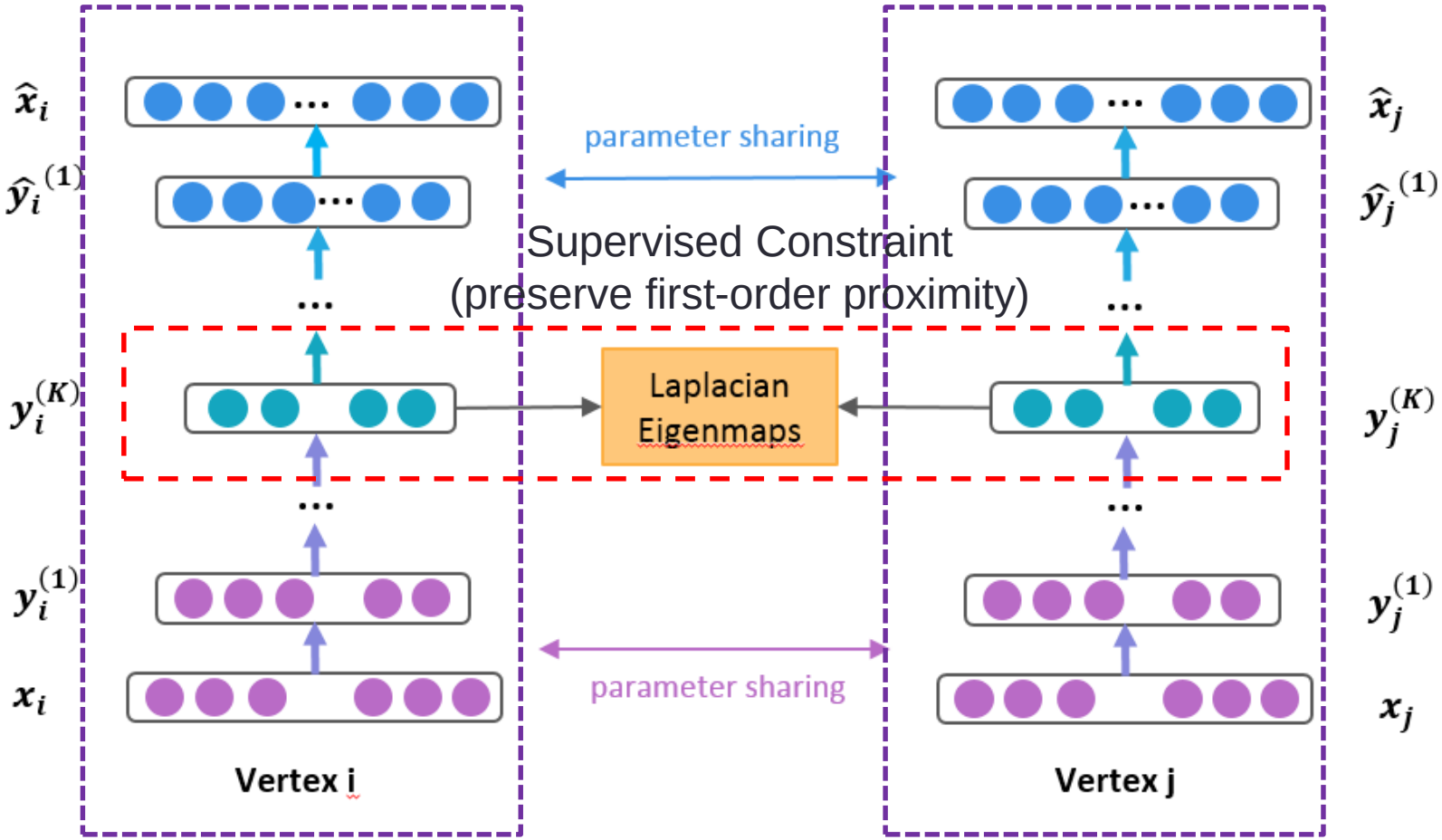
- Idea



- Challenges
  - How to represent network data in deep neural networks?
  - How to preserve the first and second-order proximity in deep neural networks?

Daixin Wang, Peng Cui, Wenwu Zhu. Structural Deep Network Embedding. *KDD*, 2016.

Framework of SDNE

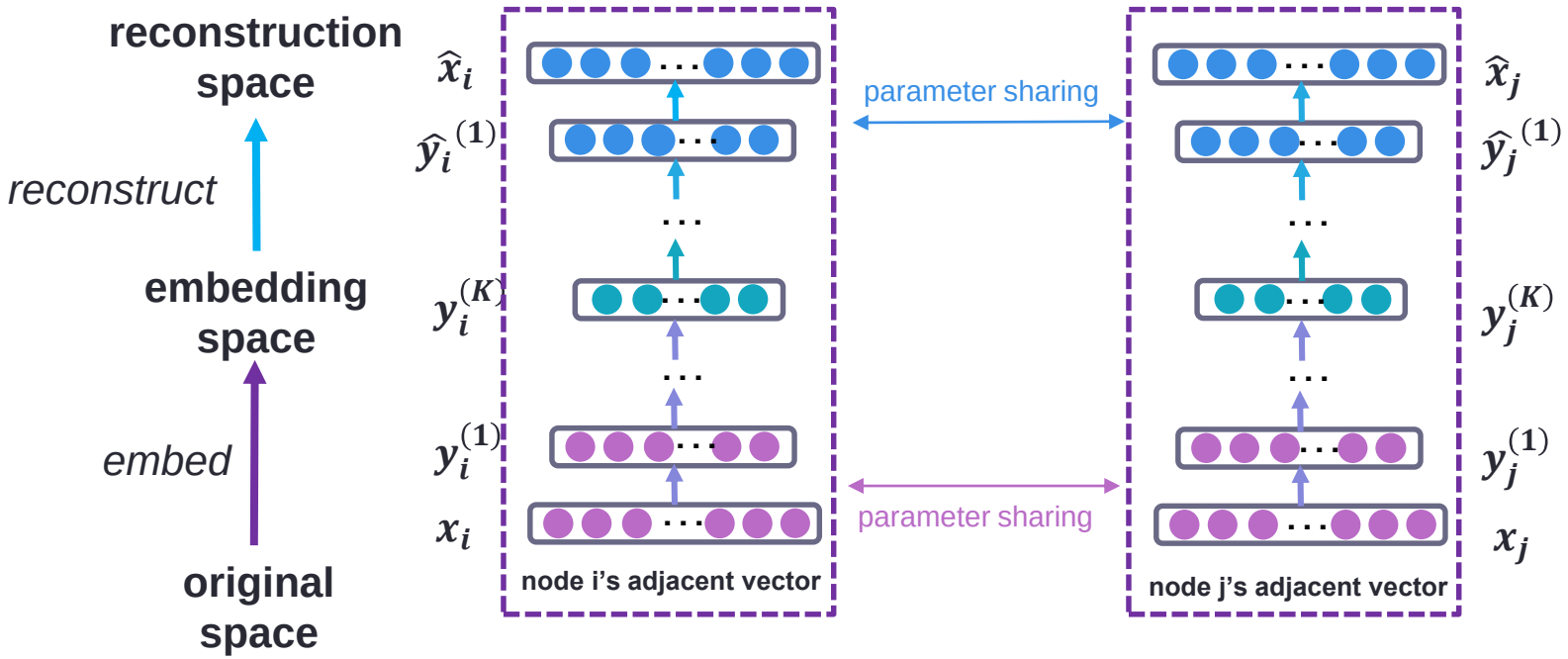


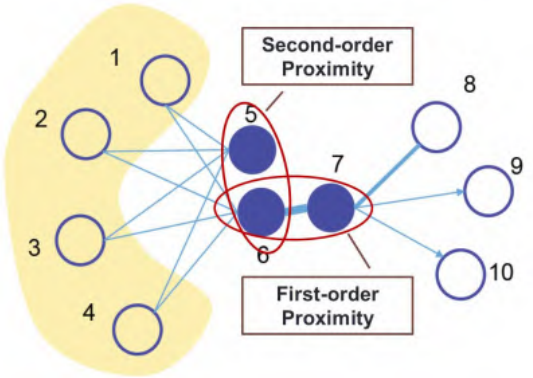
Unsupervised Autoencoder
(preserve second-order proximity)

Unsupervised Autoencoder
(preserve second-order proximity)

Daixin Wang, Peng Cui, Wenwu Zhu. Structural Deep Network Embedding. *KDD*, 2016.

Preserve second-order proximity





Similar neighborhoods

↓

Similar adjacent vector

↓

Similar embedding vector

- Reconstruct the neighborhood structure of each vertex through deep autoencoder

$$L_{2nd} = \| \left(\hat{X} - X\right) \odot B \|_F^2$$

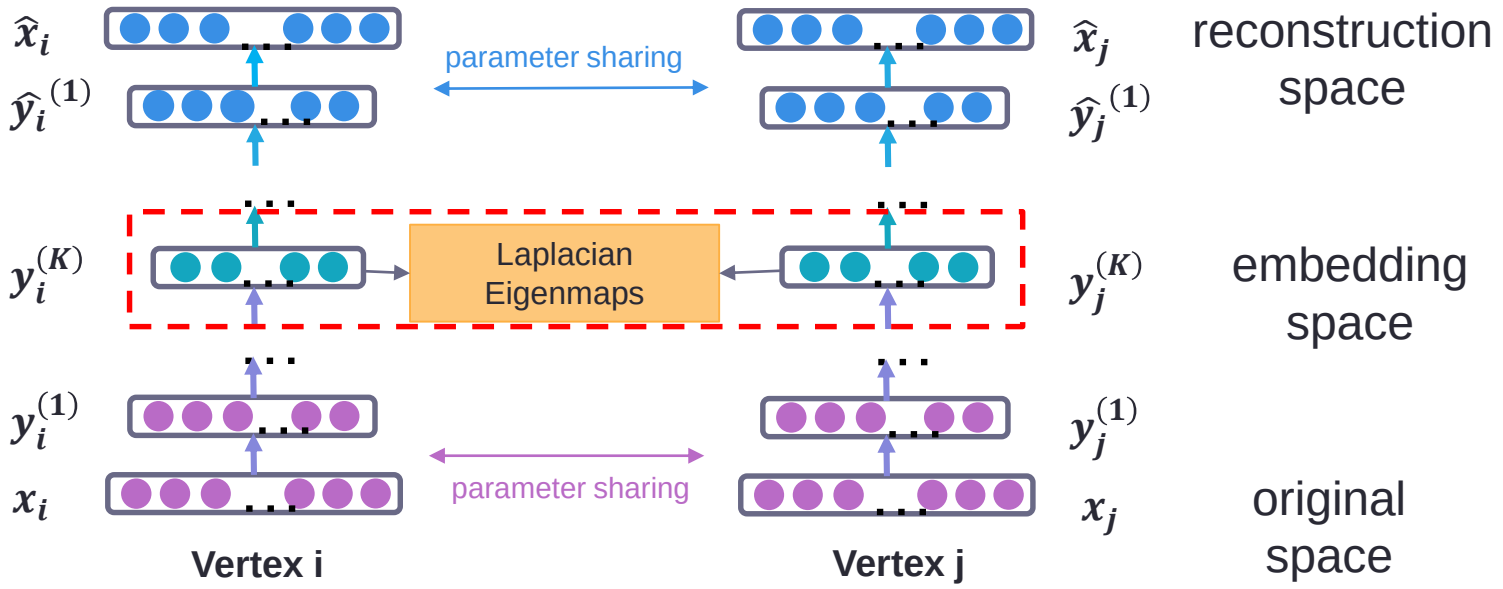
reconstruction of adjacency matrix

adjacency matrix

balancing weight

Daixin Wang, Peng Cui, Wenwu Zhu. Structural Deep Network Embedding. *KDD*, 2016.

Preserve first-order proximity



- Incur penalty when connected vertexes are mapped far away in the embedding space

$$L_{1st} = \sum_{i,j=1}^{n} a_{i,j} \parallel y_i - y_j \parallel_2^2$$

Connection between the i-th and j-th vertex

The embedding of the $i(j)$-th vertex

Daixin Wang, Peng Cui, Wenwu Zhu. Structural Deep Network Embedding. *KDD*, 2016.

Algorithm

- Objective Function

second-order loss　　first-order loss

$$L = L_{2nd} + \alpha L_{1st} + \upsilon L_{reg} \longrightarrow \text{regularization term}$$

$$= \| \left(\hat{X} - X\right) \odot B \|_F^2 + \alpha \sum_{i,j=1}^{n} s_{i,j} \| y_i - y_j \|_2^2 + \upsilon L_{reg}$$

- Algorithm

Algorithm 1 Training Algorithm for the semi-supervised deep model of *SDNE*

Input: the network $G = (V, E)$ with adjacency matrix S, the parameters α and ν

Output: Network representations Y and updated Parameters: θ

1: Pretrain the model through deep belief network to obtain the initialized parameters $\theta = \{\theta^{(1)}, ..., \theta^{(K)}\}$
2: $X = S$
3: **repeat**
4: Based on X and θ, apply Eq. 1 to obtain $\hat{X}$ and $Y = Y^K$.
5: $\mathcal{L}_{mix}(X; \theta) = \|(\hat{X} - X) \odot B\|_F^2 + 2\alpha tr(Y^T L Y) + \nu \mathcal{L}_{reg}$.
6: Based on Eq. 6, use $\partial L_{mix} / \partial \theta$ to back-propagate through the entire network to get updated parameters θ.
7: **until** converge
8: Obtain the network representations $Y = Y^{(K)}$

Daixin Wang, Peng Cui, Wenwu Zhu. Structural Deep Network Embedding. *KDD*, 2016.

Experimental Results

The precision keeps at least 0.9

Table 5: *precision@k* on ARXIV GR-QC for link prediction

Algorithm	*P@2*	*P@10*	*P@100*	*P@200*	*P@300*	*P@500*	*P@800*	*P@1000*	*P@10000*
SDNE	**1**	**1**	**1**	**1**	**1***	**0.99****	**0.97****	**0.91****	**0.257****
LINE	1	1	1	1	0.99	0.936	0.74	0.79	0.2196
DeepWalk	1	0.8	0.6	0.555	0.443	0.346	0.2988	0.293	0.1591
GraRep	1	0.2	0.04	0.035	0.033	0.038	0.035	0.035	0.019
Common Neighbor	1	1	1	0.96	0.9667	0.98	0.8775	0.798	0.192
LE	1	1	0.93	0.855	0.827	0.66	0.468	0.391	0.05

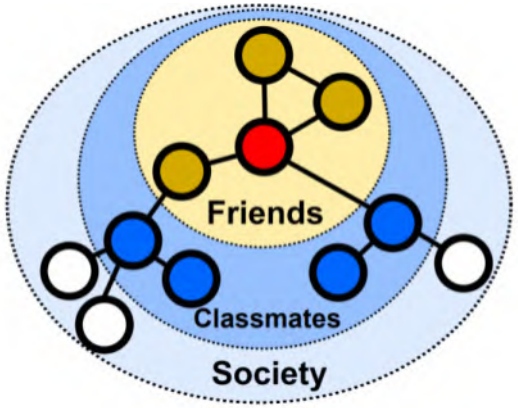
Significantly outperforms Line at the: ** 0.01 and * 0.05 level, paired t-test.

Improve at least 15%

Daixin Wang, Peng Cui, Wenwu Zhu. Structural Deep Network Embedding. *KDD*, 2016.

What is the *right* order?

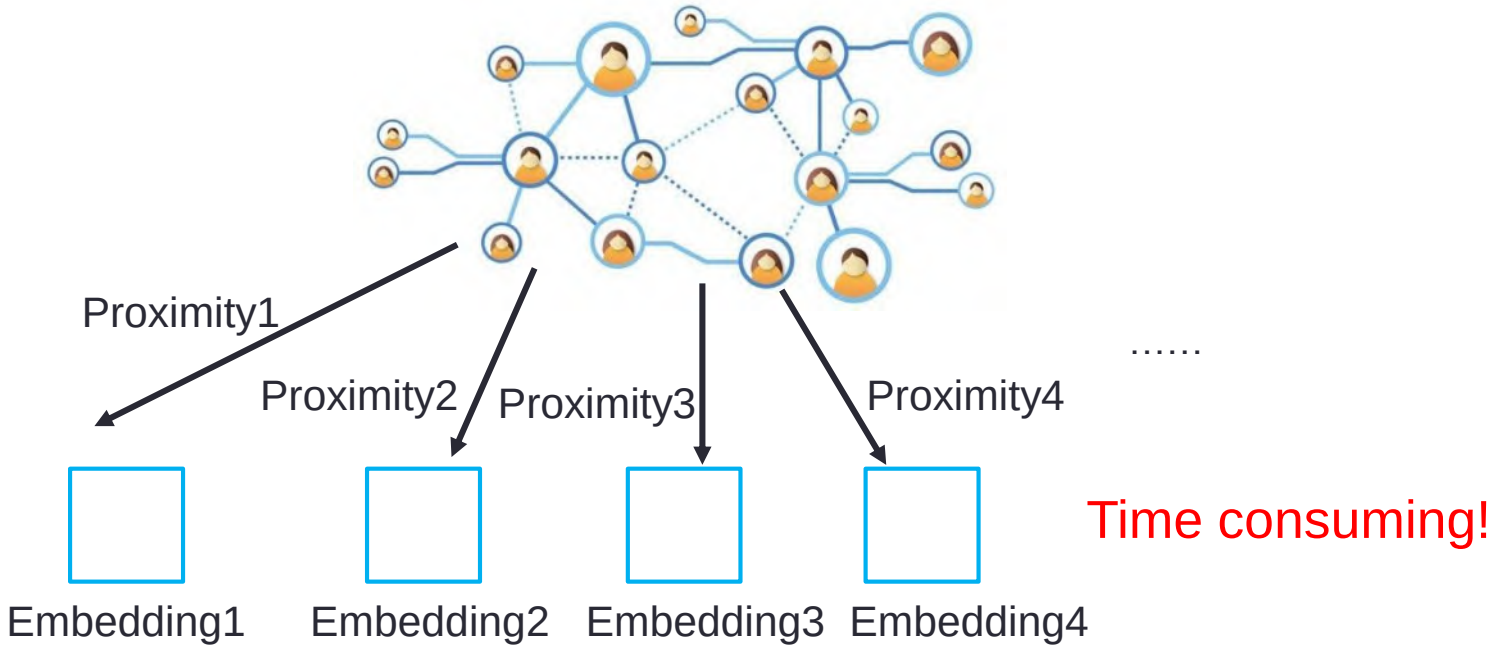
- Different networks/tasks require different high-order proximities
  - E.g., multi-scale classification (Bryan Perozzi, et al, 2017)



  - E.g., networks with different scales and sparsity
- Proximities of different orders can also be arbitrarily weighted
  - E.g., equal weights, exponentially decayed weights (Katz)

What is the *right* order?

- Existing methods can only preserve one fixed high-order proximity
  - Different high-order proximities are calculated separately



How to preserve arbitrary-order proximity while guaranteeing accuracy and efficiency?

→ What is the underlying relationship between different proximities?

Problem Formulation

- High-order proximity: a polynomial function of the adjacency matrix

$$S = f(A) = w_1 A^1 + w_2 A^2 + \cdots + w_q A^q$$

  - q: order; $w_1 \ldots w_q$: weights, assuming to be non-negative
  - A: could be replaced by other variations (such as the Laplacian matrix)
- Objective function: matrix factorization

$$\min_{U^*, V^*} \left\| S - U^* {V^*}^T \right\|_F^2$$

  - $U^*, V^* \in \mathbb{R}^{N \times d}$: left/right embedding vectors
  - d: dimensionality of the space
- Optimal solution: Singular Value Decomposition (SVD)
  - $[U, \Sigma, V]$: top-d SVD results

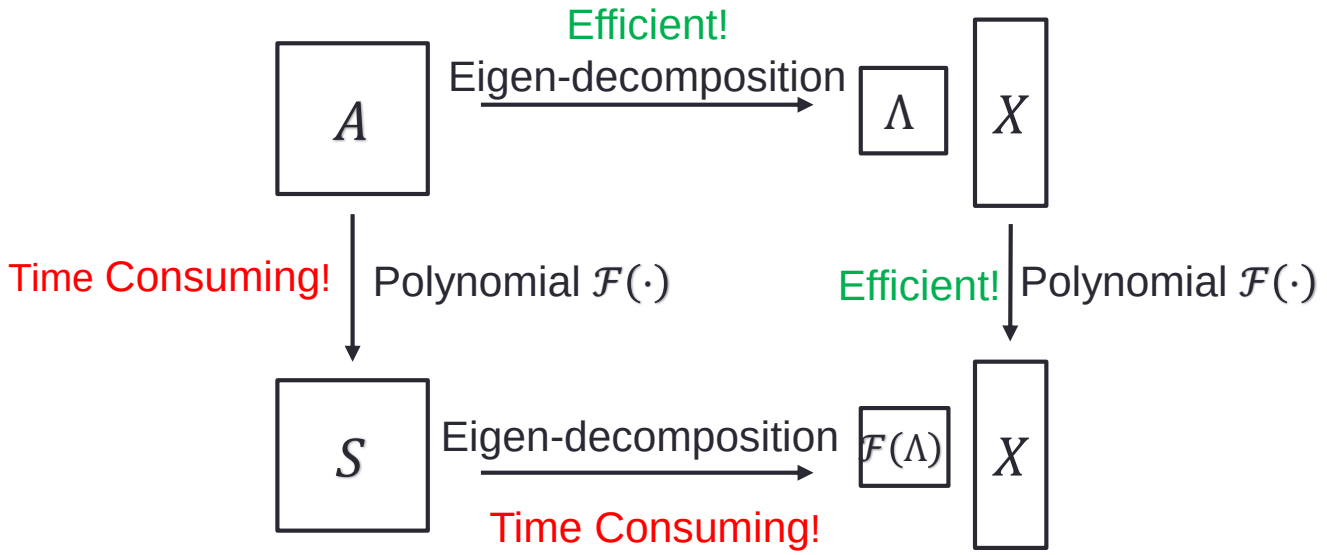
$$U^* = U\sqrt{\Sigma}, \qquad V^* = V\sqrt{\Sigma}$$

Ziwei Zhang, et al. Arbitrary-Order Proximity Preserved Network Embedding. *KDD, 2018.*

Eigen-decomposition Reweighting

- Eigen-decomposition reweighting

THEOREM 4.2 (EIGEN-DECOMPOSITION REWEIGHTING). *If* $[\lambda, \mathbf{x}]$ *is an eigen-pair of* $\mathbf{A}$, *then* $[\mathcal{F}(\lambda), \mathbf{x}]$ *is an eigen-pair of* $\mathbf{S} = \mathcal{F}(\mathbf{A})$.



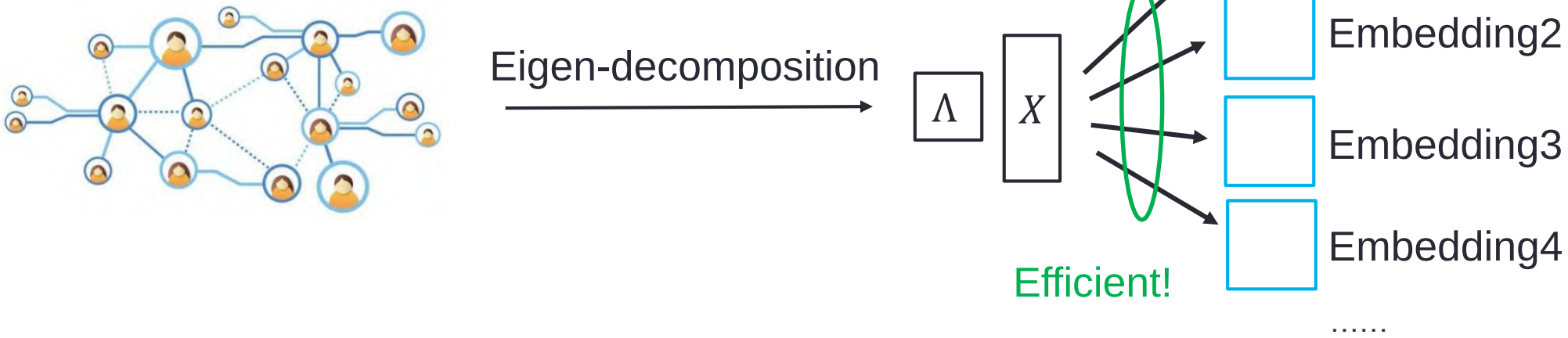
- **Insights**: high-order proximity is simply re-weighting dimensions!

$$U^* = U\sqrt{\Sigma}, V^* = V\sqrt{\Sigma}$$

Ziwei Zhang, et al. Arbitrary-Order Proximity Preserved Network Embedding. *KDD, 2018.*

Preserving Arbitrary-Order Proximity

- Shifting across different orders/weights:

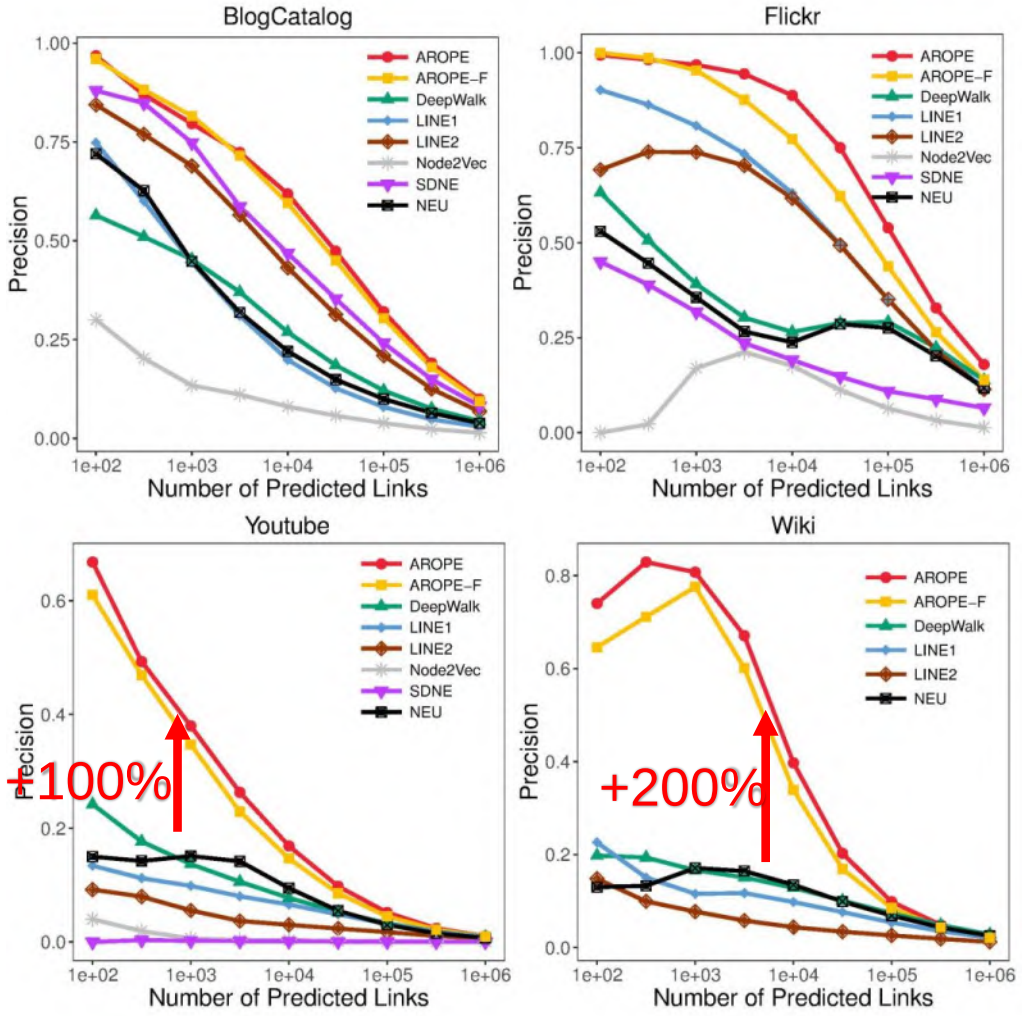


- Preserving arbitrary-order proximity
- Low marginal cost
- Accurate and efficient

Ziwei Zhang, et al. Arbitrary-Order Proximity Preserved Network Embedding. *KDD, 2018.*

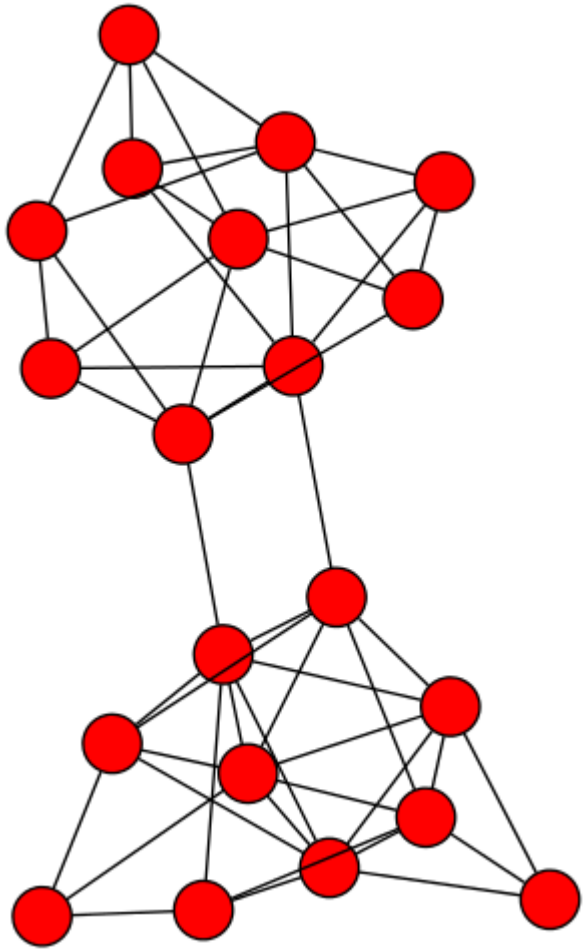
Experimental Results

- Link Prediction



Ziwei Zhang, et al. Arbitrary-Order Proximity Preserved Network Embedding. *KDD, 2018.*

Section Summary



Nodes & Links

↓

Pair-wise Proximity

↓

Community Structures

↓

Hyper Edges

↓

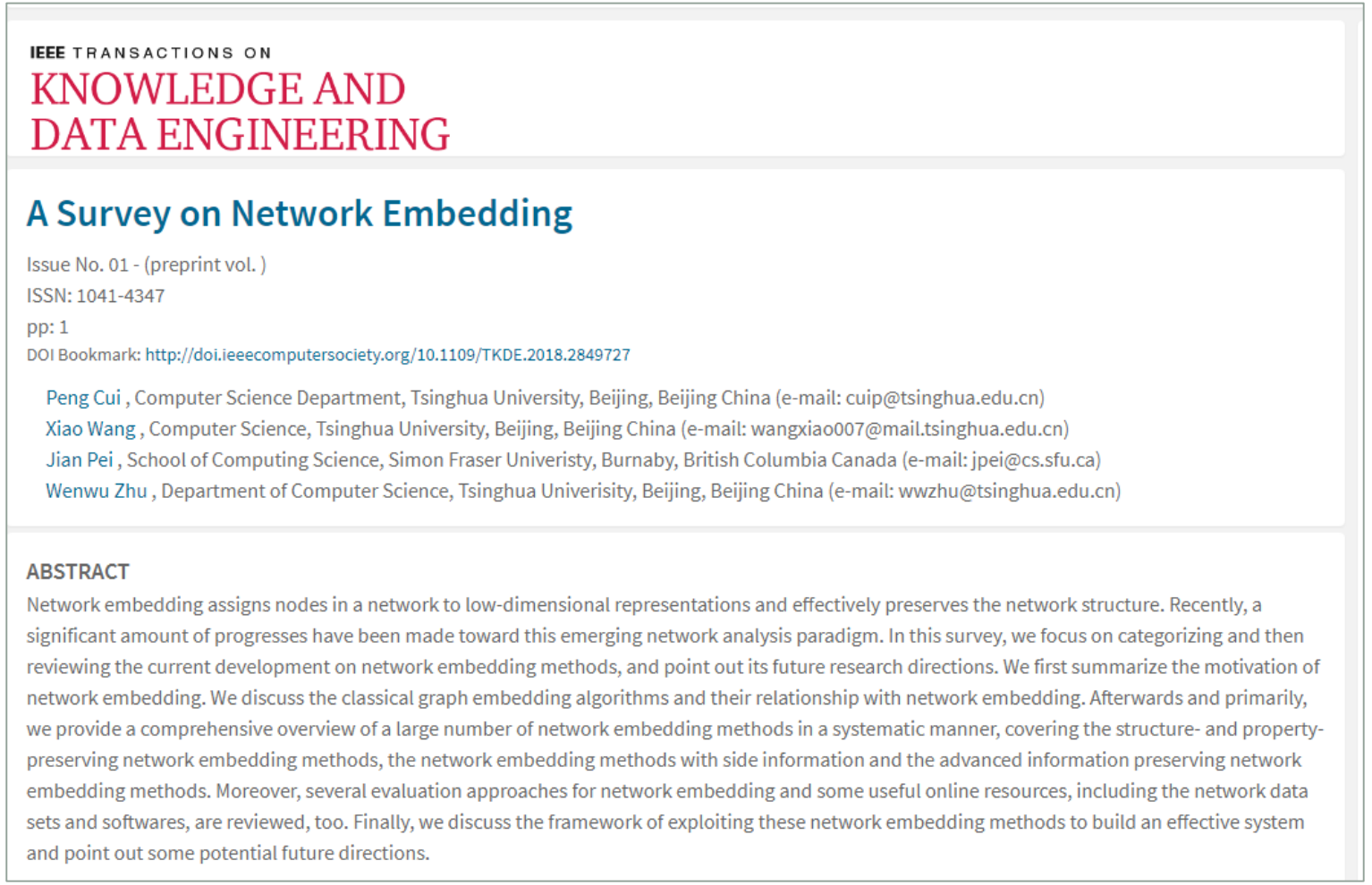
Global Structure

Network Characteristics

Application Characteristics

Dedicated Network Embedding Tutorial at KDD 2019:
http://cuip.thumedialab.com/papers/KDD19%20Tutorial%20on%20NE_Peng.pdf

A Survey on Network Embedding



IEEE TRANSACTIONS ON
KNOWLEDGE AND
DATA ENGINEERING

A Survey on Network Embedding

Issue No. 01 - (preprint vol.)
ISSN: 1041-4347
pp: 1
DOI Bookmark: http://doi.ieeecomputersociety.org/10.1109/TKDE.2018.2849727

Peng Cui , Computer Science Department, Tsinghua University, Beijing, Beijing China (e-mail: cuip@tsinghua.edu.cn)
Xiao Wang , Computer Science, Tsinghua University, Beijing, Beijing China (e-mail: wangxiao007@mail.tsinghua.edu.cn)
Jian Pei , School of Computing Science, Simon Fraser Univeristy, Burnaby, British Columbia Canada (e-mail: jpei@cs.sfu.ca)
Wenwu Zhu , Department of Computer Science, Tsinghua Univerisity, Beijing, Beijing China (e-mail: wwzhu@tsinghua.edu.cn)

ABSTRACT

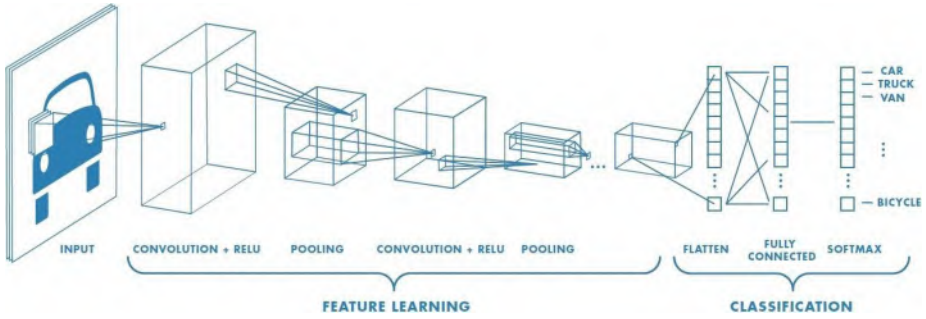
Network embedding assigns nodes in a network to low-dimensional representations and effectively preserves the network structure. Recently, a significant amount of progresses have been made toward this emerging network analysis paradigm. In this survey, we focus on categorizing and then reviewing the current development on network embedding methods, and point out its future research directions. We first summarize the motivation of network embedding. We discuss the classical graph embedding algorithms and their relationship with network embedding. Afterwards and primarily, we provide a comprehensive overview of a large number of network embedding methods in a systematic manner, covering the structure- and property-preserving network embedding methods, the network embedding methods with side information and the advanced information preserving network embedding methods. Moreover, several evaluation approaches for network embedding and some useful online resources, including the network data sets and softwares, are reviewed, too. Finally, we discuss the framework of exploiting these network embedding methods to build an effective system and point out some potential future directions.

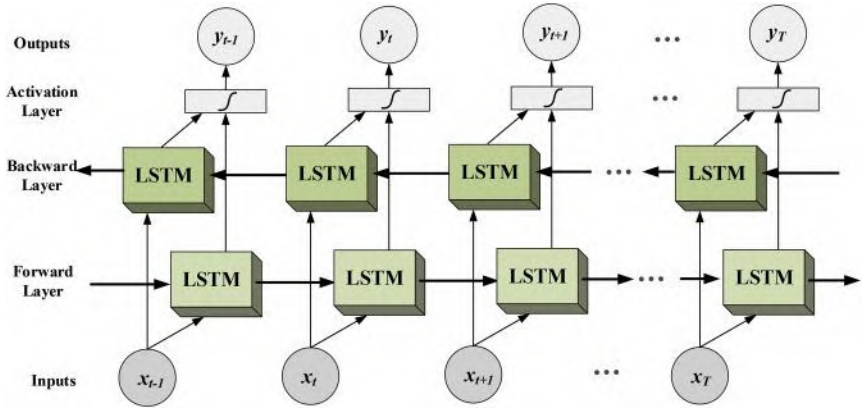
Peng Cui, Xiao Wang, Jian Pei, Wenwu Zhu. **A Survey on Network Embedding**. *IEEE TKDE, 2018*.

Learning from networks

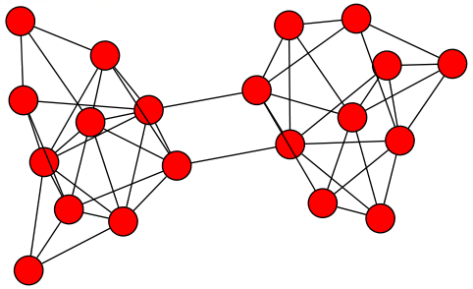
GNN

Graph Neural Networks





$G = (V, E)$



?

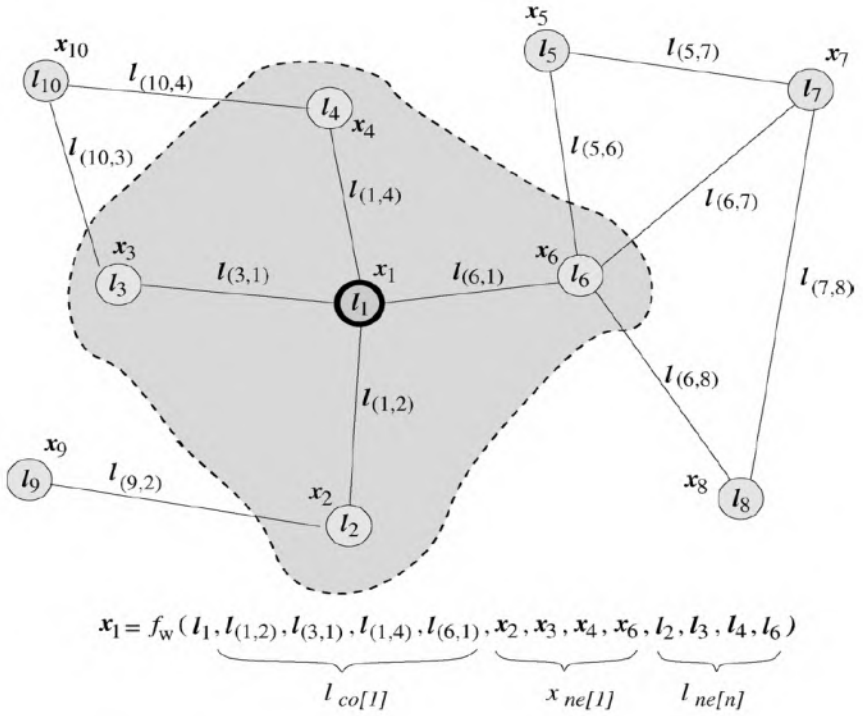
Can we design a learning mechanism to directly work on graphs?

The First Graph Neural Network

- Basic idea: a recursive definition of states

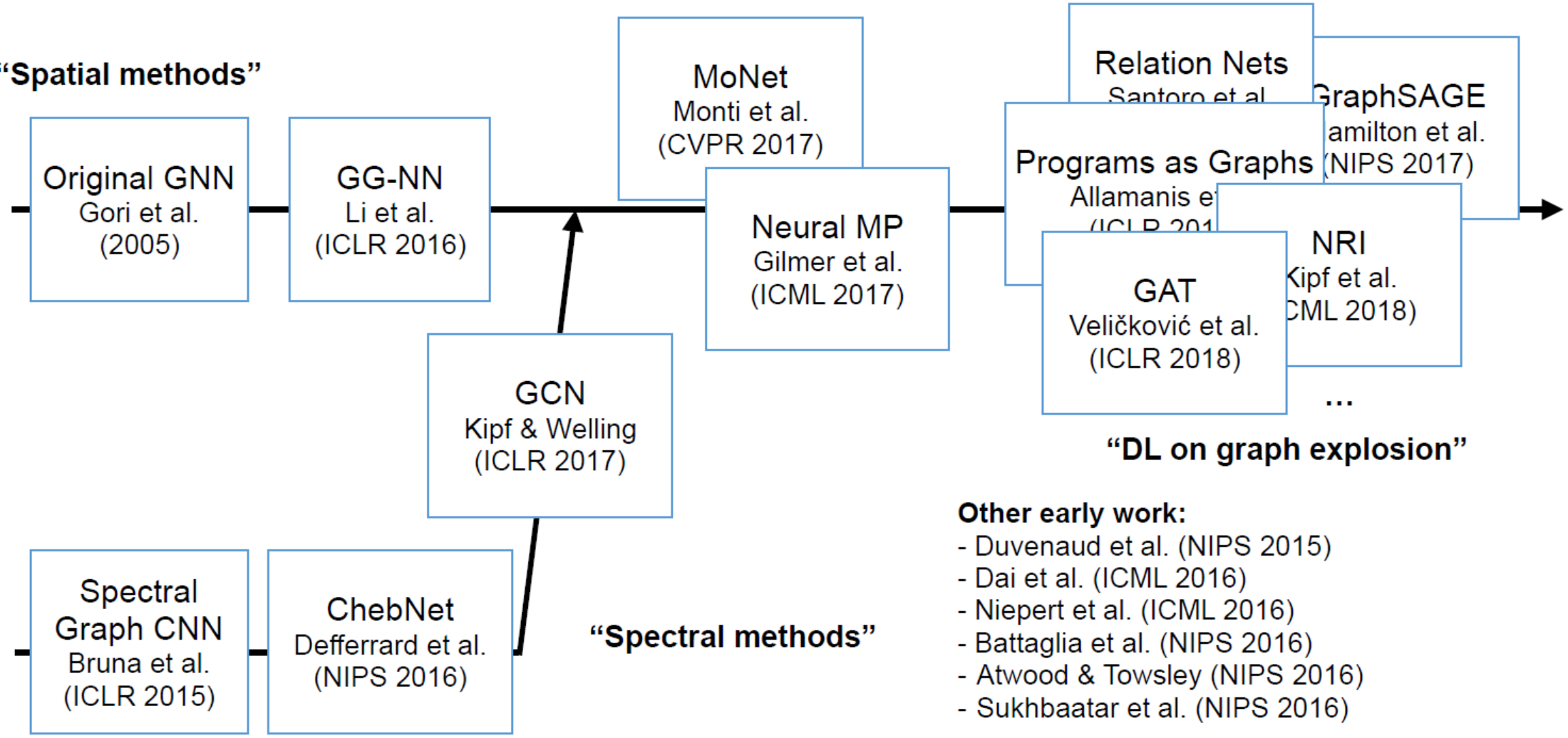
$$\mathbf{s}_i = \sum_{j \in \mathcal{N}(i)} \mathcal{F}\left(\mathbf{s}_i, \mathbf{s}_j, \mathbf{F}_i^V, \mathbf{F}_j^V, \mathbf{F}_{i,j}^E\right)$$

- A simple example: PageRank: $s_i = \alpha \sum_{j \in \mathcal{N}(i)} \frac{s_j}{d_j} + \frac{(1-\alpha)}{n}$



F. Scarselli, et al. The graph neural network model. *IEEE TNN, 2009.*

Many GNNs have emerged since then

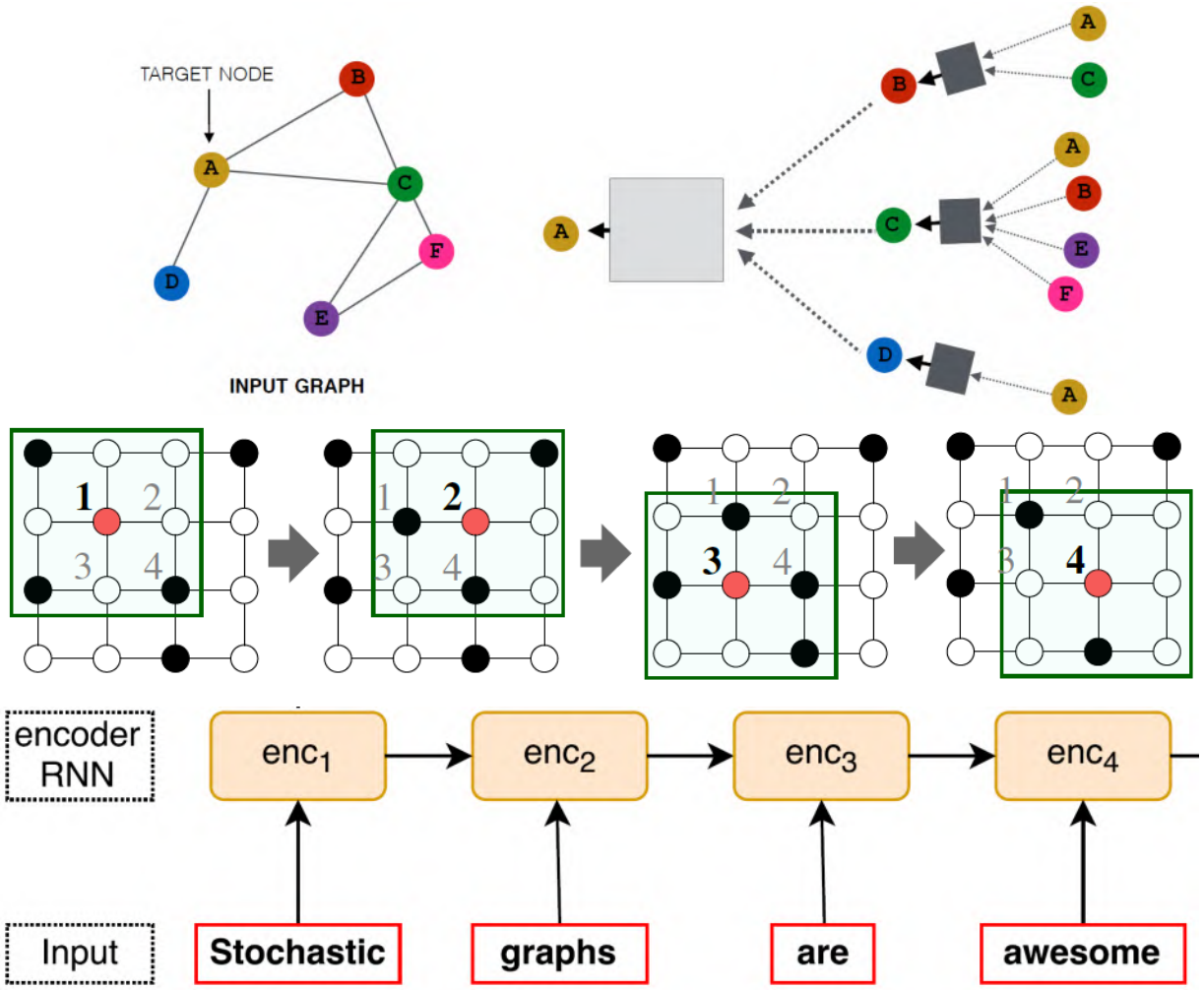


Other early work:
- Duvenaud et al. (NIPS 2015)
- Dai et al. (ICML 2016)
- Niepert et al. (ICML 2016)
- Battaglia et al. (NIPS 2016)
- Atwood & Towsley (NIPS 2016)
- Sukhbaatar et al. (NIPS 2016)

(slide inspired by Alexander Gaunt's talk on GNNs)

Picture credit to Thomas Kipf

How are GNNs compared with other NNs?



- Capture information from graph neighborhoods
- Capture information from nearby grids (i.e., a 2-D graph)
- Capture information from contexts (i.e., a 1-D graph)

We need to exchange information within neighborhoods

Picture credit to GraphSAGE (NIPS 17), PATCHY-SAN (ICML 16)

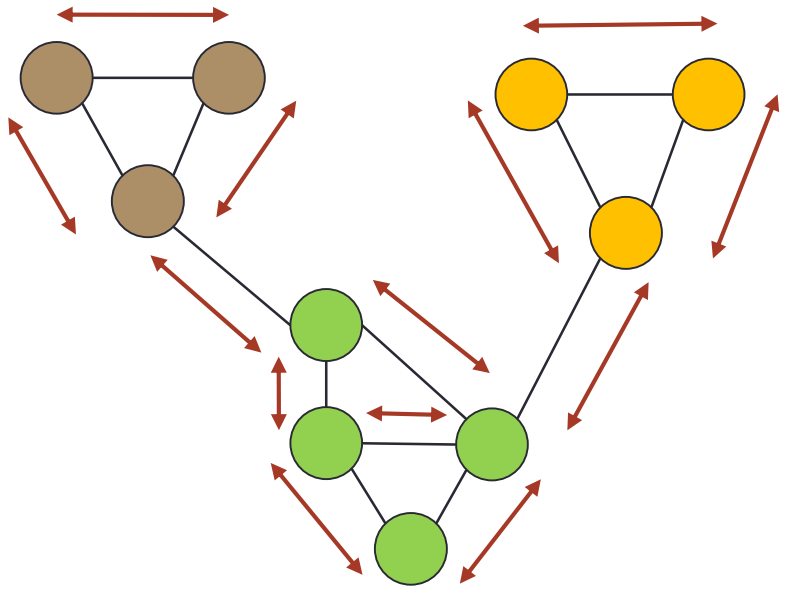
Message-passing Framework

- Formulation:

$$\mathbf{m}_i^{(l)} = \text{AGG}(\{\mathbf{h}_j^{(l)}, \forall j \in \tilde{\mathcal{N}}_i\})$$

$$\mathbf{h}_i^{(l+1)} = \text{UPDATE}([\mathbf{h}_i^{(l)}, \mathbf{m}_i^{(l)}])$$

  - $\mathrm{h}_i^{(l)}$: representation of node v_i in the l^{th} layer
  - $\mathrm{m}_i^{(l)}$: messages for node v_i in the l^{th} layer by aggregating neighbor representations



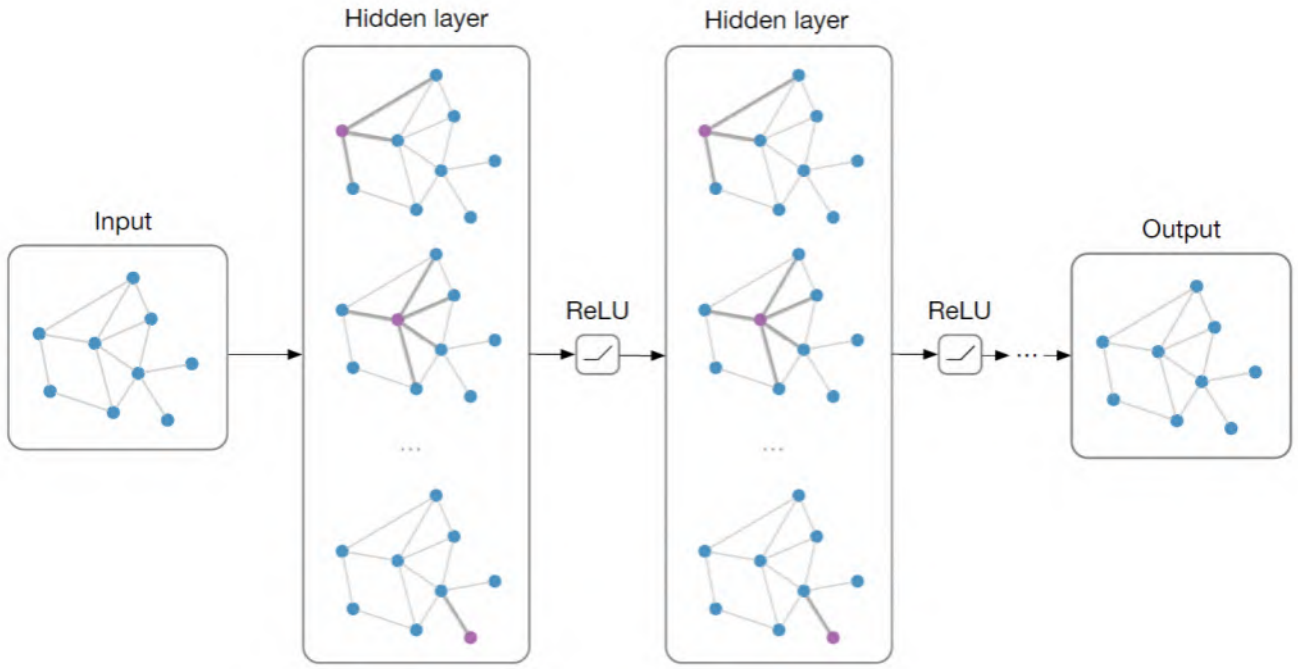
J. Gilmer, et al. Neural message passing for quantum chemistry. *ICML, 2017.*
W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. *NIPS, 2017.*

Graph Convolutional Networks (GCN)

- Main idea: averaging messages from direct neighborhoods

$$\mathbf{H}^{l+1} = \rho\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{l}\mathbf{\Theta}^{l}\right)$$

- Stacking multiple layers like standard CNNs:
  - State-of-the-art results on node classification

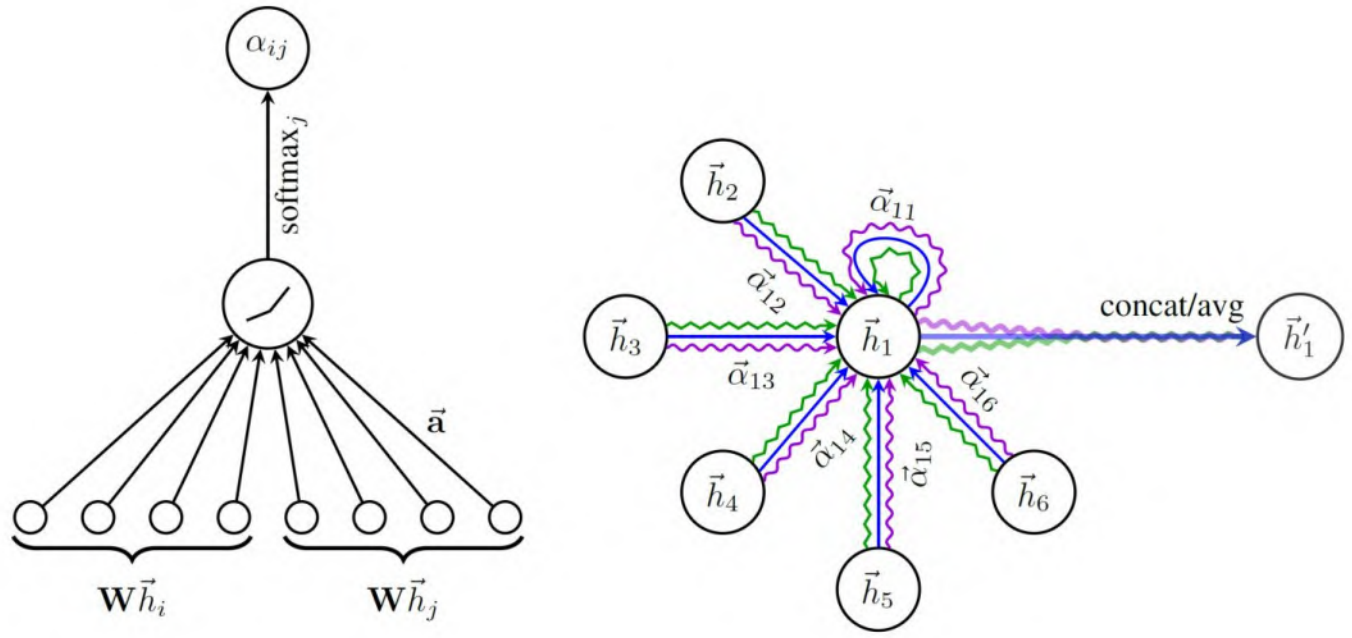


T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *ICLR, 2017.*

Graph Attention Network (GAT)

- Main idea: weight messages using attention mechanism

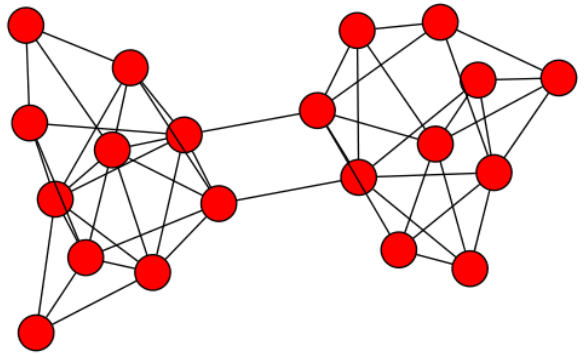
$$h_i^{l+1} = \rho\left(\sum_{j \in N(i)} \alpha_{ij}^l h_j^l\right)$$

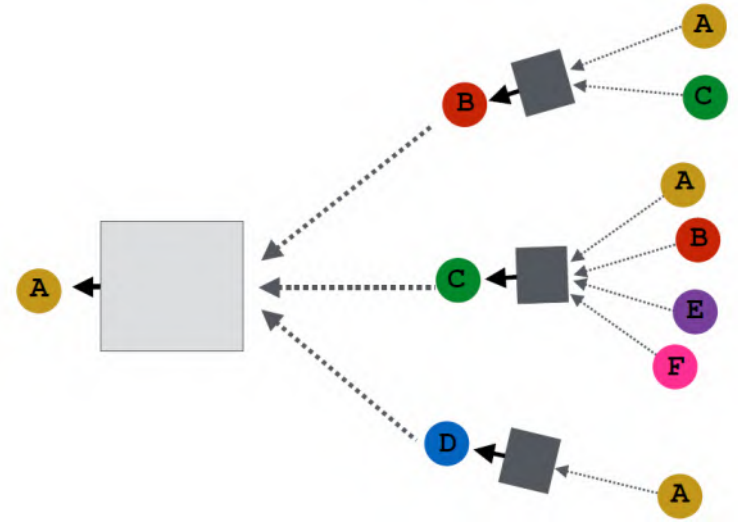


Graph Attention Networks. *ICLR, 2018.*

Expected Properties of GNNs

$G = (V, E)$





- Some expected properties of GNNs:
  - Trained end-to-end for downstream tasks
    - Vs. network embedding: unsupervised representation learning to handle various tasks
  - Utilize node features and graph structures simultaneously
  - Can handle real applications with data represented as graphs

Are existing GNNs good enough?

Outline

- Does GNN fuse *feature* and *topology* optimally?
- Technical challenges in real applications: robustness

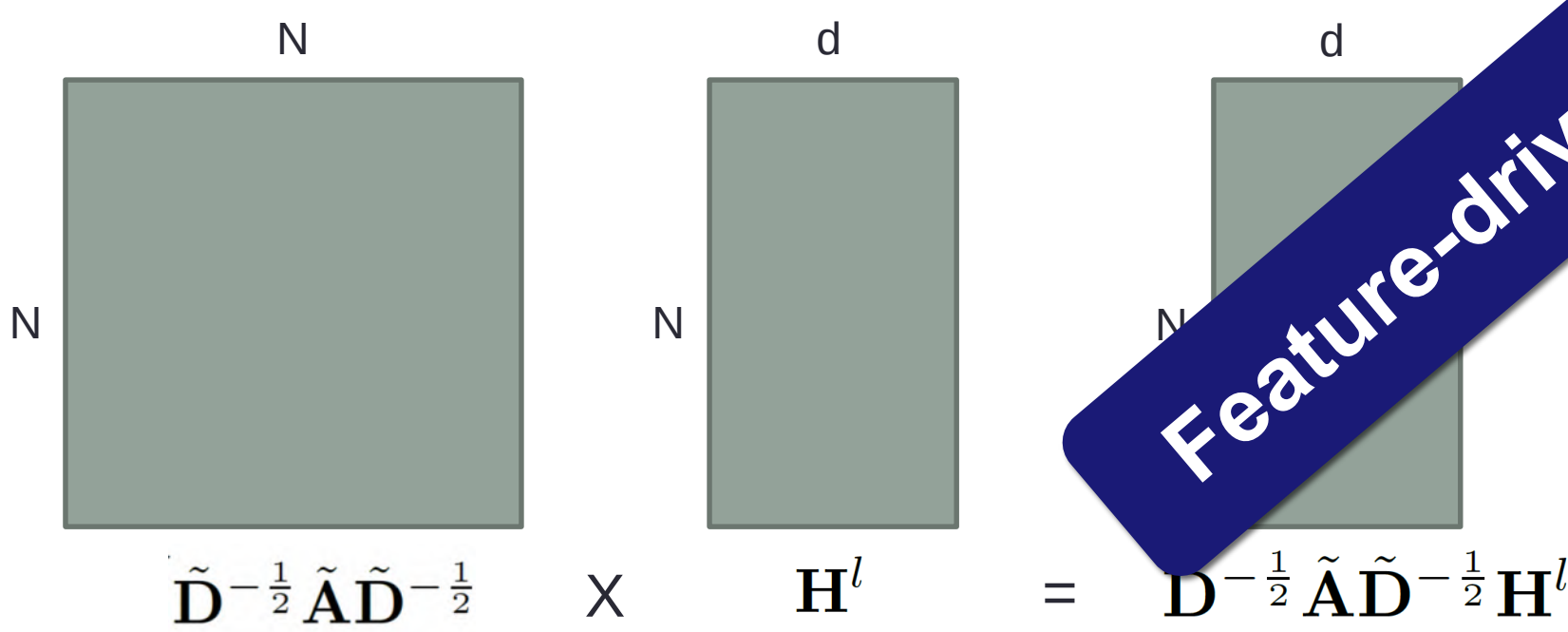
Outline

- Does GNN fuse *feature* and *topology* optimally?
- Technical challenges in real applications: robustness

The intrinsic problem GCN is solving

$$\mathbf{H}^{l+1} = \rho\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{l}\mathbf{\Theta}^{l}\right)$$

Fusing topology and features in the way of smoothing features with the assistance of topology.



Revisiting Graph Neural Networks All We Have is Low-Pass Filters, *arXiv 1905.09550*
Understanding the Representation Power of Graph Neural Networks in Learning Graph Topology, *NeurIPS 2019*

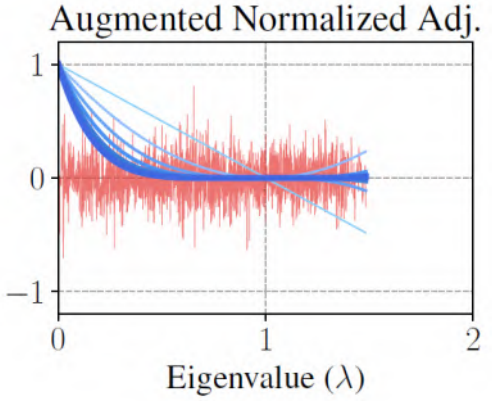
The intrinsic problem GCN is solving

- Theoretical analysis: node features as “true signal”, GNNs as a low-pass filtering
  - Simplified GCN[1]: removing all non-linearity

$$H^{(l)} = S^l H^{(0)}$$

$$S = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2} = I - \tilde{L}_{Sym}$$

  - From graph signal processing, $f' = S^l f$ corresponds to a spectral filter [2]



$$g(\lambda) = \left(1 - \hat{\lambda}\right)^l$$

- GCNs are a special form of Laplacian smoothing for node features[3]

1. Simplifying Graph Convolutional Networks, *ICML 2019*
2. Revisiting Graph Neural Networks All We Have is Low-Pass Filters, *arXiv 1905.09550*
3. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning, AAAI

Can GNNs Fully Preserve Graph Structures?

- When feature plays the key role, GNN performs good
- How about the contrary?
- Synthesis data: stochastic block model + random features
  - DeepWalk greatly outperforms all the GCNs
  - Recall the message-passing framework

$$\mathbf{m}_i^{(l)} = \text{AGG}(\{\mathbf{h}_j^{(l)}, \forall j \in \tilde{\mathcal{N}}_i\})$$
$$\mathbf{h}_i^{(l+1)} = \text{UPDATE}([\mathbf{h}_i^{(l)}, \mathbf{m}_i^{(l)}])$$

Initialized as node features

Graph structures only provide neighborhoods in aggregation

- Initial node features provide important inductive bias!

Method	Results
Random	10.0
GCN-1	29.7
GCN-2	48.4
GCN-3	56.2
GCN-5	53.3
DeepWalk	99.9

GCN-X: X number of layers

A New Perspective to Understand GNNs

- A new perspective: treating GNNs as a type of (non-linear) dimensionality reduction
  - A slightly modified framework:

$$\mathbf{H}^{(l+1)} = \sigma\left(\mathcal{F}(\mathbf{A})\,\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right)$$

Non-linear mapping

Graph structures

Previous bases

Linear transform

Method	$\mathcal{F}(\mathbf{A})$
GCN, SGC	$\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$
DCNN	$\left(\mathbf{D}^{-1}\mathbf{A}\right)^{K}$
DGCN	$\mathbf{D}_P^{-\frac{1}{2}}\mathbf{A}_P\mathbf{D}_P^{-\frac{1}{2}}$
PPNP	$\alpha\left(\mathbf{I}_N-(1-\alpha)\,\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\right)^{-1}$

  - Three-steps

(1) Projecting graph structures into a subspace spanned by node representations in the last step

(2) The projected representations are linearly transformed followed by a non-linear mapping

(3) Repeat the process by using the new node representations as bases

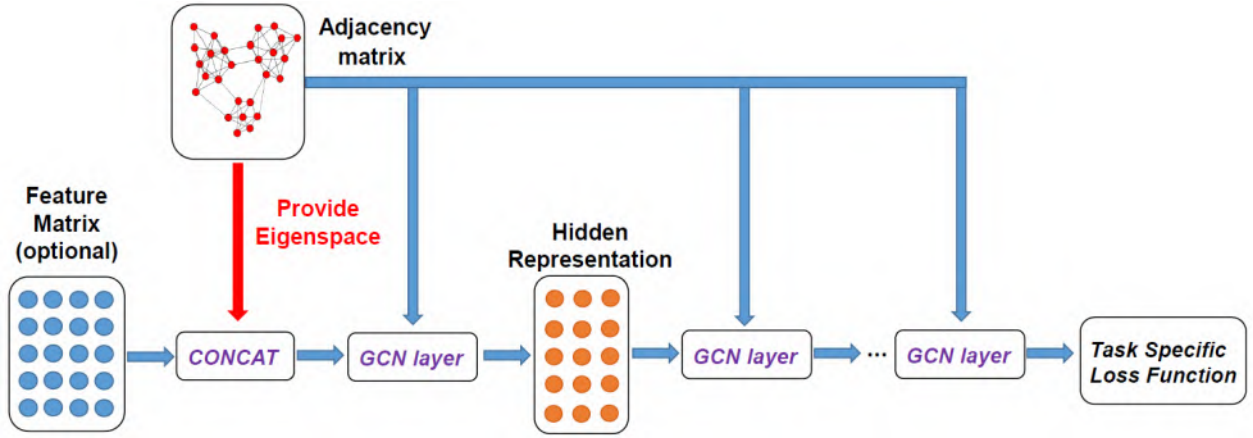
- Why are the existing GNNs feature-centric?

→ The initial space is solely determined by node features!

Ziwei Zhang, Peng Cui, Jian Pei, Xin Wang, Wenwu Zhu. Eigen-GNN: A Graph Structure Preserving Plug-in for GNNs. *arXiv, 2006.04330*.

Eigen-GNN: A Graph Structure Preserving Plug-in

- Framework



$$\mathbf{H}^{(0)} = [\mathbf{X}, f(\mathbf{Q})]$$

Union space of X and $f(Q)$

X: node features; Q: top-d eigenvector of a graph structure matrix; $f(\cdot)$: a simple function such as normalization

- Experimental results:

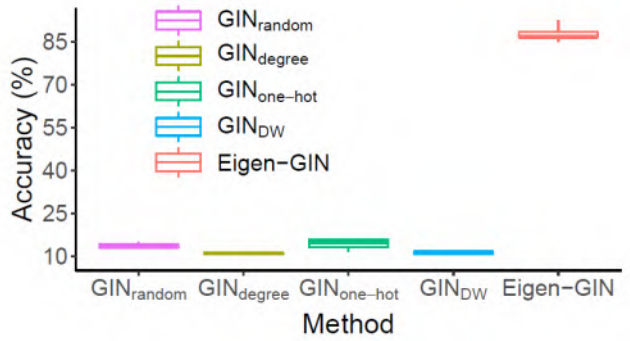
Node classification

Data	Method	Harvard	Columbia	Stanford
A,Y	GCN_{random}	74.6±0.5	63.6±1.6	68.2±1.5
	GCN_{degree}	74.4±2.0	63.8±2.3	67.8±1.6
	GCN_{DW}	82.5±1.0	**76.0 ± 1.3**	76.6±1.3
	Eigen-GCN_{struc}	**82.7 ± 1.2**	**76.0 ± 1.9**	**78.9 ± 1.3**
A,X,Y	GCN_{feat}	70.6±1.3	74.8±1.7	71.3±1.6
	$GCN_{feat+DW}$	83.1±0.7	77.6±1.3	78.3±1.4
	Eigen-$GCN_{feat+struc}$	**84.6 ± 1.4**	**78.6 ± 1.1**	**79.7 ± 1.2**

Link Prediction

Dataset	C.elegans	E.coli	Power
SEAL	77.6±0.9	91.5±0.8	69.9±1.6
$SEAL_{DW}$	77.1±1.5	92.1±0.7	69.8±1.5
Eigen-SEAL	**79.5 ± 0.8***	**92.5 ± 0.6***	**73.2 ± 2.4***
Gain†	+1.9	+0.4	+3.3

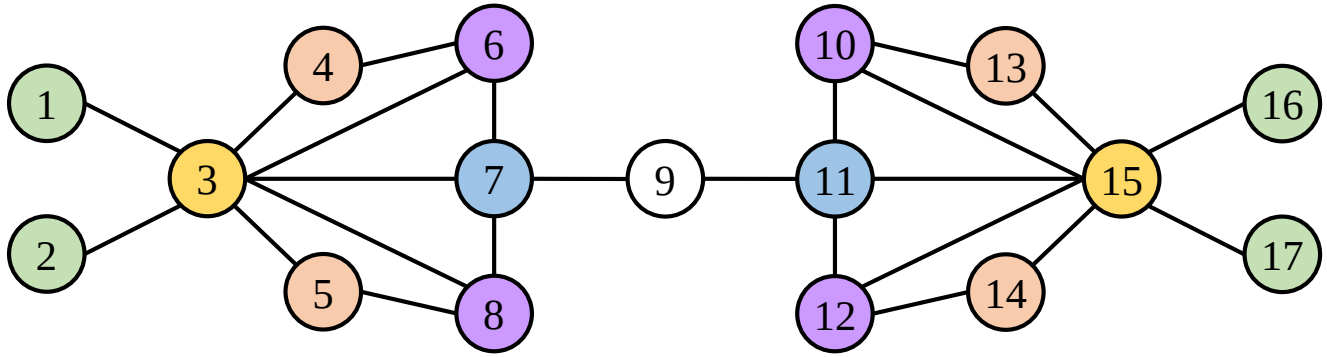
Graph Isomorphism Test



Ziwei Zhang, Peng Cui, Jian Pei, Xin Wang, Wenwu Zhu. Eigen-GNN: A Graph Structure Preserving Plug-in for GNNs. *arXiv, 2006.04330*.

Permutation-equivariance of GNNs

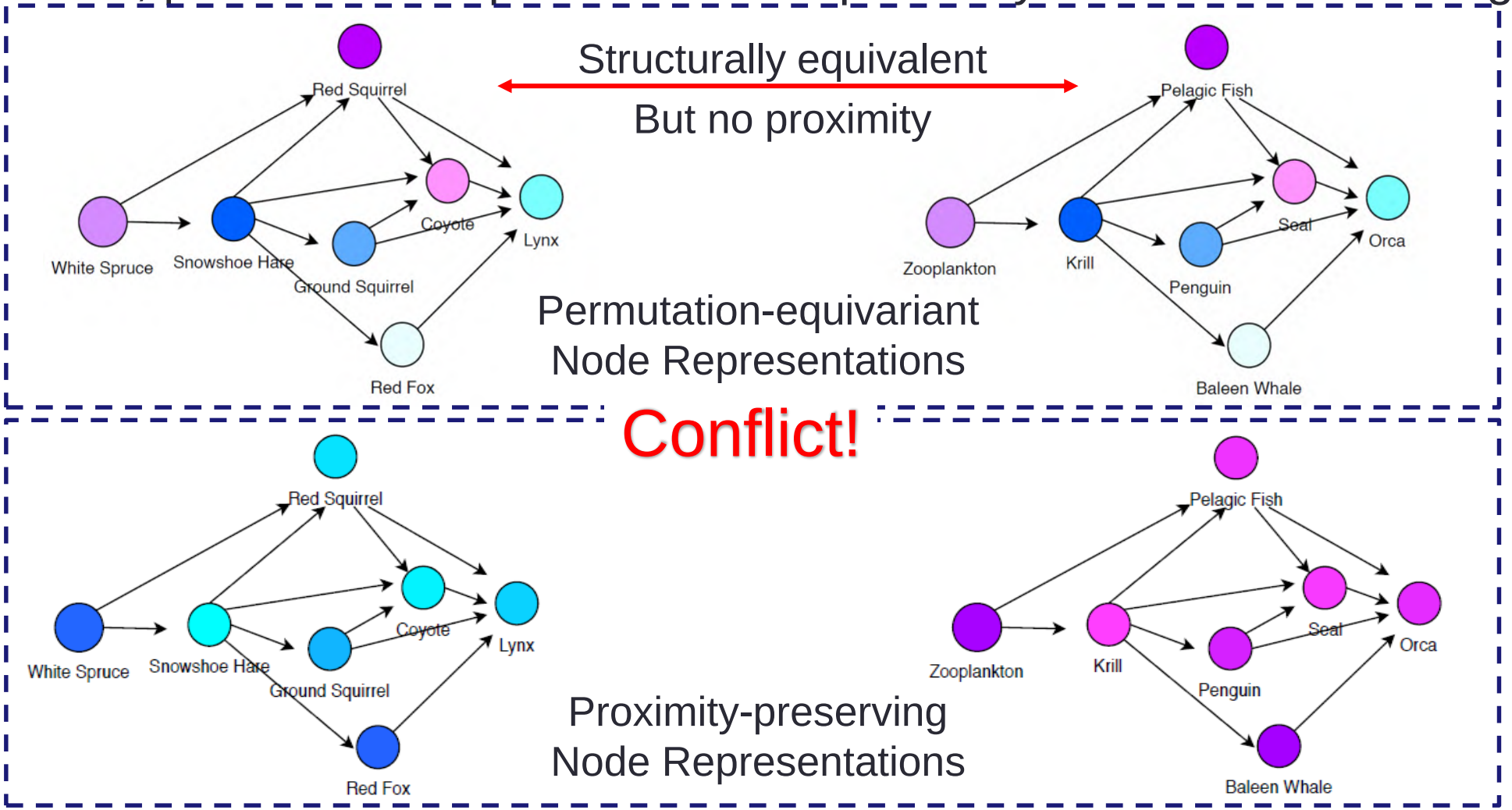
- Permutation-equivariance property
  - If we randomly permute the IDs of nodes while maintaining the graph structure, the representations of nodes in GNNs should be permuted accordingly
- Pros:
  - Guarantees that the representations of automorphic nodes are identical
  - Automatically generalize to all the $O(N!)$ permutations when training with only one permutation
- Most of the existing message-passing GNNs satisfy permutation-equivariance



Permutation-equivariant Node Embeddings

Permutation-equivariance vs. Proximity-aware

- However, permutation-equivariance and proximity-aware are conflicting

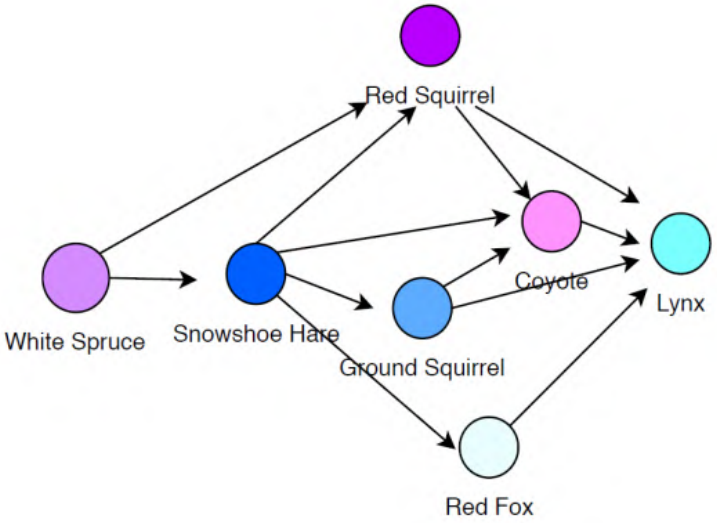


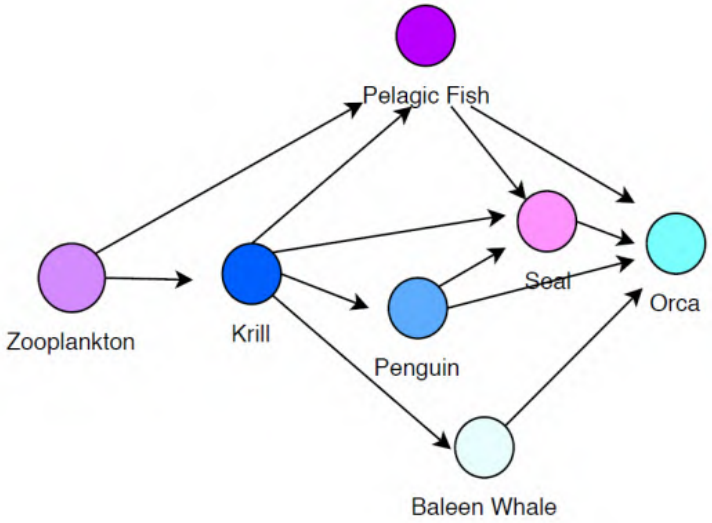
Position-aware graph neural networks. *ICML 2019.*

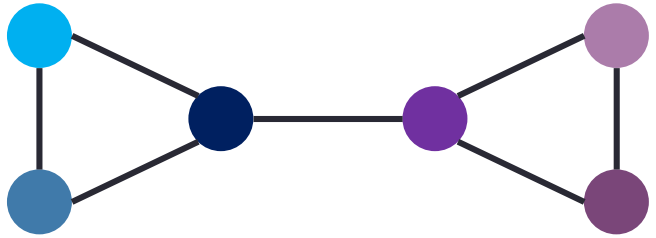
On the Equivalence between Positional Node Embeddings and Structural Graph Representations. *ICLR 2020.*

Unique Node Identifiers

- The key problem is we can only differentiate nodes with unique identifiers





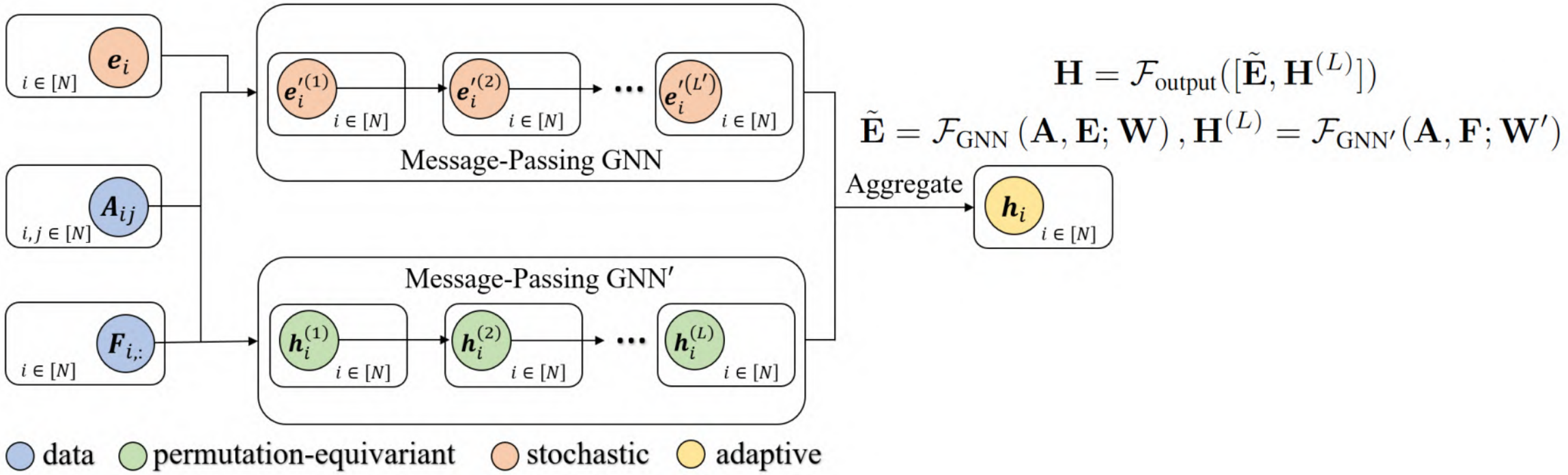


- Theoretical analysis: unique node identifiers are one necessary condition for GNNs to be universal approximation

What graph neural networks cannot learn depth vs width, *ICLR 2020*

Stochastic Message Passing (SMP)

- Assign stochastic features as node identifiers
  - Gaussian features: associate with random projection literature
- A dual GNN architecture:



Ziwei Zhang, Chenhao Niu, Peng Cui, et al. A Simple and General Graph Neural Network with Stochastic Message Passing. *arXiv, 2009.02562.*

Theoretical Guarantee

- SMP can preserve node proximities

Theorem 2. *An SMP in Eq. (9) with the message-passing matrix $\tilde{\mathbf{A}}$ and the number of propagation steps K can preserve the walk-based proximity $\tilde{\mathbf{A}}^K(\tilde{\mathbf{A}}^K)^T$ with high probability if the dimensionality of the stochastic matrix d is sufficiently large, where the superscript T denotes matrix transpose. The theorem is regardless of whether $\mathbf{E}$ are fixed or resampled.*

- SMP can recover the existing permutation-equivariant GNNs

Corollary 2. *For any task, Eq. (8) with the aforementioned linear $\mathcal{F}_{output}(\cdot)$ is at least as powerful as the permutation-equivariant $\mathcal{F}_{GNN'}(\mathbf{A}, \mathbf{F}; \mathbf{W}')$, i.e., the minimum training loss of using $\mathbf{H}$ in Eq. (8) is equal to or smaller than using $\mathbf{H}^{(L)} = \mathcal{F}_{GNN'}(\mathbf{A}, \mathbf{F}; \mathbf{W}')$.*

- An adaptive GNN that maintains both proximity-awareness and permutation-equivariance

Ziwei Zhang, Chenhao Niu, Peng Cui, et al. A Simple and General Graph Neural Network with Stochastic Message Passing. *arXiv, 2009.02562.*

Experimental Results

Model	Node Labels		
	Community	Social Status	Both
Random	10.0	50.0	5.0
SGC	10.0±0.0	51.0±1.4	5.0±0.0
GCN	8.7 ±1.1	91.6±1.8	8.9±0.9
GAT	10.0±0.0	73.4±12.9	5.0±0.0
P-GNN	64.1±4.8	54.9±9.8	5.6±1.2
SMP-Linear	**98.8±0.6**	**93.9±0.9**	**93.8±1.6**

Fail

Fail

Fail

Table 4: The results of link prediction on the PPA dataset.

Model	Hits@100
SGC	0.1187±0.0012
GCN	0.1867±0.0132
GraphSAGE	0.1655±0.0240
P-GNN	Out of Memory
Node2vec	0.2226±0.0083
Matrix Factorization	0.3229±0.0094
SMP-Identity	0.2018±0.0148
SMP-Linear	**0.3582±0.0070**

+10%

Ziwei Zhang, Chenhao Niu, Peng Cui, et al. A Simple and General Graph Neural Network with Stochastic Message Passing. *arXiv, 2009.02562.*

Outline

- Does GNN fuse *feature* and *topology* optimally?
- Technical challenges in real applications: robustness

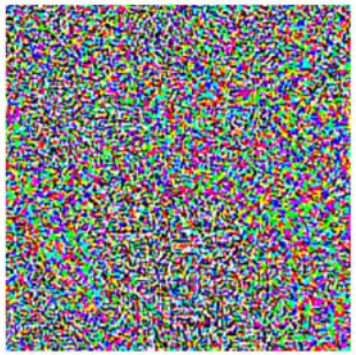
Technical challenges in real applications

Research → **Application**

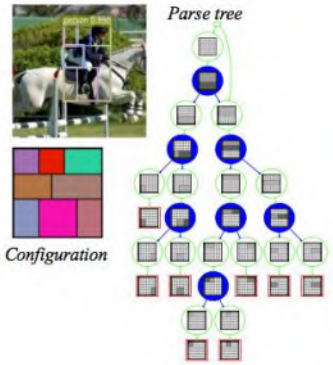
Robustness | Interpretability | Applicability

Hot directions in computer vision:

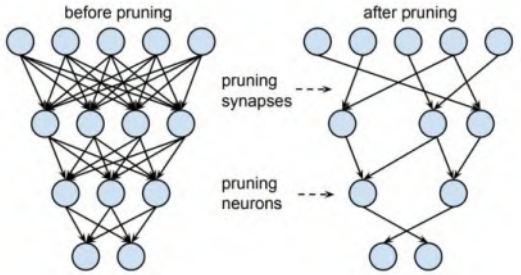
Adversarial



Explainable

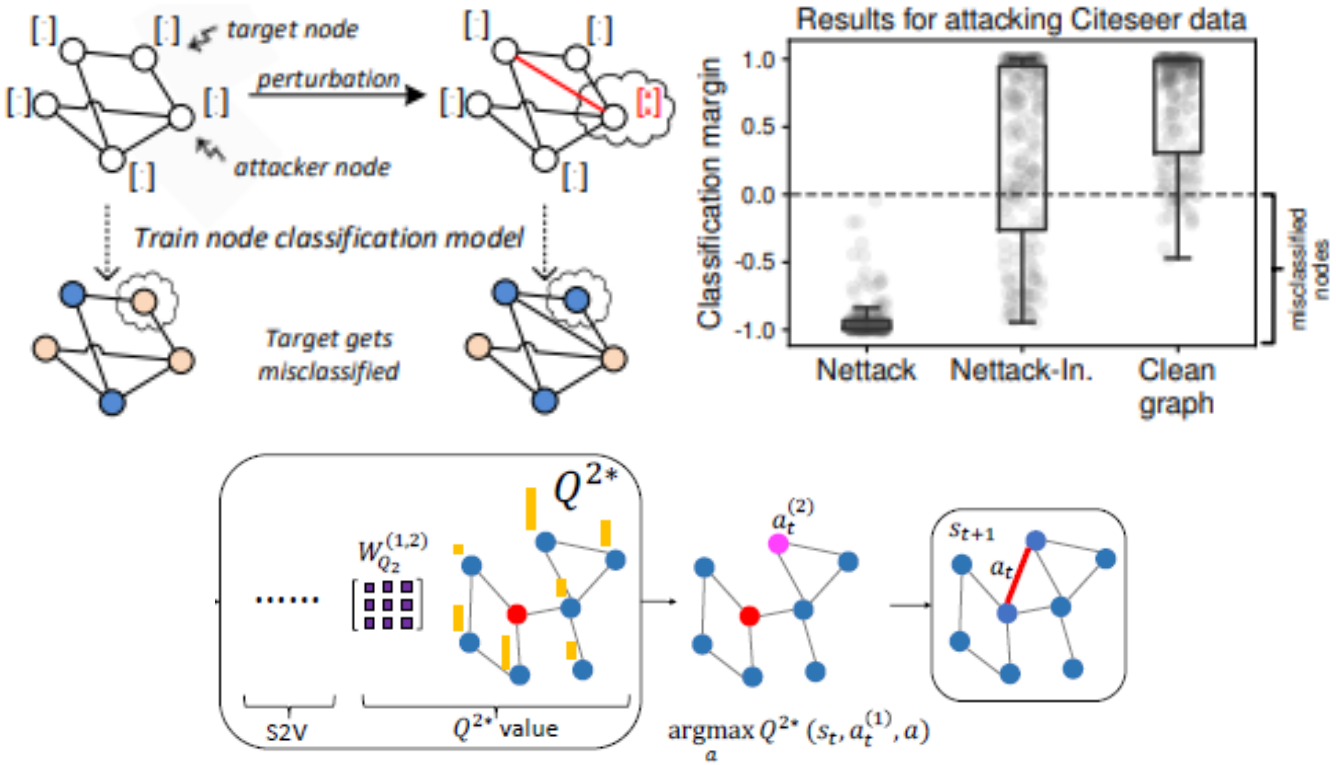


Scalable



Robustness in GNNs

- **Adversarial attacks**
  - Small perturbations in graph structures and node attributes lead to great changes
  - A serious concern for applying GNNs to real-world applications



Adversarial Attacks on Neural Networks for Graph Data, *KDD 2018*

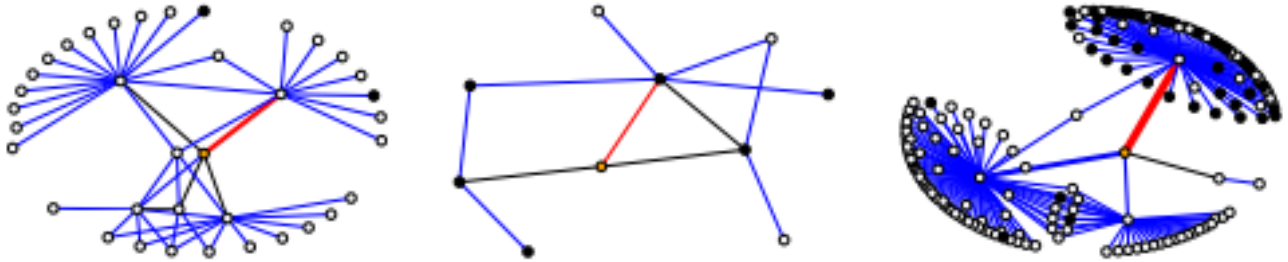
Adversarial Attacks on GNNs

- Categories
  - Targeted vs. Non-targeted
    - Targeted: the attacker focus on misclassifying some target nodes
    - Non-targeted: the attacker aims to reduce the overall model performance
  - Direct vs. Influence
    - Direct: the attacker can directly manipulate the edges/features of the target nodes
    - Influence: the attacker can only manipulate other nodes except the targets
  - Attacker knowledge:

Settings	Parameters	Predictions	Labels	Training Input
White-**B**ox **A**ttack (WBA)	✓	✓	✓	✓
Practical **W**hite-box **A**ttack (PWA)		✓	✓	✓
Restrict **B**lack-box **A**ttack (RBA)				✓

Robust Graph Convolutional Networks

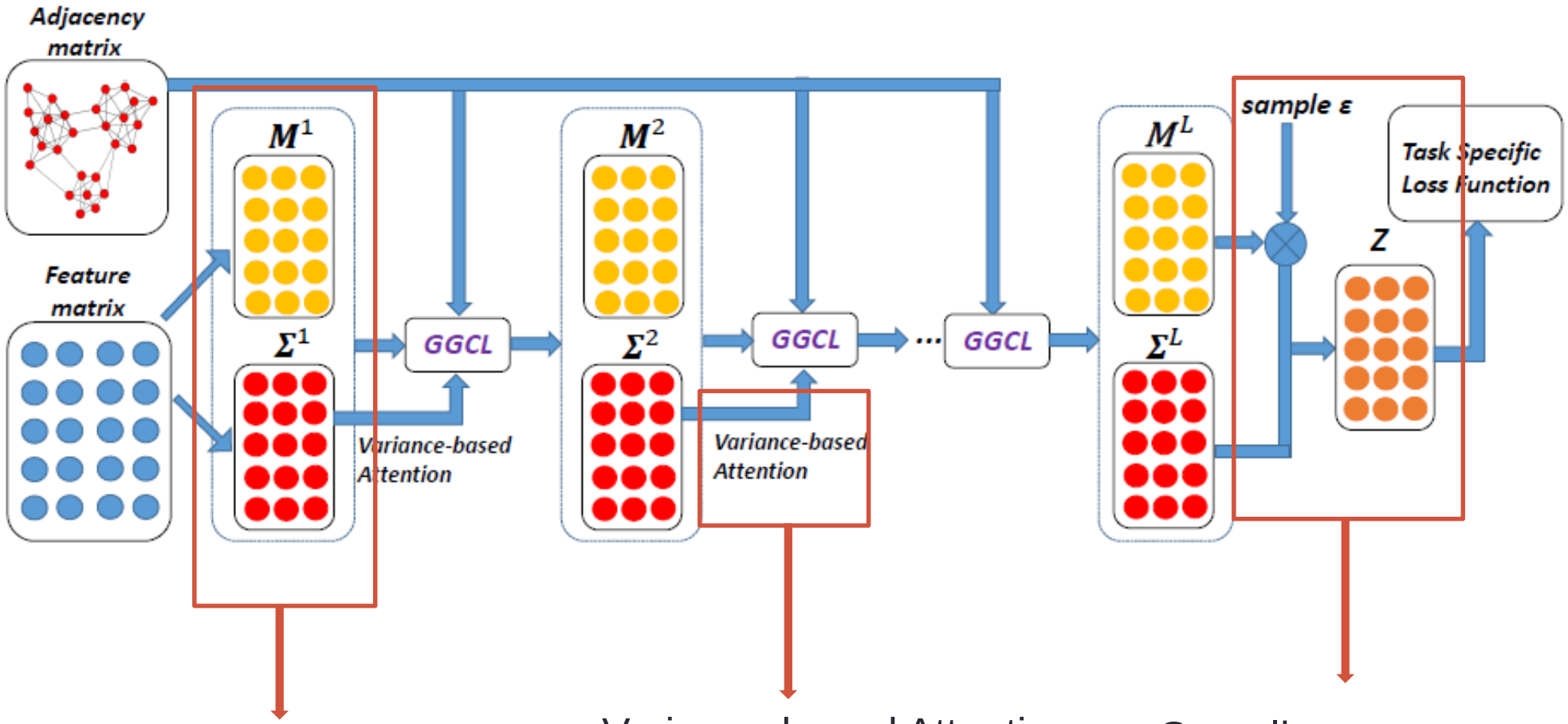
- How to enhance the robustness of GNNs against adversarial attacks?
- Adversarial attacks in node classification
  - Connect nodes from different communities to confuse the classifier



- Distribution vs. plain vectors
  - Plain vectors cannot adapt to such changes
  - Variances can help to absorb the effects of adversarial changes
  - Gaussian distributions → Hidden representations of nodes

Dingyuan Zhu, Ziwei Zhang, Peng Cui, Wenwu Zhu. Robust Graph Convolutional Networks Against Adversarial Attacks. ***KDD***, 2019.

The Framework of RGCN



Gaussian based representations: variance terms absorb the effects of adversarial attacks

Variance-based Attention: Remedy the propagation of adversarial attacks

Sampling process: explicitly considers mathematical relevance between means and variances

Dingyuan Zhu, Ziwei Zhang, Peng Cui, Wenwu Zhu. Robust Graph Convolutional Networks Against Adversarial Attacks. ***KDD***, 2019.

Experimental Results

- Node Classification on Clean Datasets

	Cora	Citeseer	Pubmed
GCN	81.5	70.9	79.0
GAT	83.0	**72.5**	79.0
RGCN	**83.1**	71.3	**79.2**

- Against Non-targeted Adversarial Attacks

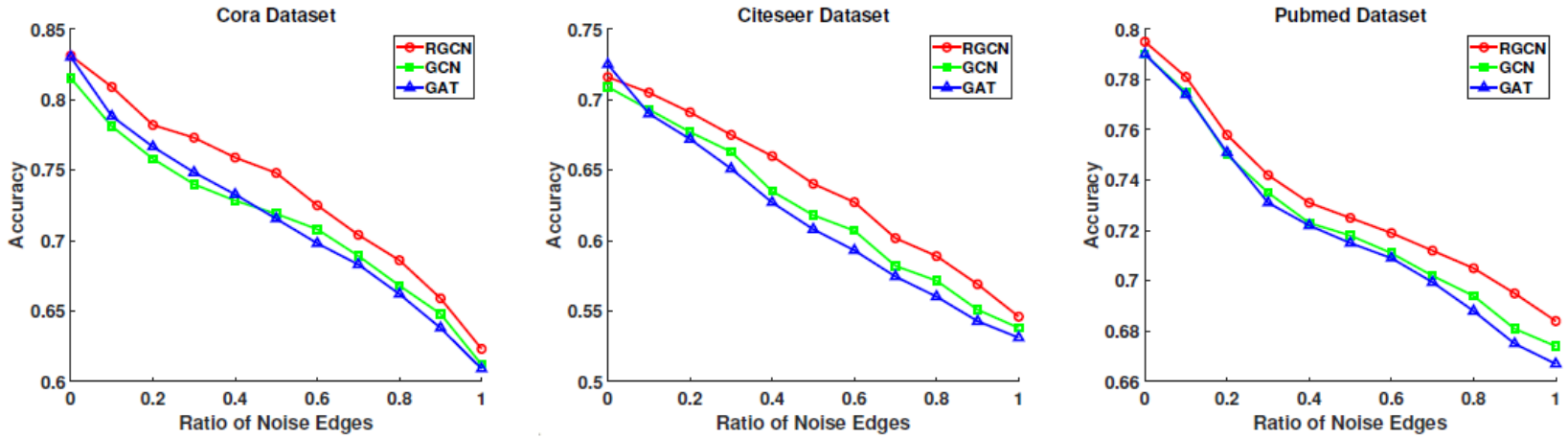


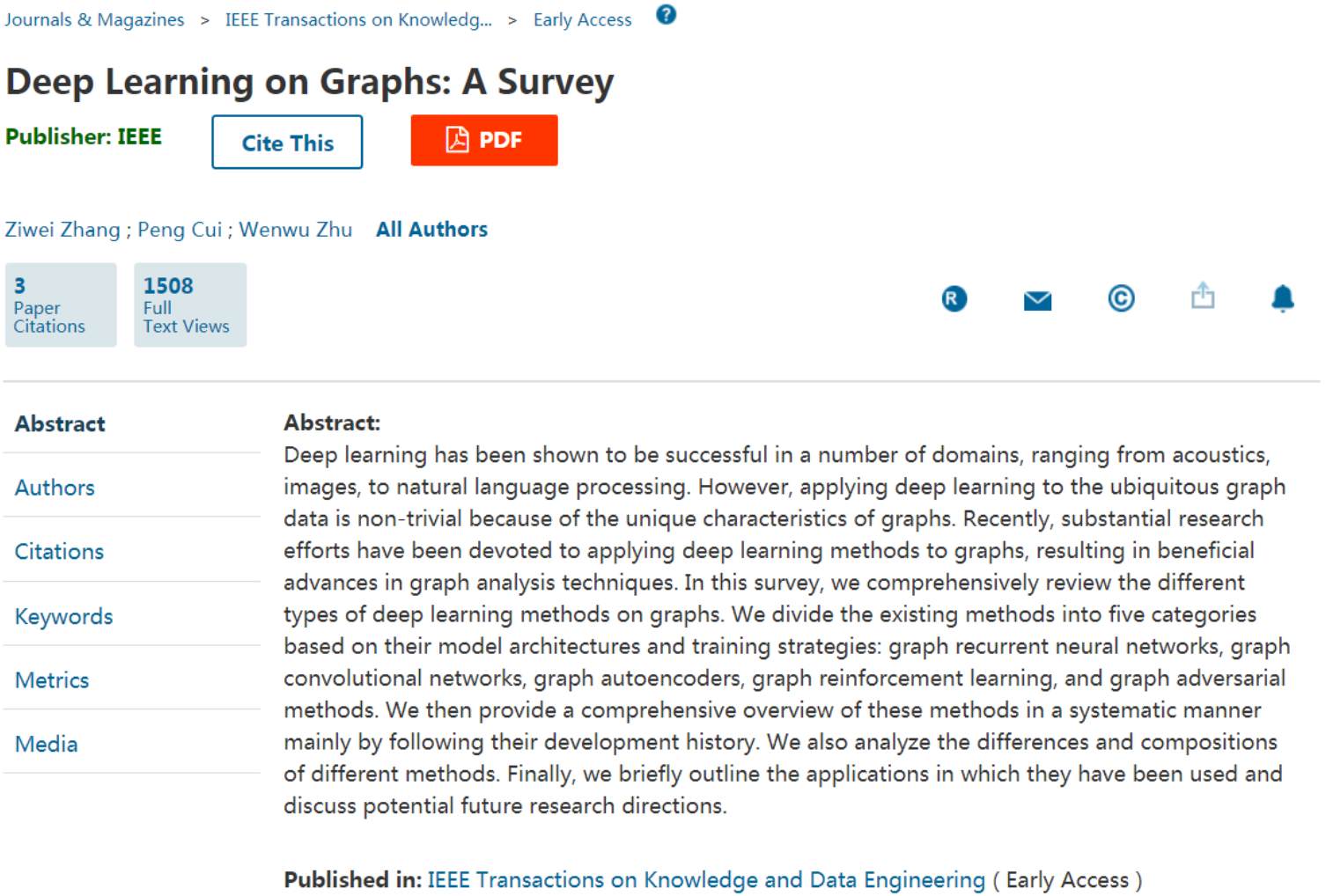
Figure 2: Results of different methods when adopting Random Attack as the attack method.

Dingyuan Zhu, Ziwei Zhang, Peng Cui, Wenwu Zhu. Robust Graph Convolutional Networks Against Adversarial Attacks. ***KDD***, 2019.

Recap: Graph Neural Networks

- Message-passing framework of GNNs
- Frontiers:
  - Does GNN fuse *feature* and *topology* optimally?
  - Technical challenges in real applications: robustness

Deep Learning on Graphs: A Survey



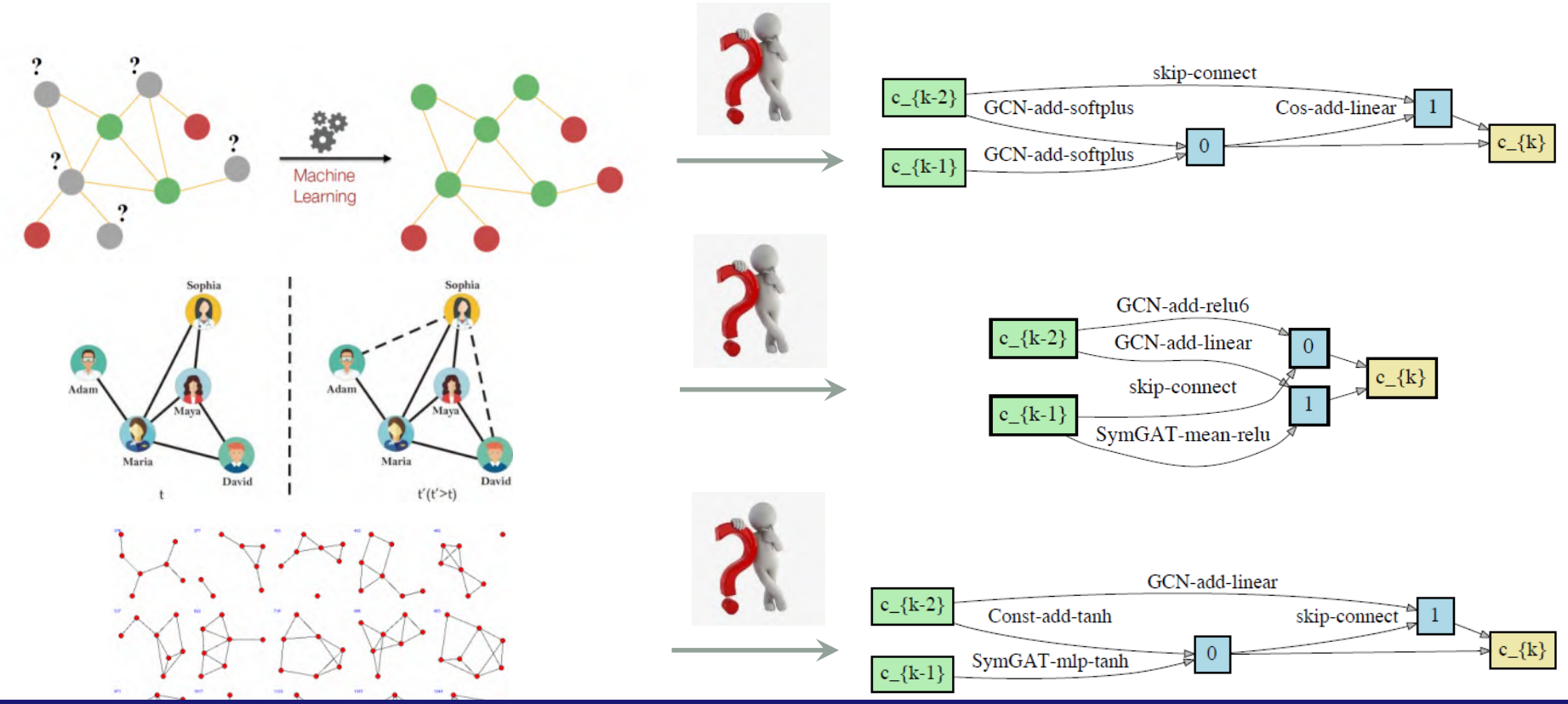
Ziwei Zhang, Peng Cui, Wenwu Zhu. **Deep Learning on Graphs: A Survey**. *IEEE TKDE, 2020.*

Learning from networks

Automated Graph Machine Learning

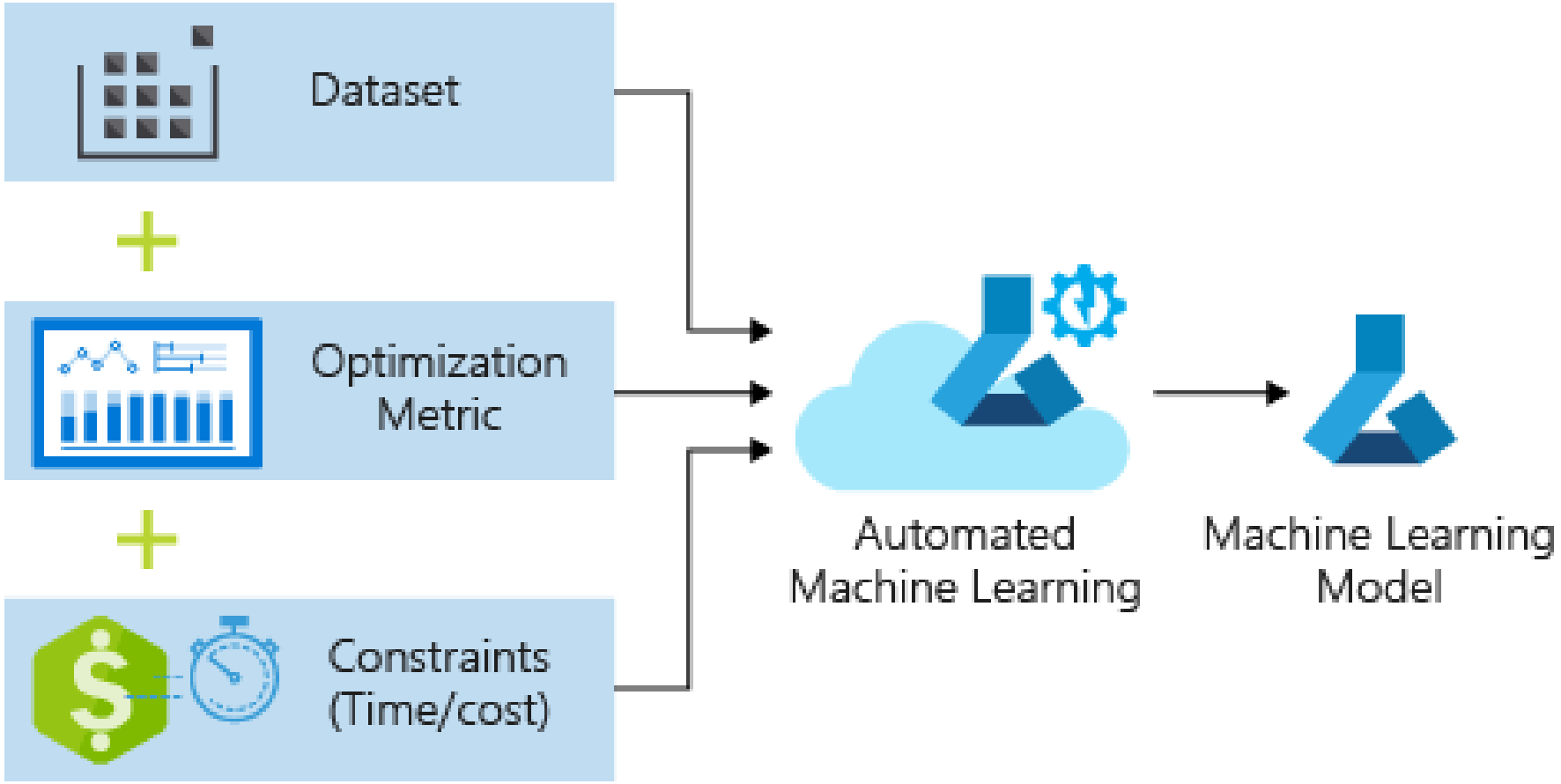
Problems in Existing Graph Learning Methods

- Manually design architectures and hyper-parameters through trial-and-error
- Each task needs be handled separately



Automated graph machine learning is critically needed!

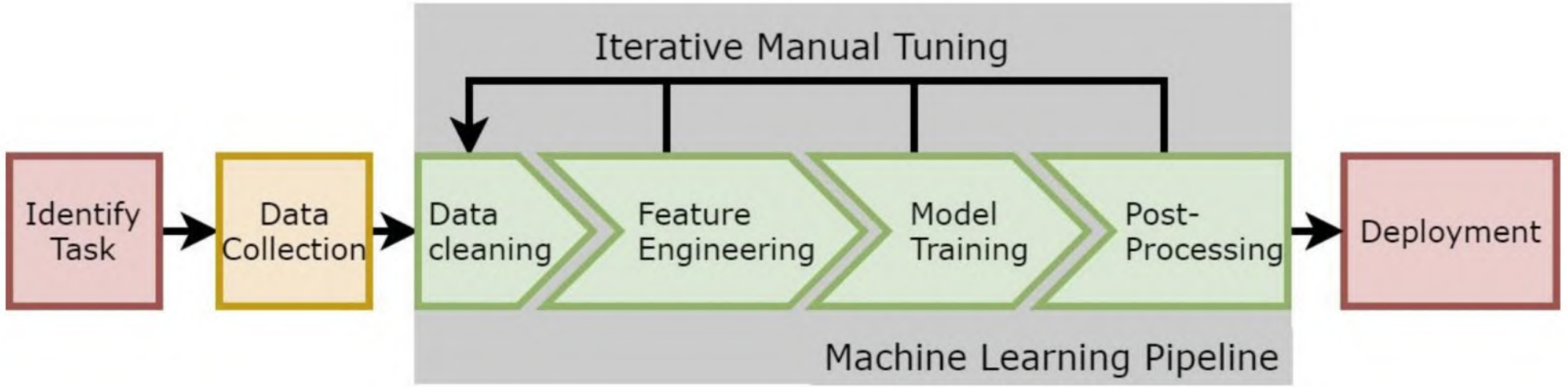
A Glance of AutoML



Design ML methods → Design AutoML methods

Picture credit to Microsoft Azure Machine Learning AutoML

ML vs. AutoML



- Rely on **expert knowledge**
- **Tedious** trail-and-error
- **Low** tuning **efficiency**
- **Limited** by human design

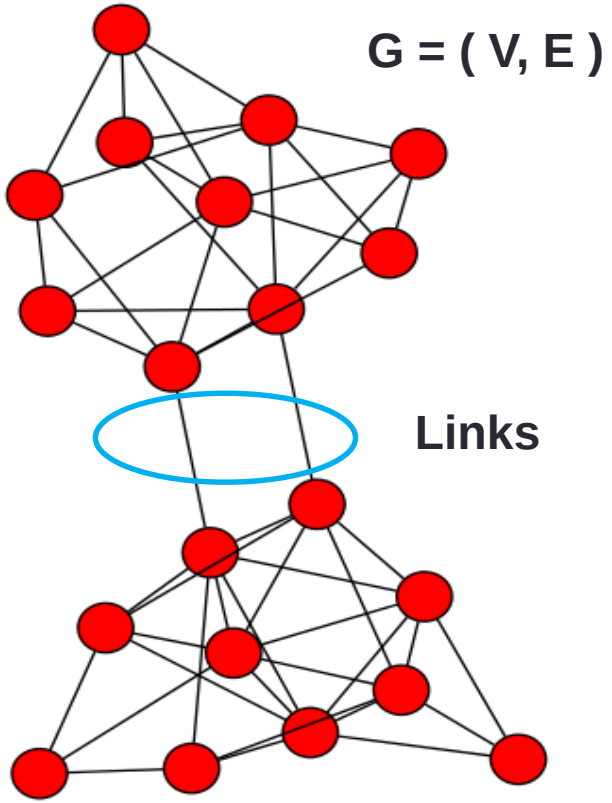
- **Free human** out of the loop
- **High** optimization **efficiency**
- **Discover & extract** patterns and combinations **automatically**

Automated Graph Learning

- Automated Machine Learning on Graph
  - Graph Hyper-Parameter Optimization (HPO)
  - Graph Neural Architecture Search (NAS)

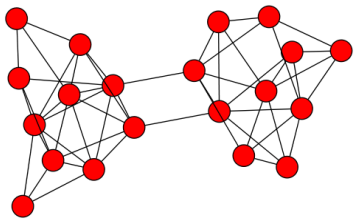
- The key: *Graph Structure!*

Various diverse graph structures may place complex impacts on graph HPO and graph NAS



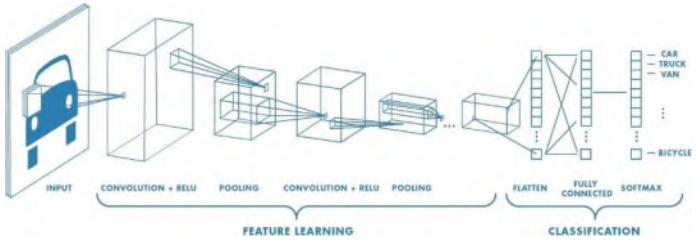
Challenge: Unique Graph ML Architecture

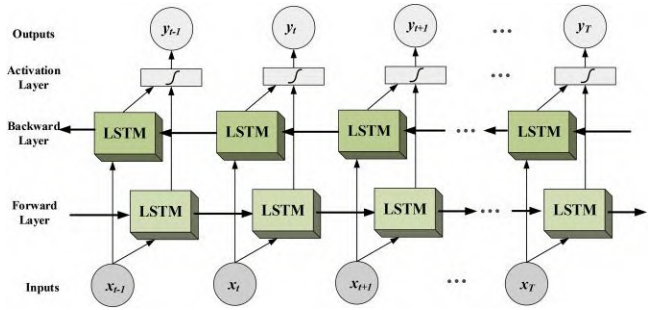
Data

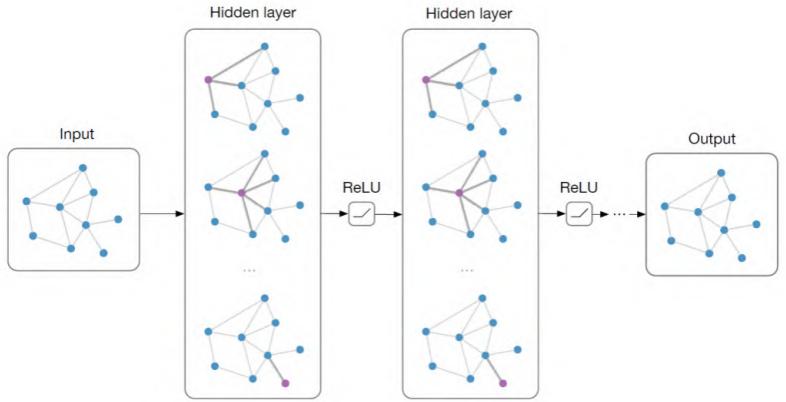


G = (V, E)

NN architecture







Search Space

- zeroize
- skip-connect
- 1×1 conv
- 3×3 conv
- 3×3 avg pool

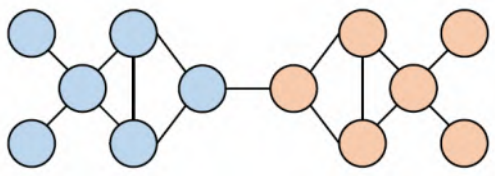
predefined operation set

Linear: $f(x_1, .., x_n) = W_1 x_1 + .. + W_n x_n + b$,
Blending (element wise): $f(z, x, y) = z \odot x + (1 - z) \odot y$
Element wise product and sum,
Activations: Tanh, Sigmoid, and LeakyReLU.

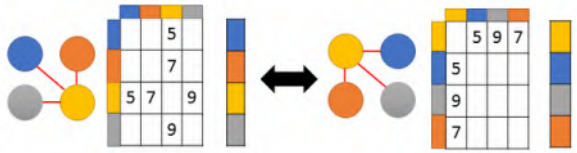
?

Semi-Supervised Classification with Graph Convolutional Networks, *ICLR 2017*
NAS-Bench-201 Extending the Scope of Reproducible Neural Architecture Search, *ICLR 2020*
NAS-Bench-NLP Neural Architecture Search Benchmark for Natural Language Processing, *arXiv 2020*

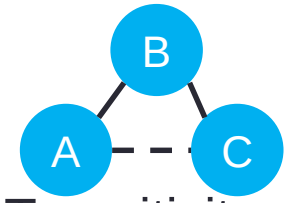
Challenge: Complexity and diversity of graph tasks



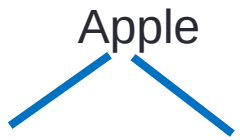
High-order Proximity



Permutation-equivariance



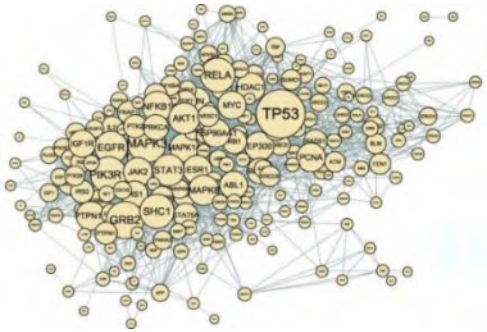
Transitivity



Non-transitivity

- Link Prediction
- Community Detection
- Node Classification
- Network Distance
- Node Importance
- Graph Classification
- Graph Matching

…



Various graph properties

Various applications

Various domains

- No single method can perfectly handle all scenarios

Challenge: Scalability

Social Networks

- WeChat: 1.2 billion monthly active users (Sep 2020)
- Facebook: 2.8 billion active users (2020)

E-commerce Networks

- Millions of sellers, about 0.9 billion buyers, 10.6 trillion turnovers in China (2019)

Citation Networks

- 133 million authors, 277 million publications, 1.1 billion citations (AMiner, Feb 2021)

Challenge: how to handle billion-scale graphs?

Outline

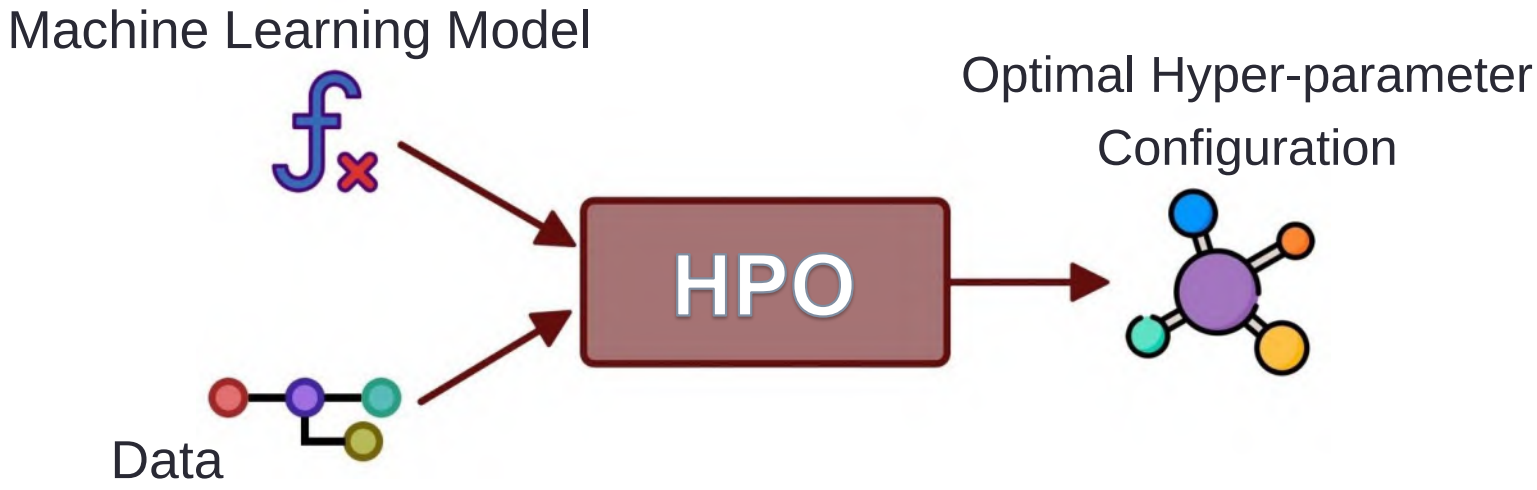
- Graph Hyper-parameter Optimization
- Graph Neural Architecture Search
- Automated Graph Learning Libraries

Outline

- Graph Hyper-parameter Optimization
- Graph Neural Architecture Search
- Automated Graph Learning Libraries

Hyper-Parameter Optimization

- Goal: automatically find the optimal hyper-parameters



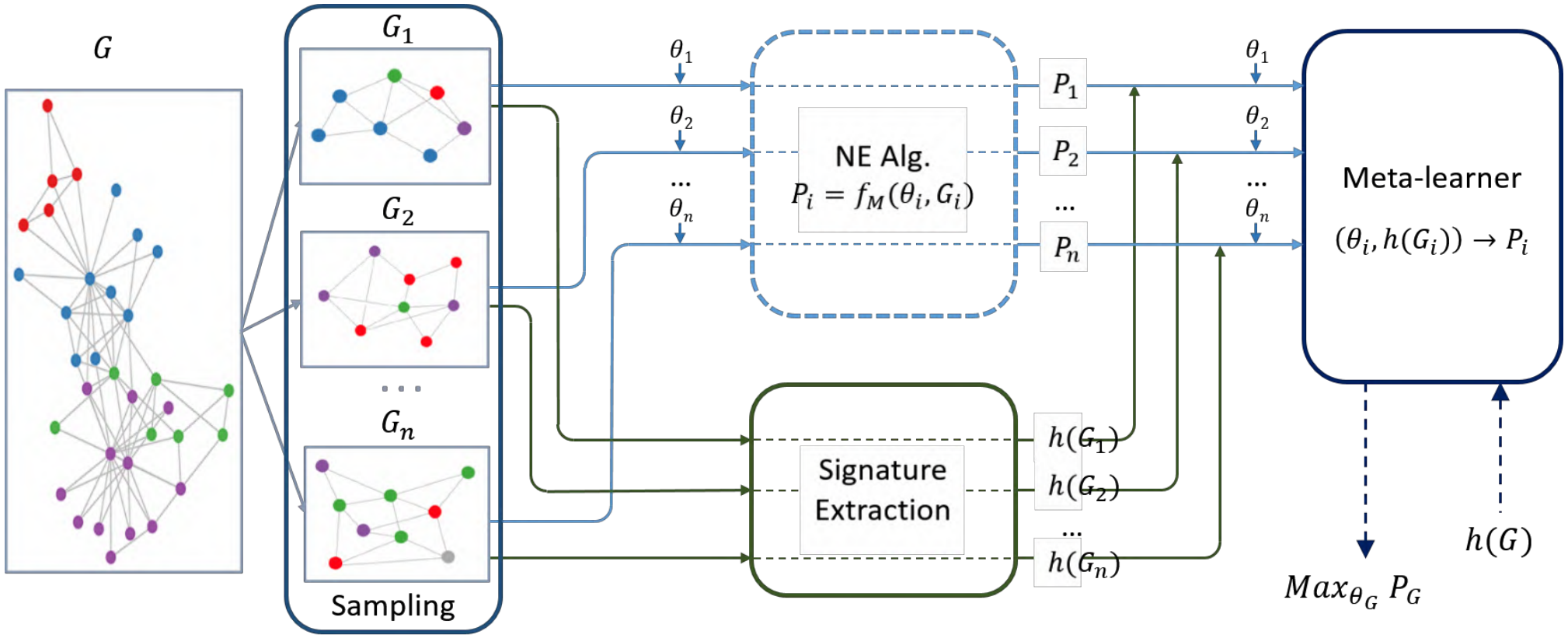
- Formulation: bi-level optimization

$$\min_{\alpha \in \mathcal{A}} \mathcal{L}_{val}\left(\mathbf{W}^*(\alpha), \alpha\right)$$

$$\text{s.t.} \quad \mathbf{W}^*(\alpha) = \arg\min_{\mathbf{W}} \left(\mathcal{L}_{train}\left(\mathbf{W}, \alpha\right)\right)$$

- Challenge: each trial of the inner loop on graph is computationally expensive, especially for large-scale graphs

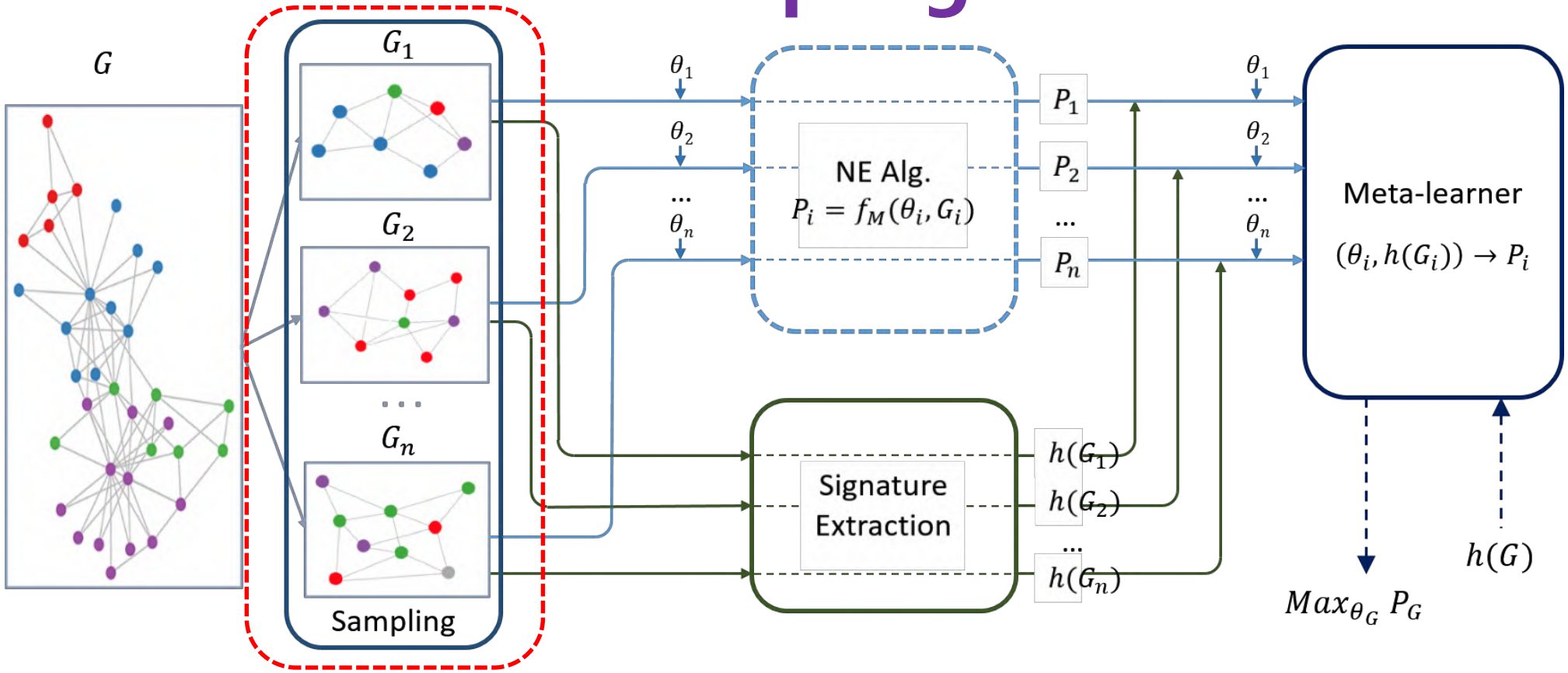
AutoNE: Framework



Transfer the knowledge about optimal hyper-parameters from sampled subgraphs to the original massive graph

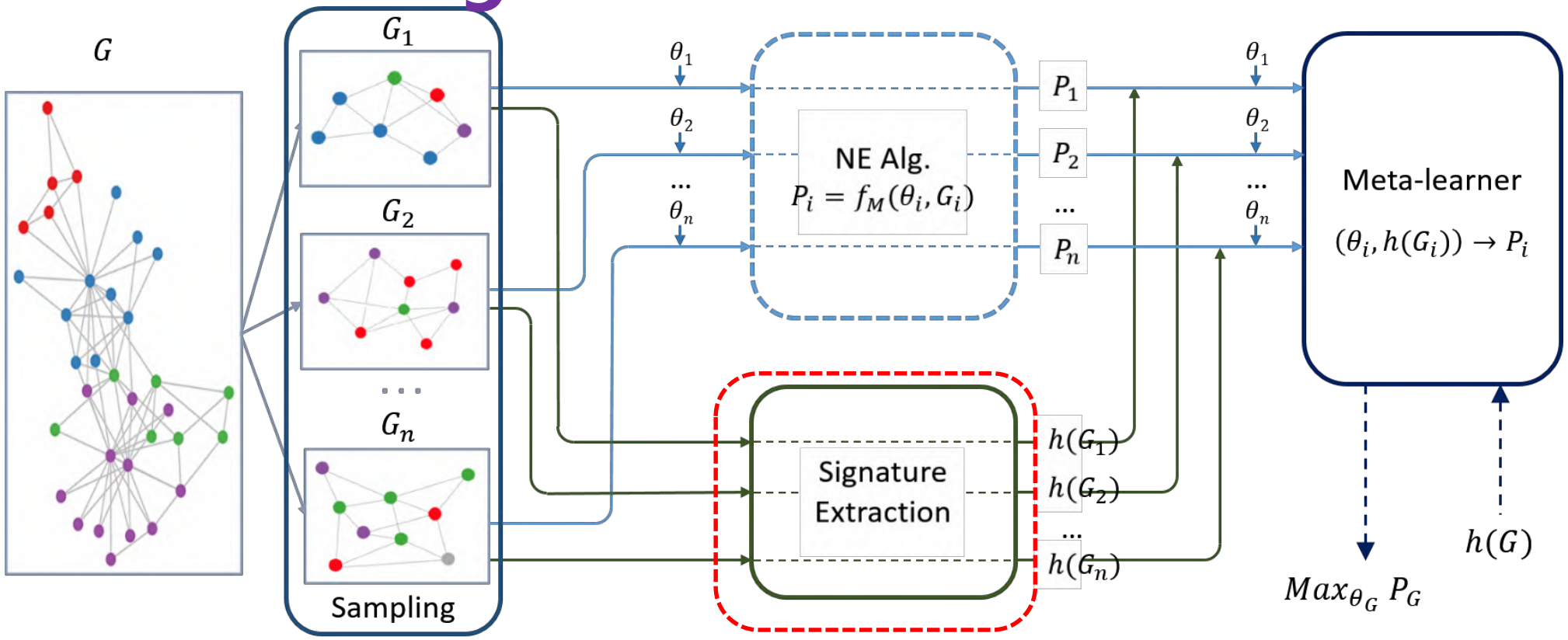
Tu Ke, Jianxin Ma, Peng Cui, Jian Pei, and Wenwu Zhu.
AutoNE: Hyperparameter optimization for massive network embedding. ***KDD 2019***.

AutoNE: Sampling Module



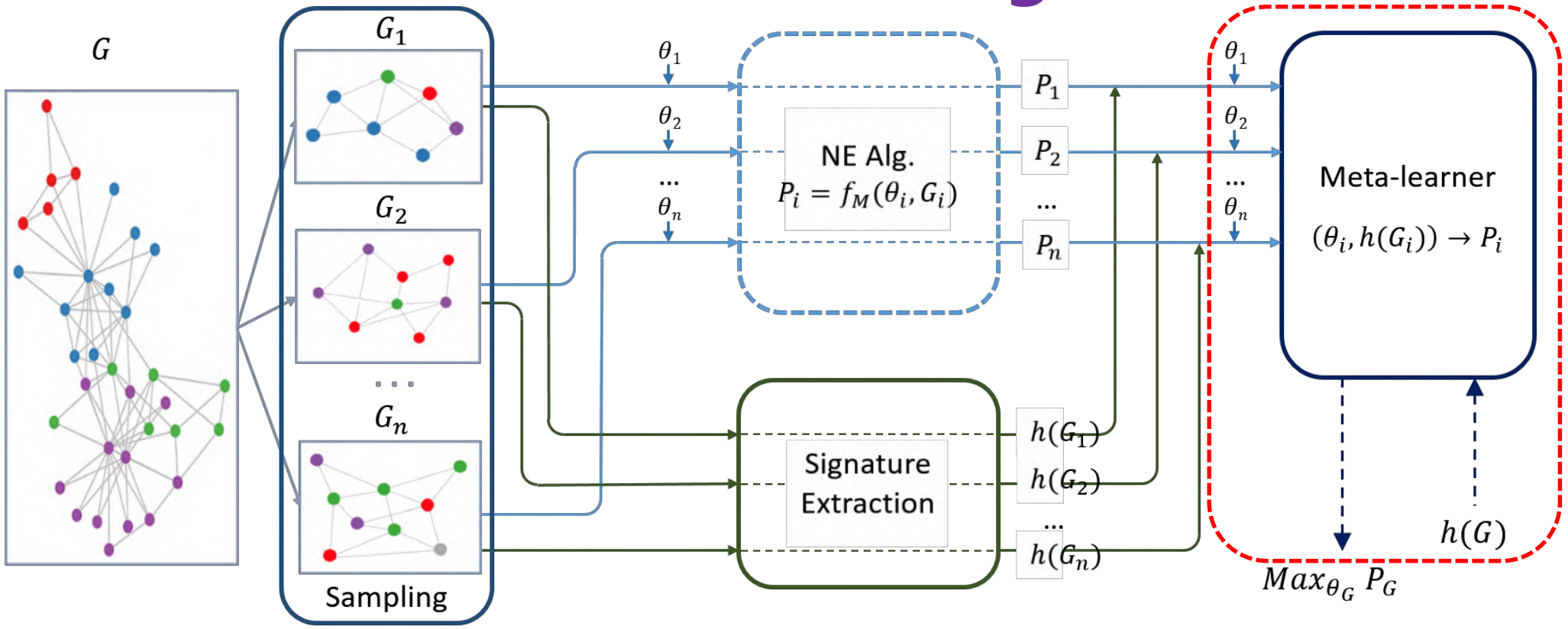
- **Goal**: sample representative subgraphs that share similar properties with the original large-scale graph
- **Challenge**: preserve diversity of the origin graph
- **Method**: multi-start random walk strategy
  - Supervised: nodes with different labels
  - Unsupervised: from different discovered communities, e.g., a greedy algorithm that maximizes modularity

AutoNE: Signature Extraction Module



- **Goal**: learn a vector representation for each subgraph so that knowledge can be transferred
- **Challenge**: learn comprehensive graph signatures
- **Method**: NetLSD [Tsitsulin et al. KDD18]
  - Based on spectral graph theory, heat diffusion process on a graph $h_t(G) = tr(H_t) = tr\left(e^{-tL}\right) = \sum_j e^{-t\lambda_j}$

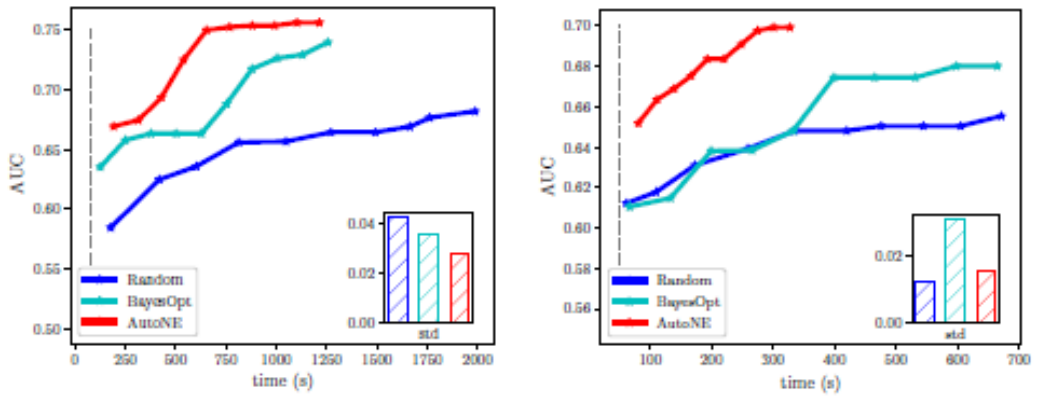
AutoNE: Meta-Learning Module



- **Goal**: transfer knowledge about hyper-parameters of subgraphs to the original large-scale graph
- **Assumption**: two similar graphs have similar optimal hyper-parameter
- **Method**: Gaussian Process based meta-learner

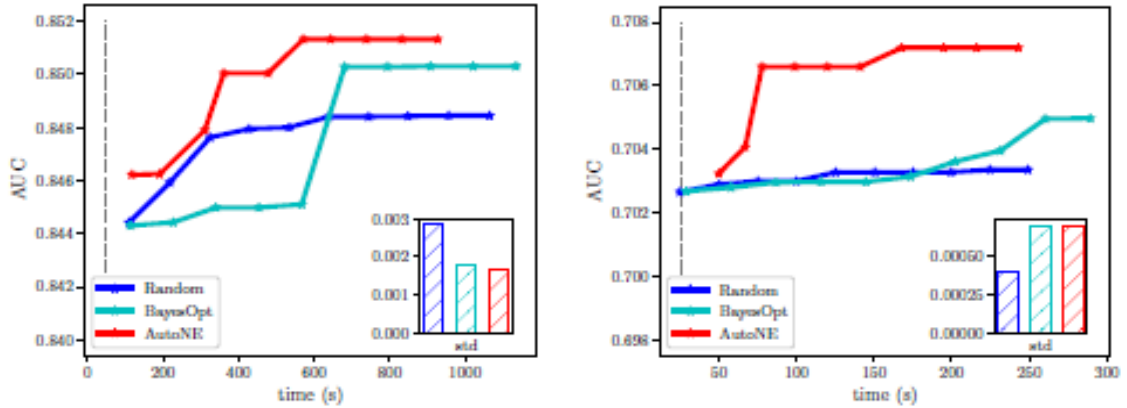
$$\ln p(\mathbf{f} \mid \mathbf{X}) = -\frac{1}{2}\mathbf{f}^{\top} K(\mathbf{X}, \mathbf{X})^{-1}\mathbf{f} - \frac{1}{2}\ln\det(K(\mathbf{X}, \mathbf{X})) + constant.$$

AutoNE: Experiments



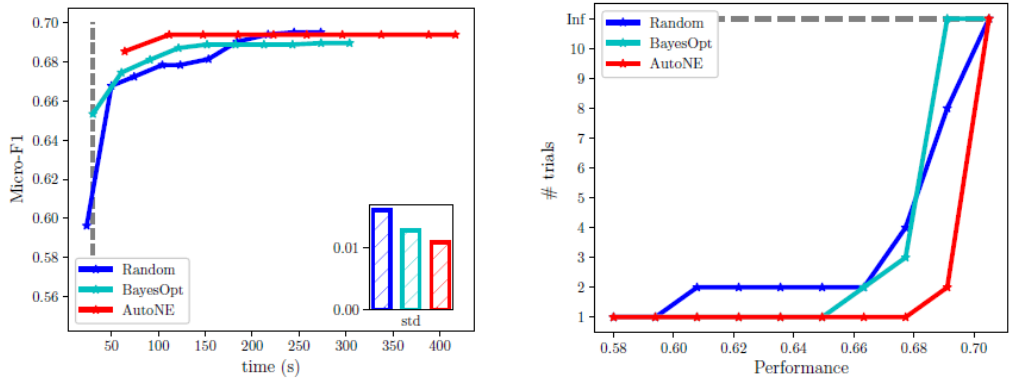
(c) Link prediction on BlogCatalog (d) Link prediction on Wikipedia

Sampling-Based Graph ML



(c) Link prediction on BlogCatalog (d) Link prediction on Wikipedia

Factorization-Based Graph ML



(a) The performance achieved by each method within various time thresholds. (b) The number of trials required to reach a certain performance threshold.

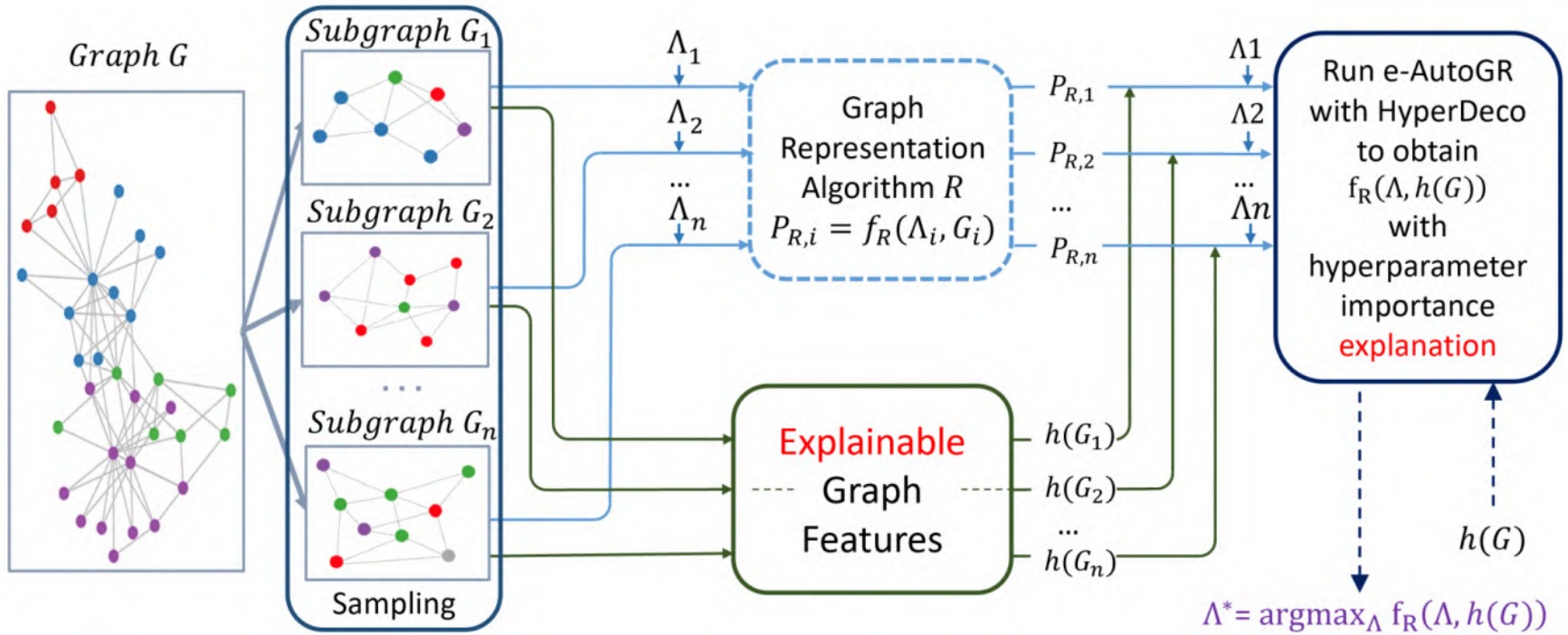
Graph Neural Networks

Table 1: Results on a massive network with around thirty million edges, where we can only afford to run a NE algorithm on the whole network for a few times.

Method	Trial 1		Trial 2		Trial 3	
	AUC	Time(s)	AUC	Time(s)	AUC	Time(s)
AutoNE	0.717	1067.9	0.726	1856.2	**0.769**	2641.9
Random	0.714	698.3	0.727	1426.3	0.715	2088.6
BayesOpt	0.715	702.5	0.714	1405.1	0.727	2307.7

Large-Scale Graphs

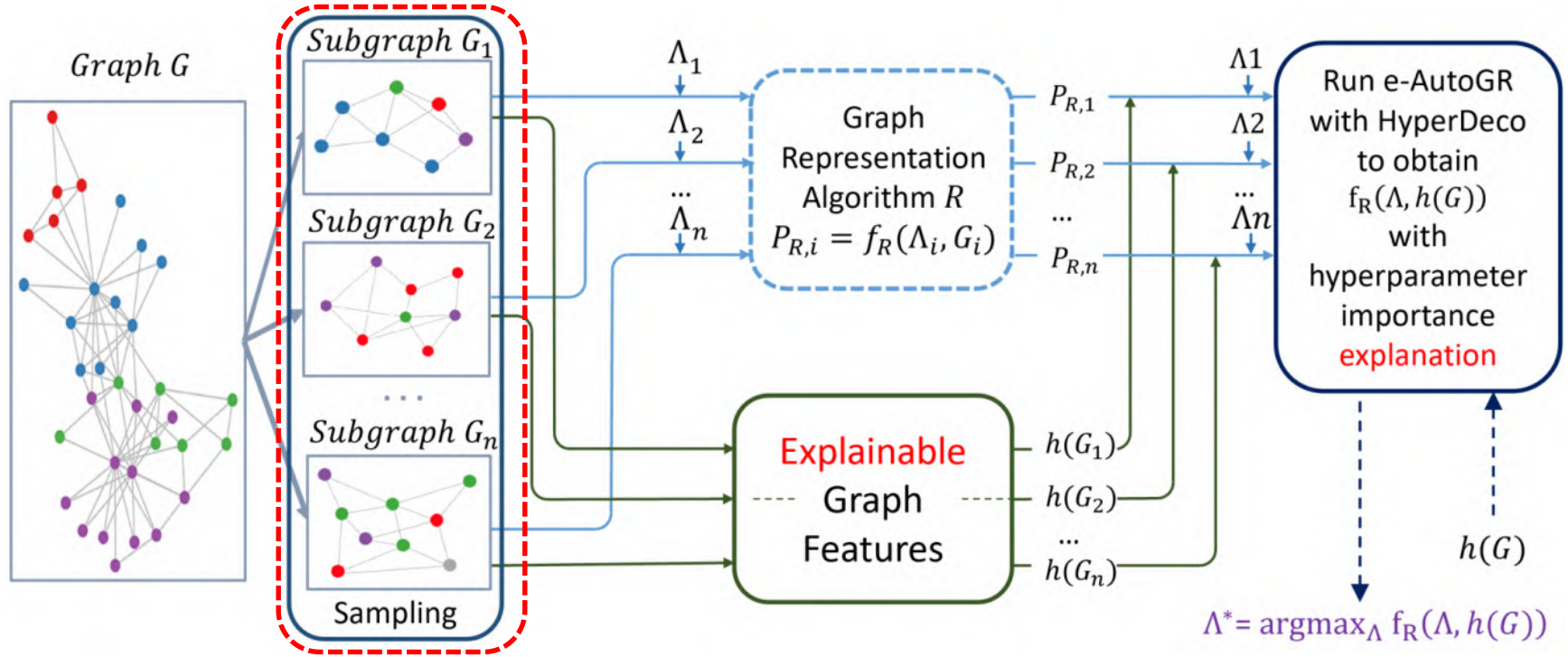
e-AutoGR: Overview



Transfer the knowledge about optimal hyperparameters from the sub-graphs to the original graph in an explainable way

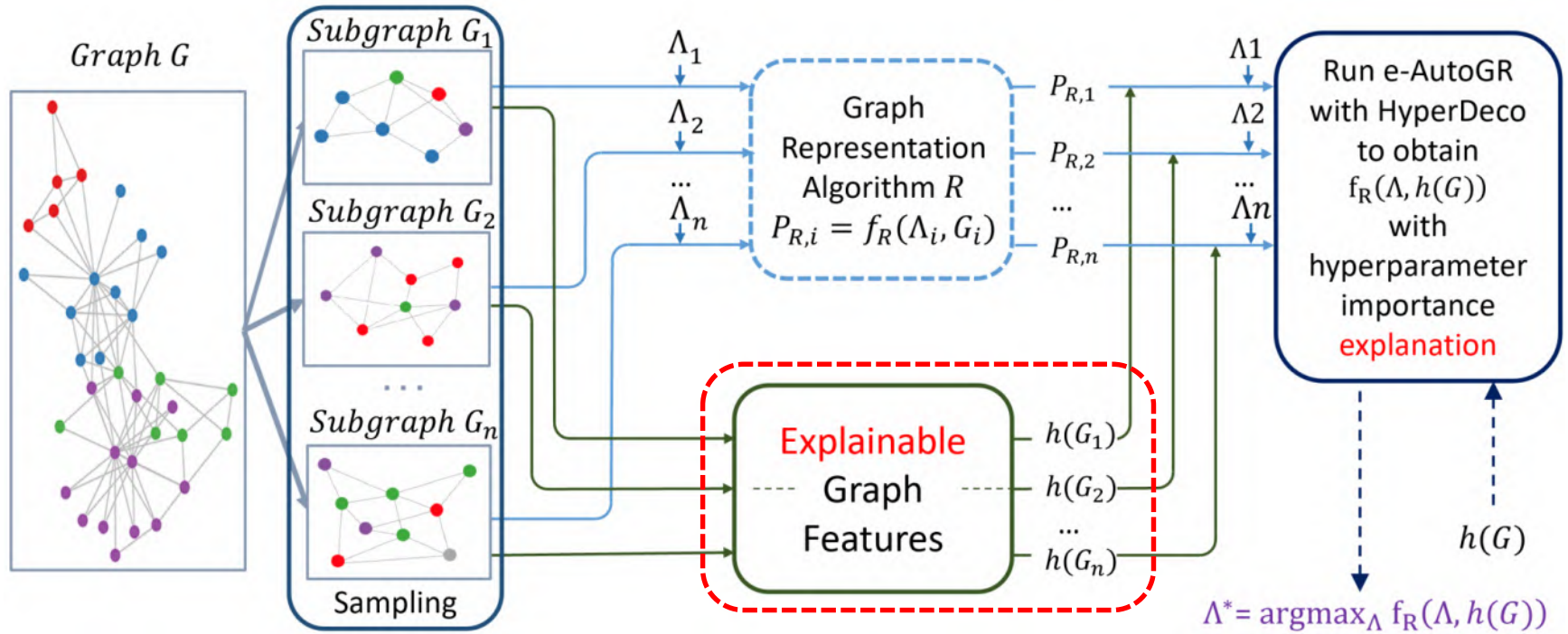
Wang Xin, Shuyi Fan, Kun Kuang, and Wenwu Zhu. Explainable Automated Graph Representation Learning with Hyperparameter Importance. ***ICML 2021***.

e-AutoGR: Overview



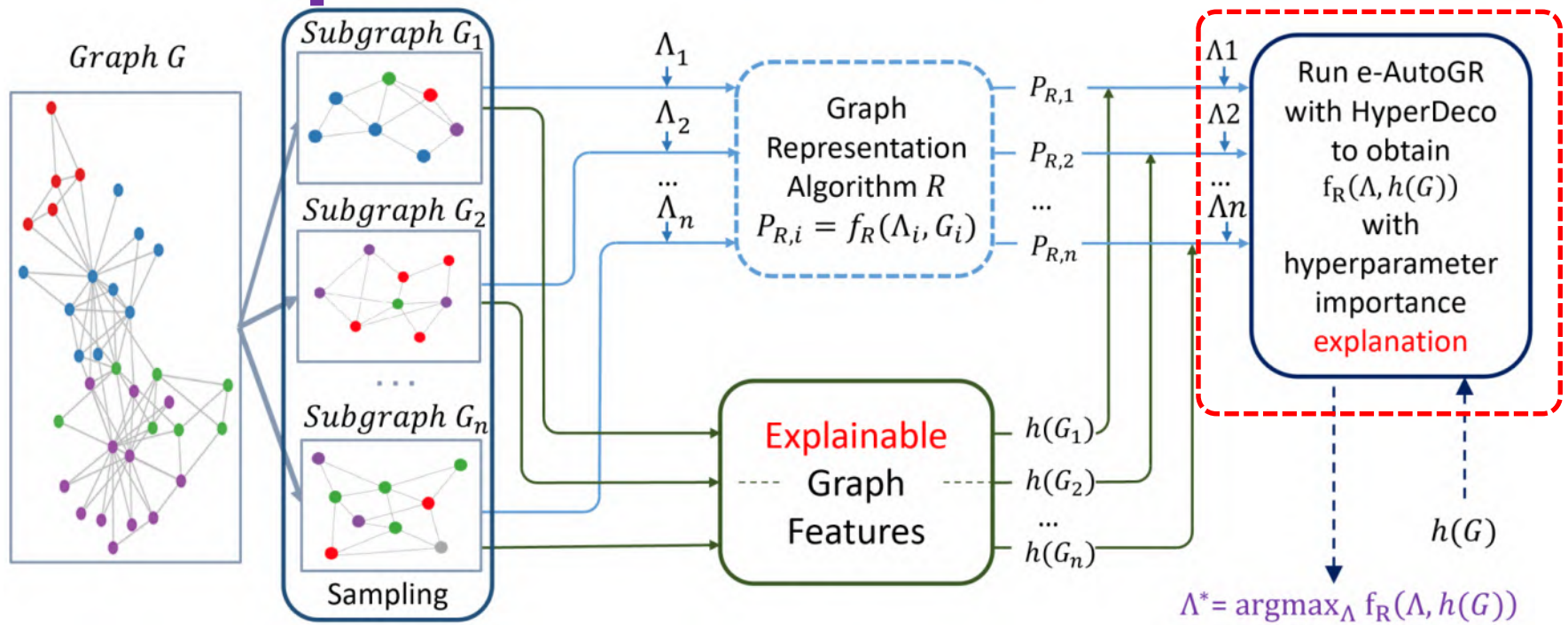
- **Goal**: sample representative subgraphs that share similar properties with the original large-scale graph, similar to AutoNE.
- **Challenge**: preserve diversity of the origin graph
- **Method**: multi-start random walk strategy
  - Supervised: nodes with different labels
  - Unsupervised: from different discovered communities, e.g., a greedy algorithm that maximizes modularity

e-AutoGR: Sampling Module



- **Goal**: Extract explainable graph features that can measure similarities.
- **Six explainable graph features**
  - Number of nodes: |V|
  - Number of edges: |E|
  - Number of triangles
  - Global clustering coefficient: $\frac{3 * Number\ of\ triangles}{Number\ of\ triplets}$
  - Maximum total degree value
  - Number of components
- **Similarity: Canberra Distance** $g^i = d\,(f^i, f) = \sum_{k=1}^{6} \frac{|f_k^i - f_k|}{|f_k^i| + |f_k|}$

e-AutoGR: Explainable Feature Extraction Module



- **Goal**: Learn performance function on small sampled graphs and predict on the origin massive graph in an **explainable manner**
- **Method:**
  - Adopt **explainable** graph features
  - **Decorrelate the correlations** between different hyper-parameters given explainable graph features when learning the performance function

e-AutoGR: Algorithms

Algorithm 1 Hyperparameter Decorrelation Weighting Regression (HyperDeco)

Input: Observed $\mathbf{X} = [\mathbf{A}, \mathbf{B}]$ and performance Y, where $\mathbf{A}$ denotes hyperparameters and $\mathbf{B}$ denotes graph features.
Output: Updated parameters γ, Θ.
Initialize parameters $\gamma^{(0)}$ and $\Theta^{(0)}$,
Calculate loss function with parameters $(\gamma^{(0)}, \Theta^{(0)})$,
Initialize the iteration variable $t \leftarrow 0$,
repeat
    $t \leftarrow t + 1$,
    Update $\gamma^{(t)}$ with gradient descent by fixing Θ,
    Update $\Theta^{(t)}$ with gradient descent by fixing γ,
    Calculate loss function with parameters $(\gamma^{(t)}, \Theta^{(t)})$,
until Loss function converges or max iteration is reached

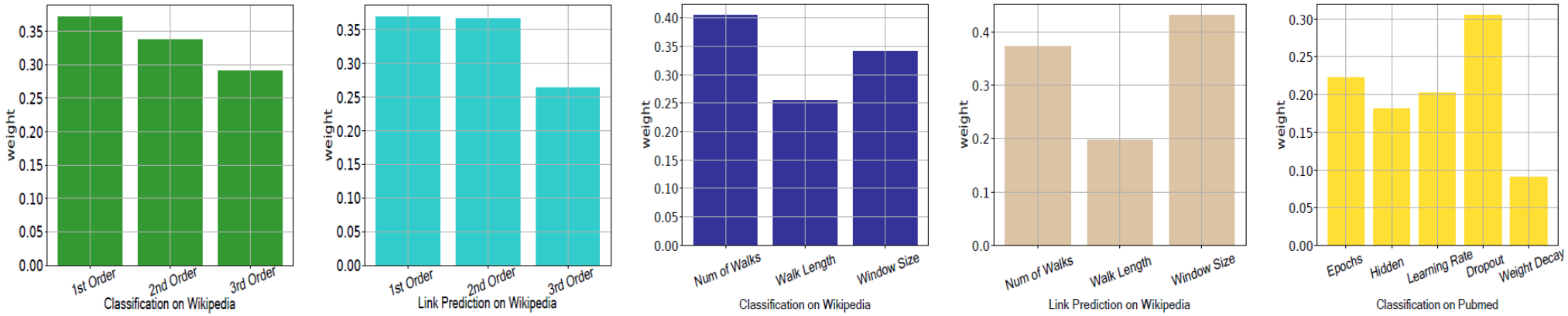
Algorithm 2 Explainable Automated Graph Representation (e-AutoGR)

Input: Graph G, Graph representation algorithm R.
Output: The optimal hyperparameter configuration Λ^*.
Sample s subgraphs $G_i, i = 1, 2, \ldots, s$ from original graph G according to Section 3.2.1.
Decide t_i for each G_i according to Section 3.2.2.
Execute algorithm R on each subgraph G_i for t_i times and obtain hyperparameter matrix A and graph feature matrix B as well as the performance vector Y.
Initialize $Count = T$.
repeat
    Execute **Step 1** to **Step 3** in Section 3.4.
    $Count = Count - 1$.
until $Count == 0$

e-AutoGR: Experiments

Table 1. Best performance for each comparable automated graph representation approach on different datasets in terms of *link prediction* and *node classification* tasks over various graph representation algorithms. Bold font denotes the best approach.

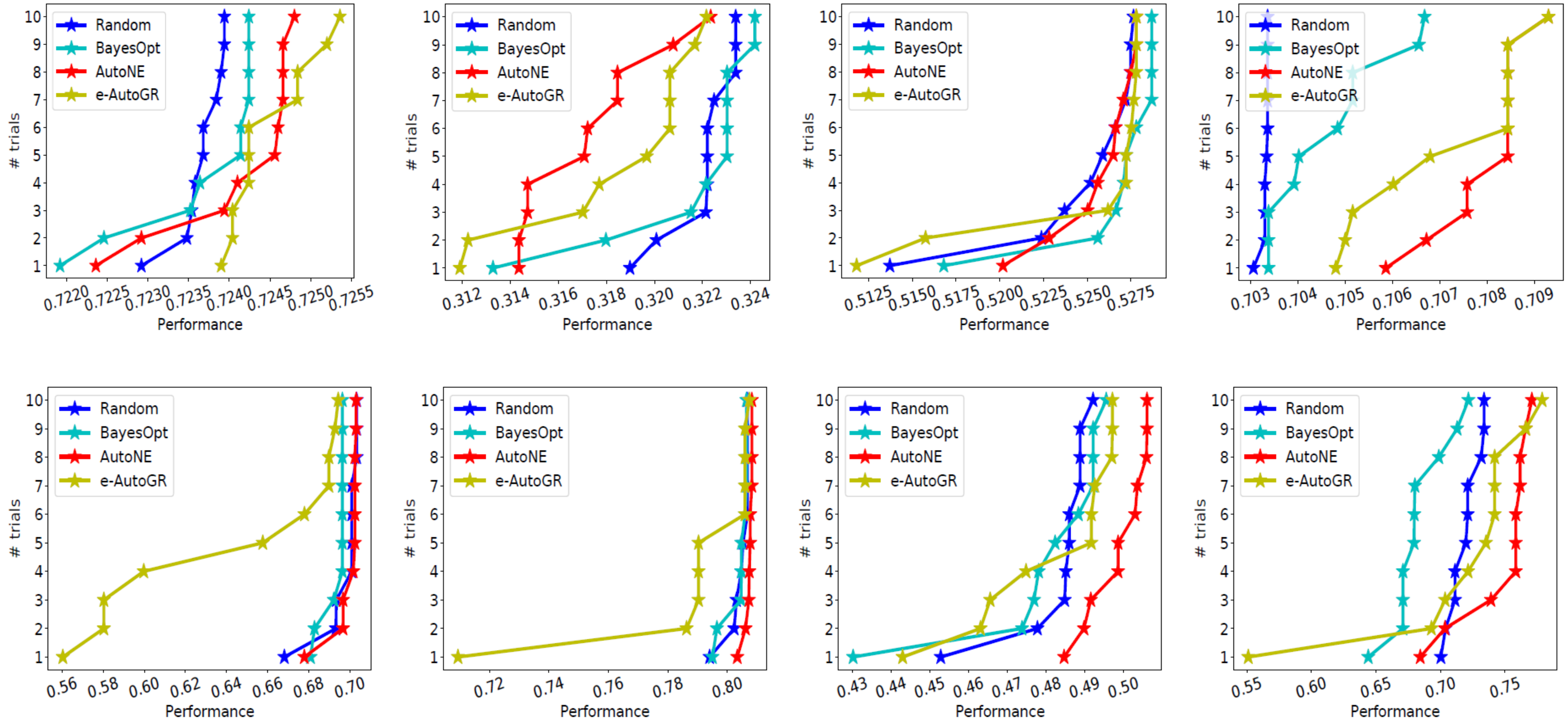
Dataset	Algorithm	Task	e-AutoGR	AutoNE	Random	Bayesian
BlogCatalog	Deepwalk	Link Prediction	**0.871817**	0.792662	0.803191	0.807158
BlogCatalog	Deepwalk	Classification	**0.414682**	0.414234	0.411551	0.407449
BlogCatalog	AROPE	Link Prediction	**0.852612**	0.851921	0.846578	0.851878
BlogCatalog	AROPE	Classification	**0.326721**	0.325060	0.326020	0.326428
Wikipedia	Deepwalk	Link Prediction	0.729228	**0.729330**	0.696462	0.713133
Wikipedia	Deepwalk	Classification	**0.519657**	0.509920	0.503319	0.502617
Wikipedia	AROPE	Link Prediction	**0.709392**	0.709383	0.703443	0.707619
Wikipedia	AROPE	Classification	0.529743	0.529011	**0.530418**	0.529732
Pubmed	Deepwalk	Link Prediction	**0.873301**	0.867633	0.853459	0.851824
Pubmed	Deepwalk	Classification	**0.810916**	0.810368	0.809199	0.810417
Pubmed	AROPE	Link Prediction	0.791435	**0.796737**	0.790228	0.790123
Pubmed	AROPE	Classification	**0.727413**	0.725412	0.725789	0.726411
Pubmed	GCN	Classification	**0.708830**	0.708712	0.708719	0.707918



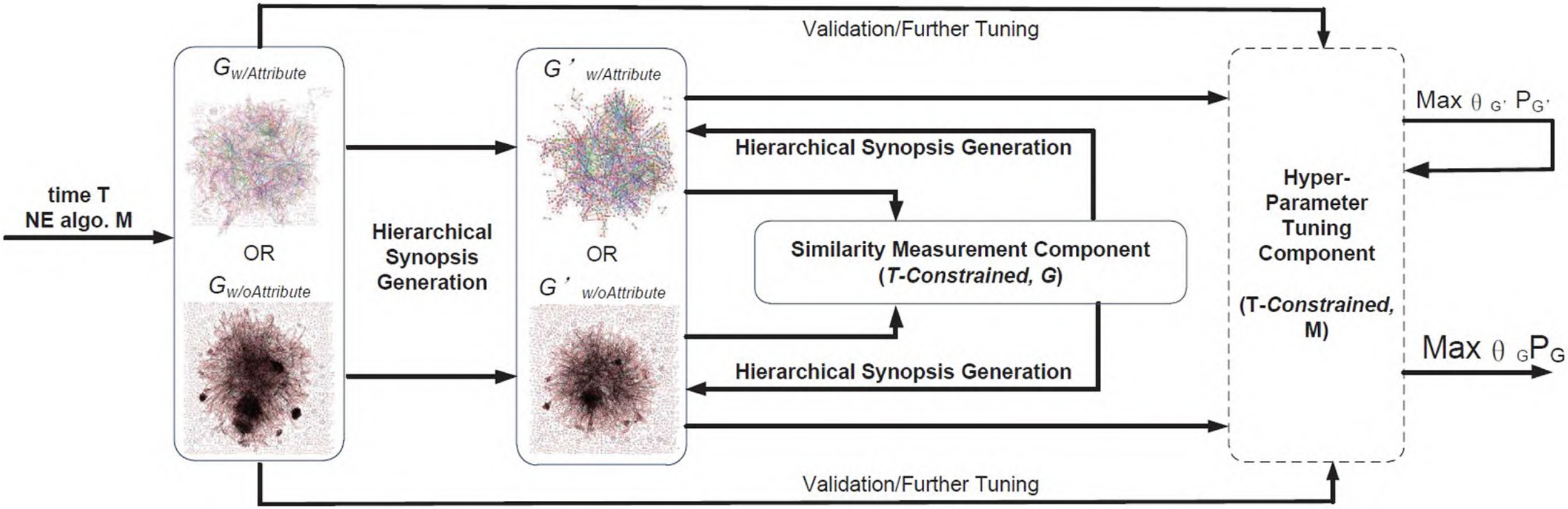
(a) AROPE (b) AROPE (c) DeepWalk (d) DeepWalk (f) GCN

Figure 3. Explaining hyperparameter importance in affecting performance.

e-AutoGR: Experiments



JITuNE

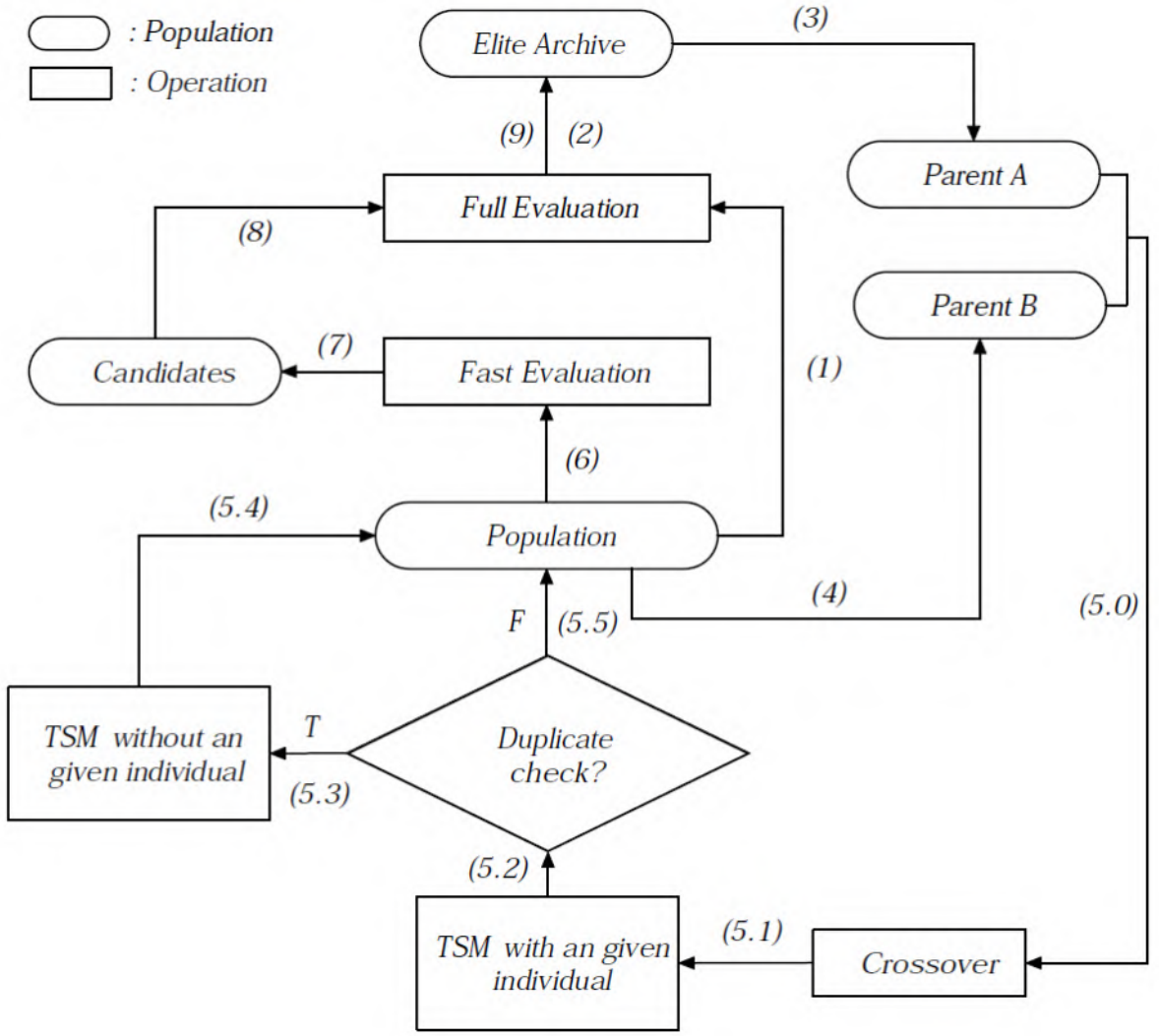


- Hierarchical Synopsis: multi-level hierarchical architecture of network
- Vs. sampling in AutoNE and eAutoGR:
  - Better time-constrained hyperparameter tuning
  - Direct transfer without the meta-learning process

JITuNE: Just-In-Time Hyperparameter Tuning for Network Embedding Algorithms. arXiv 2021.

HESGA-TSM

- Optimization: genetic algorithm
- Hierarchical evaluation
  - Fast evaluation to select candidates
    - Reduce training epochs
  - Full evaluation: fully train candidates models and test on validation set
  - Tradeoff between efficiency and effectiveness



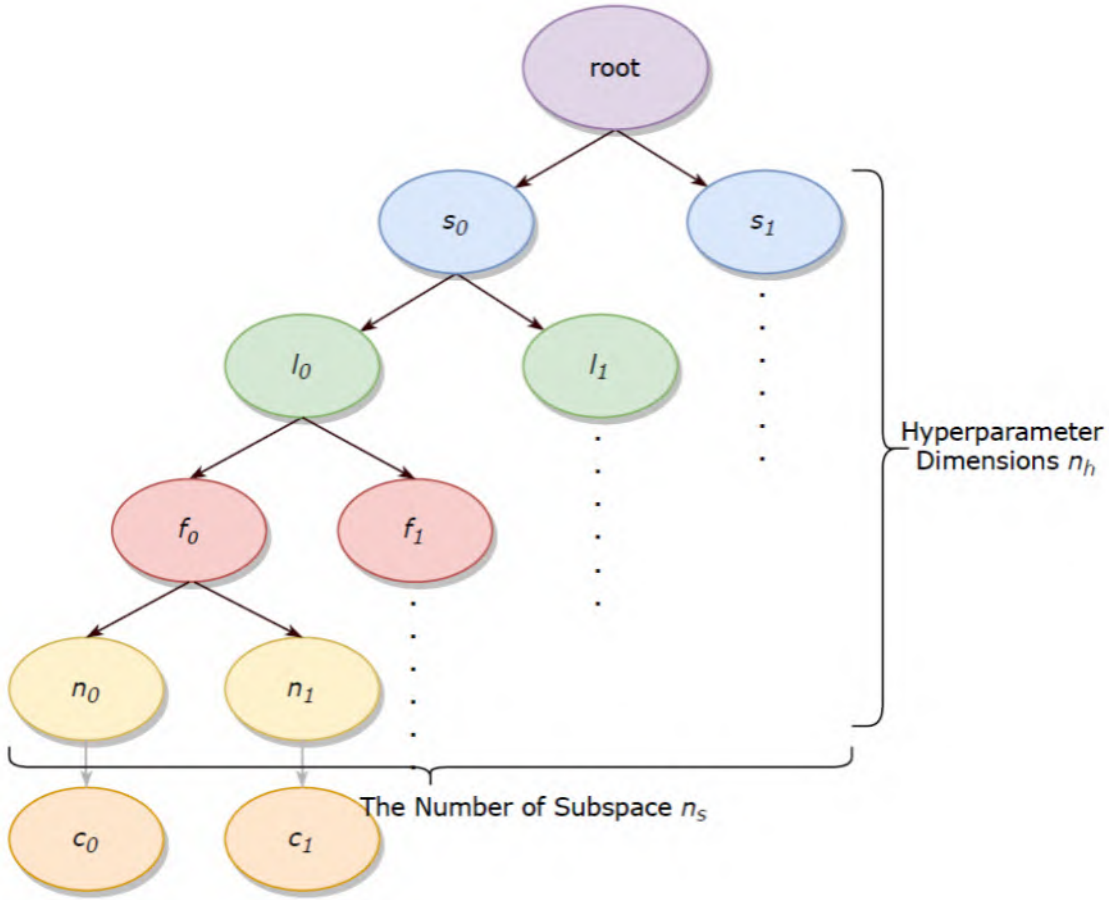
A Genetic Algorithm with Tree-structured Mutation for Hyperparameter Optimisation of Graph Neural Networks. arXiv 2021.

HESGA-TSM

- Tree-structure Mutation

Algorithm 1 TSM with An Given Individual

Input an individual s, $tree$, $t = [\ ]$
$p = tree(s)$ ▷ pathway p identification in tree
for $j = 1 \rightarrow\ n_h$ **do** ▷ in pathway p, the maximum depth of tree without leaf nodes n_h
    $t_{total} += f(t_j)$ ▷ t_j denotes the times recorded in node j, f is reciprocal function
    t.append($f(t_j)$)
end for
$t' \leftarrow$ each element in t is divided by t_{total}
$i = rws(n_h, t')$ ▷ node i, rws roulette wheel selection
$r \leftarrow$ the defined value range in node i
$v = uniform(r)$ ▷ mutated value v
$s' \leftarrow s_i$ is replaced with $binary(v)$ ▷ for s, s_i the fragment of binary coding for mutation
Output individual s'



A Genetic Algorithm with Tree-structured Mutation for Hyperparameter Optimisation of Graph Neural Networks. arXiv 2021.

AutoGM

- A unified framework for graph learning algorithms

$$
\begin{aligned}
A_{\text{samp}} &= Sample(A) \\
A_{\text{agg}} &= Aggregate(A_{\text{samp}}) \\
X_k &= f_k(X_{k-1}) = \phi(A_{\text{agg}} X_{k-1} W_k) \\
&= f_k(f_{k-1}(\ldots f_1(X_0)))
\end{aligned}
$$

- Five hyper-parameters:.
  - Dimension d: The dimensionality of the messages
  - Length k: The size of neighborhood
  - Width w: The number of message passing steps.
  - Non-linearity l: The nonlinearity in the message passing
  - Aggregation strategy a: How to aggregate messages
- Algorithm: Bayesian optimizatopn

Spoiler: very similar to search space of graph NAS (using HPO methods)

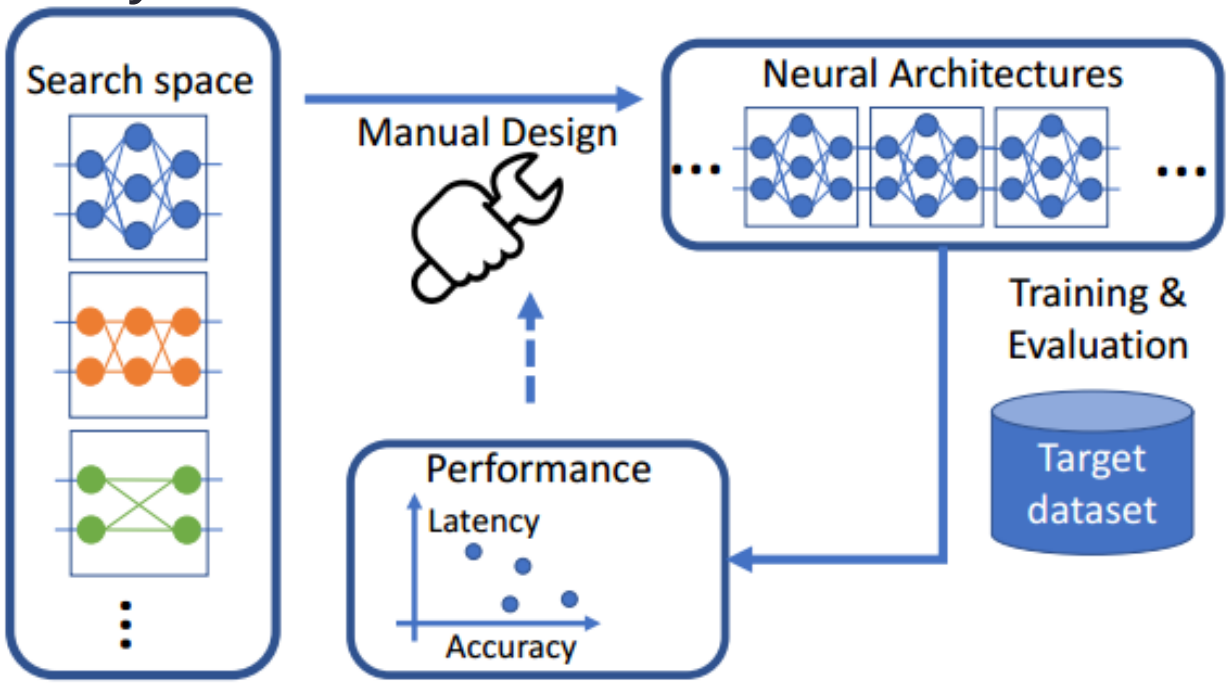
Autonomous Graph Mining Algorithm Search with Best Speed/Accuracy Trade-off. CIKM 2020.

Outline

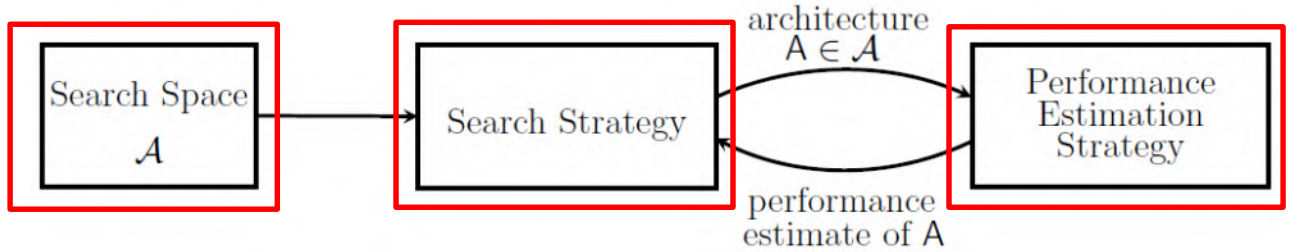
- Graph Hyper-parameter Optimization
- Graph Neural Architecture Search
- Automated Graph Learning Libraries

Neural Architecture Search (NAS)

- Goal: automatically learn the best neural architecture



- Categorization



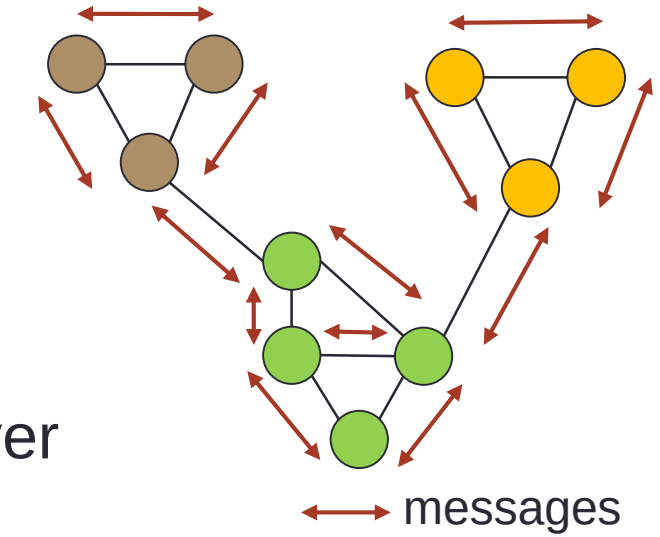
FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search, CVPR 2019
Neural Architecture Search A Survey, JMLR 2019

Graph NAS Search Space: Message-passing Framework

- Message-passing framework of GNNs

$$\mathbf{m}_i^{(l)} = \mathrm{AGG}^{(l)}\left(\left\{a_{ij}^{(l)}\mathbf{W}^{(l)}\mathbf{h}_i^{(l)}, \forall j \in \mathcal{N}(i)\right\}\right)$$

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathrm{COMBINE}^{(l)}\left[\mathbf{m}_i^{(l)}, \mathbf{h}_i^{(l)}\right]\right),$$



- $\mathrm{h}_i^{(l)}$: the representation of node v_i in the l^{th} layer
- $\mathrm{m}_i^{(l)}$: the received message of node v_i in the l^{th} layer

- All these choices can be searched

Neural message passing for quantum chemistry. *ICML, 2017.*

Graph NAS Search Space: Micro

$$\mathbf{m}_i^{(l)} = \mathrm{AGG}^{(l)}\left(\left\{a_{ij}^{(l)}\mathbf{W}^{(l)}\mathbf{h}_i^{(l)}, \forall j \in \mathcal{N}(i)\right\}\right)$$

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathrm{COMBINE}^{(l)}\left[\mathbf{m}_i^{(l)}, \mathbf{h}_i^{(l)}\right]\right),$$

- Micro search space:
  - Aggregation function $\mathrm{AGG}(\cdot)$: how to aggregate information from neighbors
    - Requirement: not depending on orders (i.e., neighbors are regarded a set instead of a sequence)
    - Common choices: mean, max, sum, etc.
  - Aggregation weights a_{ij}: the importance of different neighbors
  - Combining function $\mathrm{COMBINE}(\cdot)$: how to update representation
    - Common choices: CONCAT, SUM, MLP, etc.
  - Non-linearity $\sigma(\cdot)$: Sigmoid, ReLU, tanh, etc.

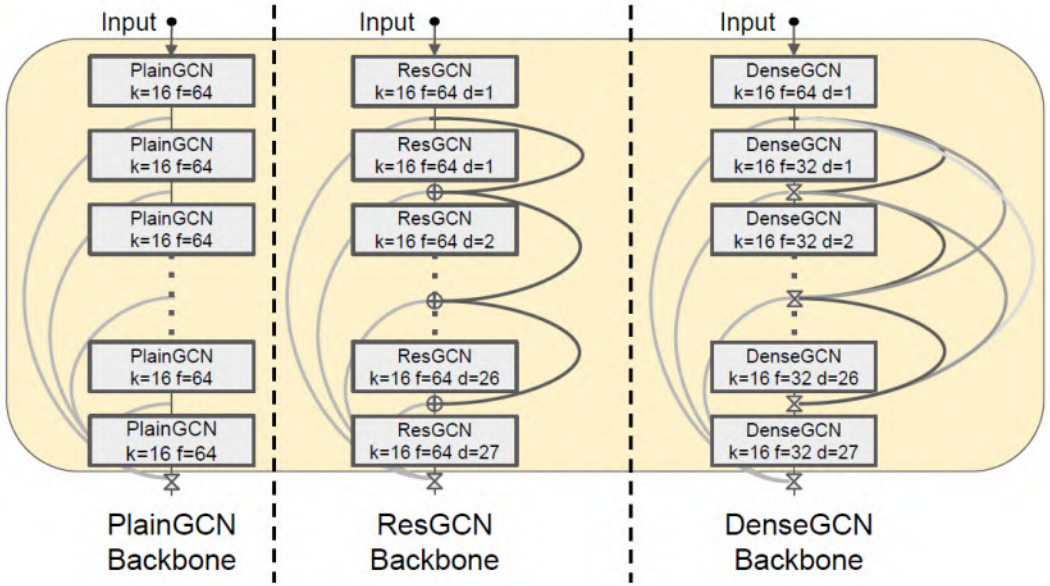
Type	Formulation
CONST	$a_{ij}^{\text{const}} = 1$
GCN	$a_{ij}^{\text{gcn}} = \frac{1}{\sqrt{\lvert\mathcal{N}(i)\rvert\lvert\mathcal{N}(j)\rvert}}$
GAT	$a_{ij}^{\text{gat}} = \text{LeakyReLU}\left(\text{ATT}\left(\mathbf{W}_a\left[\mathbf{h}_i, \mathbf{h}_j\right]\right)\right)$
SYM-GAT	$a_{ij}^{\text{sym}} = a_{ij}^{\text{gat}} + a_{ji}^{\text{gat}}$
COS	$a_{ij}^{\text{cos}} = \cos\left(\mathbf{W}_a\mathbf{h}_i, \mathbf{W}_a\mathbf{h}_j\right)$
LINEAR	$a_{ij}^{\text{lin}} = \tanh\left(\text{sum}\left(\mathbf{W}_a\mathbf{h}_i + \mathbf{W}_a\mathbf{h}_j\right)\right)$
GENE-LINEAR	$a_{ij}^{\text{gene}} = \tanh\left(\text{sum}\left(\mathbf{W}_a\mathbf{h}_i + \mathbf{W}_a\mathbf{h}_j\right)\right)\mathbf{W}_a'$

- Dimensionality of $h_i^{(l)}$, the number of attention heads (when using attention)

Graph Neural Architecture Search, *IJCAI 2020.*

Graph NAS Search Space: Macro

- Macro search space: how to arrange different layers
  - Residual connection, dense connection, etc.



- Formulation:

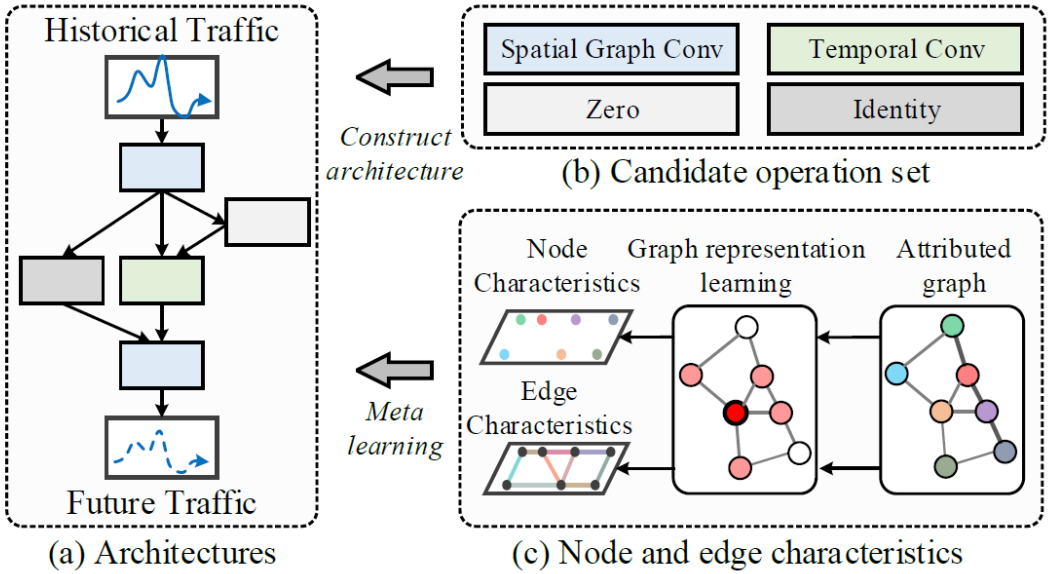
$$\mathbf{H}^{(l)} = \sum_{j<l} \mathcal{F}_{jl}\left(\mathbf{H}^{(j)}\right)$$

  - $\mathcal{F}_{jl}$: connectivity pattern from j^{th} to the l^{th} layer
    - ZERO (not connecting), IDENTITY (residual connection), MLP, etc.

DeepGCNs: Can GCNs Go as Deep as CNNs? *ICCV 2019*
Graph Neural Architecture Search, *IJCAI 2020.*

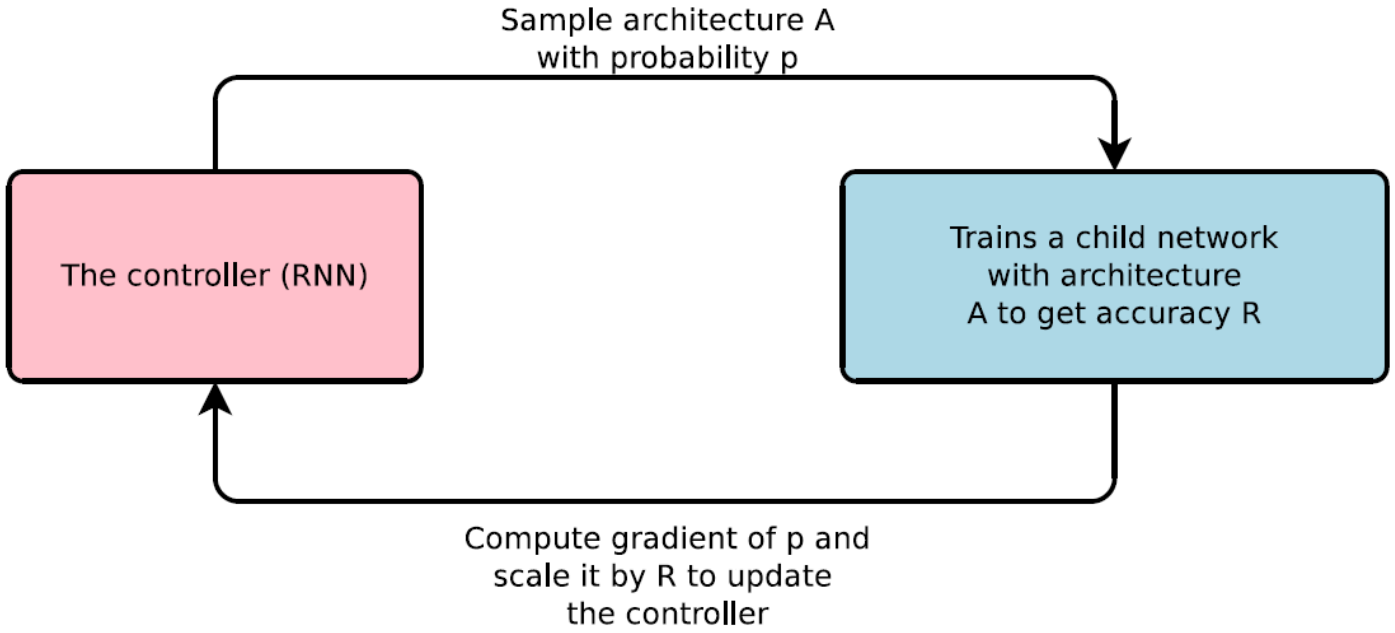
Graph NAS Search Space: Pooling

- Other search spaces
  - Pooling methods: $\mathbf{h}_{\mathcal{G}} = \mathrm{POOL}(\mathbf{H})$
    - Aggregate node-level representation into graph-level representation
  - Hyper-parameters: similar to HPO for graphs
    - Number of layers, number of epochs, optimizer, dropout rate, etc.
  - Spaces for specific tasks:
    - E.g., spatial-temporal graph operators



Graph NAS Search Strategy

- Most previous general NAS search strategies can be directly applied
  - Controller (e.g., RNN) + Reinforcement learning (RL)
  - Evolutionary
  - Differentiable



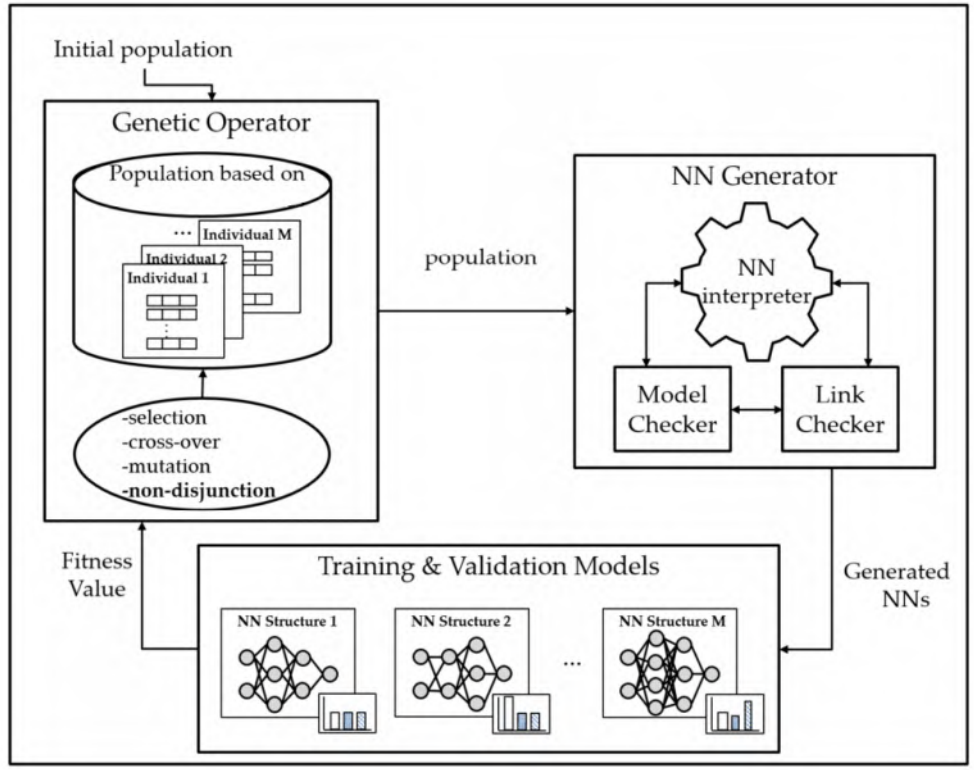
  - Controller samples architecture (e.g., as a sequence)
  - RL feedback rewards (e.g., validation performance) to update the controller

Neural Architecture Search with Reinforcement Learning, *ICLR 2017*

Graph NAS Search Strategy

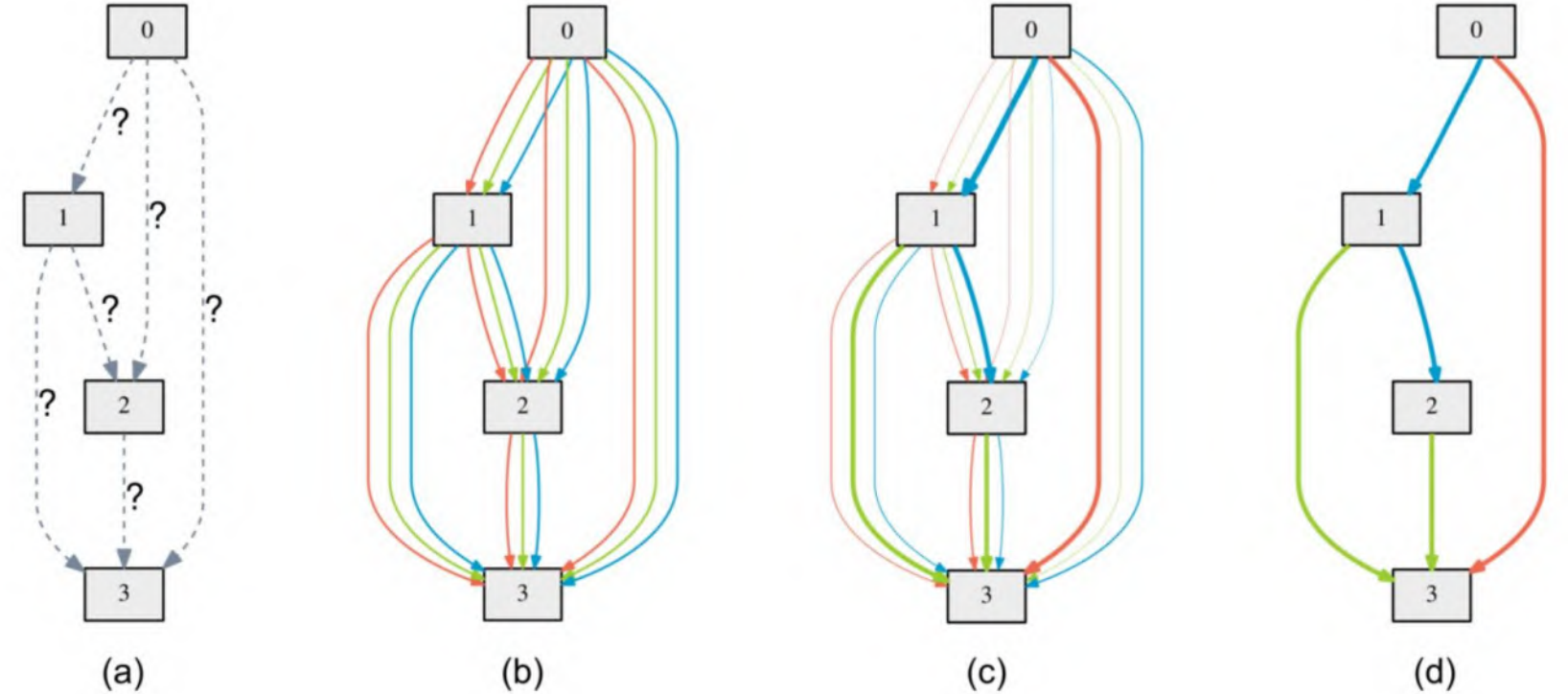
- Most previous general NAS search strategies can be directly applied
  - Controller (e.g., RNN) + Reinforcement learning (RL)
  - Evolutionary
  - Differentiable

- Need to define how to sample parents, generate offspring, and update populations
  - E.g., remove the worst individual (Real, et al., 2017), remove the oldest individual (Real, et al., 2018), or no remove (Liu, et al., 2018)



Graph NAS Search Strategy

- Most previous general NAS search strategies can be directly applied
  - Controller (e.g., RNN) + Reinforcement learning (RL)
  - Evolutionary
  - Differentiable



- Generate a super-network to combine operations of the search space
- Continuous relaxation to make the model differentiable

DARTS: Differentiable Architecture Search, *ICLR 2019*

Graph NAS Performance Estimation

- Low-fidelity training
  - Reduce number of epochs
  - Reduce training data: sample subgraphs as in HPO
- Inheriting weights
  - Challenge: parameters in graph ML (e.g., GNNs) are unlike other NNs
  - E.g., constraints by AGNN (Zhou et al., 2019)
    - Same weight shapes
    - Same attention and activation functions
- Weight sharing in differentiable NAS with one-shot model

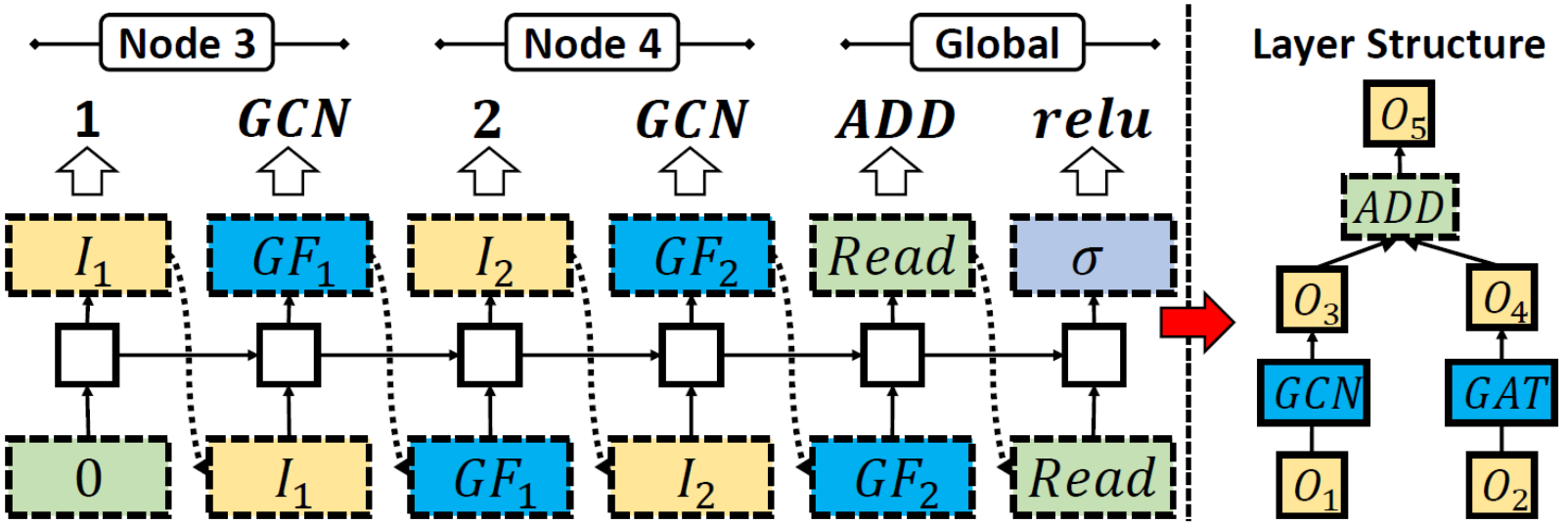
NAS for Graph Machine Learning

- Summary of NAS for graph ML

Method	Search space					Tasks		Search Strategy	Performance Estimation	Other Characteristics
	Micro	Macro	Pooling	HP	Layers	Node	Graph			
GraphNAS [2020]	✓	✓	✗	✗	Fixed	✓	✗	RNN controller + RL	-	-
AGNN [2019]	✓	✗	✗	✗	Fixed	✓	✗	Self-designed controller + RL	Inherit weights	-
SNAG [2020a]	✓	✓	✗	✗	Fixed	✓	✗	RNN controller + RL	Inherit weights	Simplify the micro search space
PDNAS [2020c]	✓	✓	✗	✗	Fixed	✓	✗	Differentiable	Single-path one-shot	-
POSE [2020]	✓	✓	✗	✗	Fixed	✓	✗	Differentiable	Single-path one-shot	Support heterogenous graphs
NAS-GNN [2020]	✓	✗	✗	✓	Fixed	✓	✗	Evolutionary algorithm	-	-
AutoGraph [2020]	✓	✓	✗	✗	Various	✓	✗	Evolutionary algorithm	-	-
GeneticGNN [2020b]	✓	✗	✗	✓	Fixed	✓	✗	Evolutionary algorithm	-	-
EGAN [2021a]	✓	✓	✗	✗	Fixed	✓	✓	Differentiable	One-shot	Sample small graphs for efficiency
NAS-GCN [2020]	✓	✓	✓	✗	Fixed	✗	✓	Evolutionary algorithm	-	Handle edge features
LPGNAS [2020b]	✓	✓	✗	✗	Fixed	✓	✗	Differentiable	Single-path one-shot	Search for quantisation options
You *et al.* [2020b]	✓	✓	✗	✓	Various	✓	✓	Random search	-	Transfer across datasets and tasks
SAGS [2020]	✓	✗	✗	✗	Fixed	✓	✓	Self-designed algorithm	-	-
Peng *et al.* [2020]	✓	✗	✗	✗	Fixed	✗	✓	CEM-RL [2019]	-	Search spatial-temporal modules
GNAS[2021]	✓	✓	✗	✗	Various	✓	✓	Differentiable	One-shot	-
AutoSTG[2021]	✗	✓	✗	✗	Fixed	✓	✗	Differentiable	One-shot+meta learning	Search spatial-temporal modules
DSS[2021]	✓	✓	✗	✗	Fixed	✓	✗	Differentiable	One-shot	Dynamically update search space
SANE[2021b]	✓	✓	✗	✗	Fixed	✓	✗	Differentiable	One-shot	-
AutoAttend[2021b]	✓	✓	✗	✗	Fixed	✓	✓	Evolutionary algorithm	One-shot	Cross-layer attention

Table 1: A summary of different NAS methods for graph machine learnings.

Graph NAS Example: GraphNAS



- Search space: micro + macro
- Search strategy: RNN controller + RL
  - GNN architecture description: a sequence of choices
- No parameter sharing

Graph Neural Architecture Search, *IJCAI 2020*

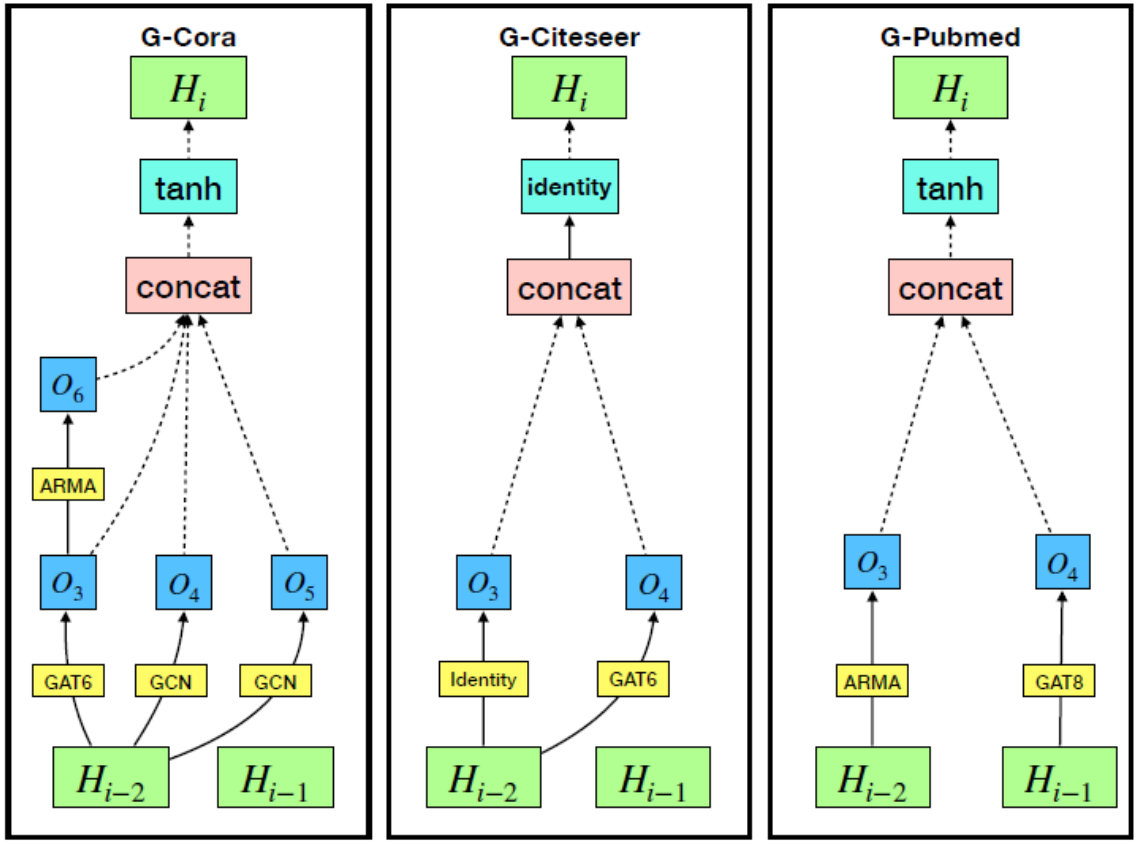
Graph NAS Example: GraphNAS

	Cora			Citeseer			Pubmed		
	semi	sup	rand	semi	sup	rand	semi	sup	rand
GCN	81.4±0.5	90.2±0.0	88.3±1.3	70.9±0.5	80.0±0.3	77.2±1.7	79.0±0.4	87.8±0.2	88.1±1.4
GAT	83.0±0.7	89.5±0.3	87.2±1.1	72.5±0.7	78.6±0.3	77.1±1.3	79.0±0.3	86.5±0.6	87.8±1.4
ARMA	82.8±0.6	89.8±0.1	88.2±1.0	72.3±1.1	79.9±0.6	76.7±1.5	78.8±0.3	88.1±0.2	88.7±1.0
APPNP	83.3±0.1	90.4±0.2	87.5±1.4	71.8±0.4	79.2±0.4	77.3±1.6	80.2±0.2	87.4±0.3	88.2±1.1
HGCN	79.8±1.2	89.7±0.4	87.7±1.1	70.0±1.3	79.2±0.5	76.9±1.3	78.4±0.6	88.0±0.5	88.0±1.6
GraphNAS-R	83.3±0.4	90.0±0.3	88.5±1.0	73.4±0.4	81.1±0.3	76.5±1.3	79.0±0.4	90.7±0.6	90.3±0.8
GraphNAS-S	81.4±0.6	90.1±0.3	88.5±1.0	71.7±0.6	79.6±0.5	77.5±2.3	79.5±0.5	88.5±0.2	88.5±1.1
GraphNAS	**83.7±0.4**	**90.6±0.3**	**88.9±1.2**	**73.5±0.3**	**81.2±0.5**	**77.6±1.5**	**80.5±0.3**	**91.2±0.3**	**91.1±1.0**

- Outperforms the existing manually designed architectures

Graph Neural Architecture Search, *IJCAI 2020*

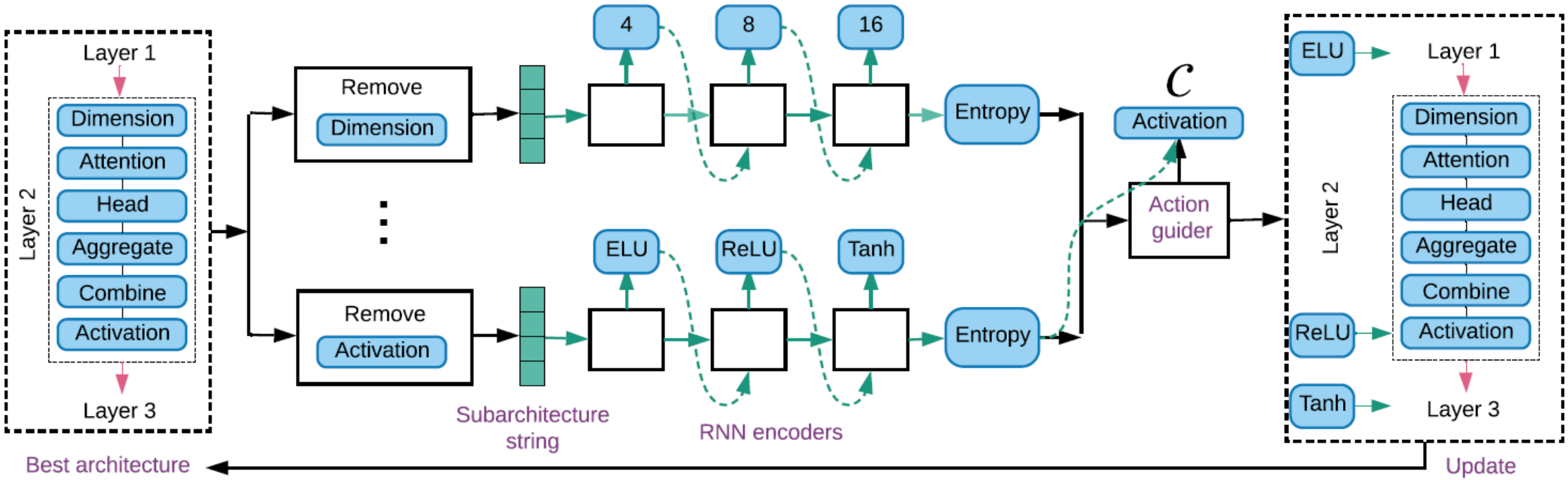
Graph NAS Example: GraphNAS



- The optimal architectures differ across different datasets

Graph Neural Architecture Search, *IJCAI 2020*

Graph NAS Example: AutoGNN

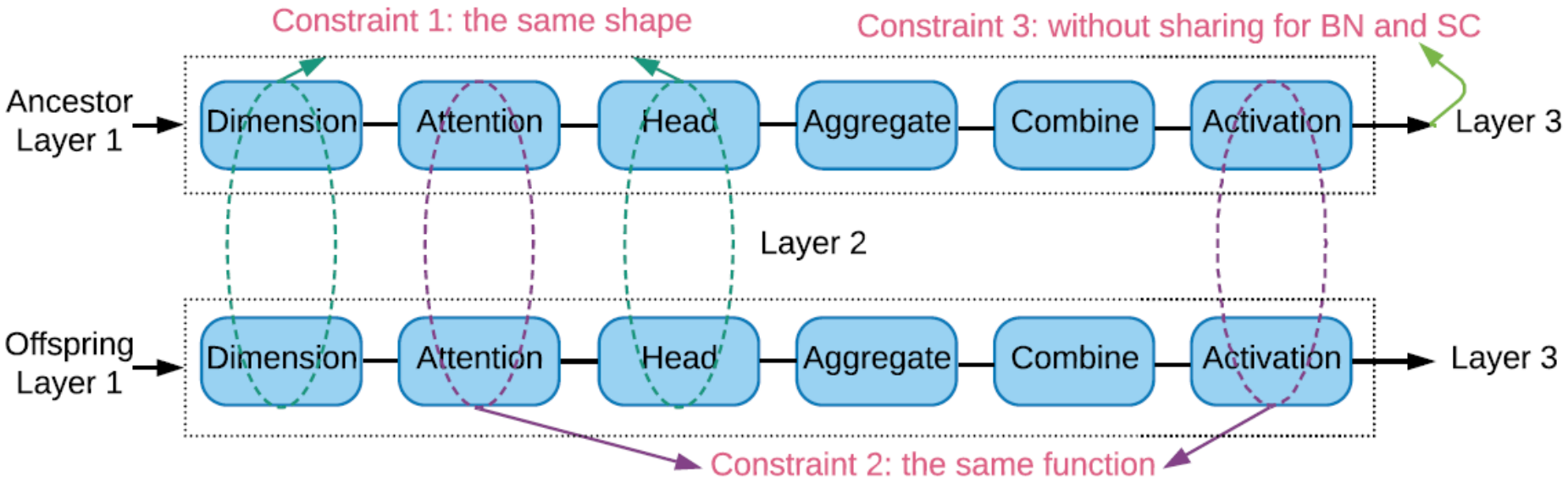


- Search space: micro
- Search strategy: reinforced conservative controller + RL
  - Conservative Explorer: maintain the best neural architecture found so far
  - Guided Architecture Modifier: modify the best architecture found so far via selecting and mutating the action classes

Auto-GNN Neural Architecture Search of Graph Neural Networks, *arXiv 2019*

Graph NAS Example: AutoGNN

- Performance estimation: constrained parameter sharing



- Three constraints:
  - Same tensor shapes
  - Same attention and activation functions
  - Skip connections and batch normalization do not share

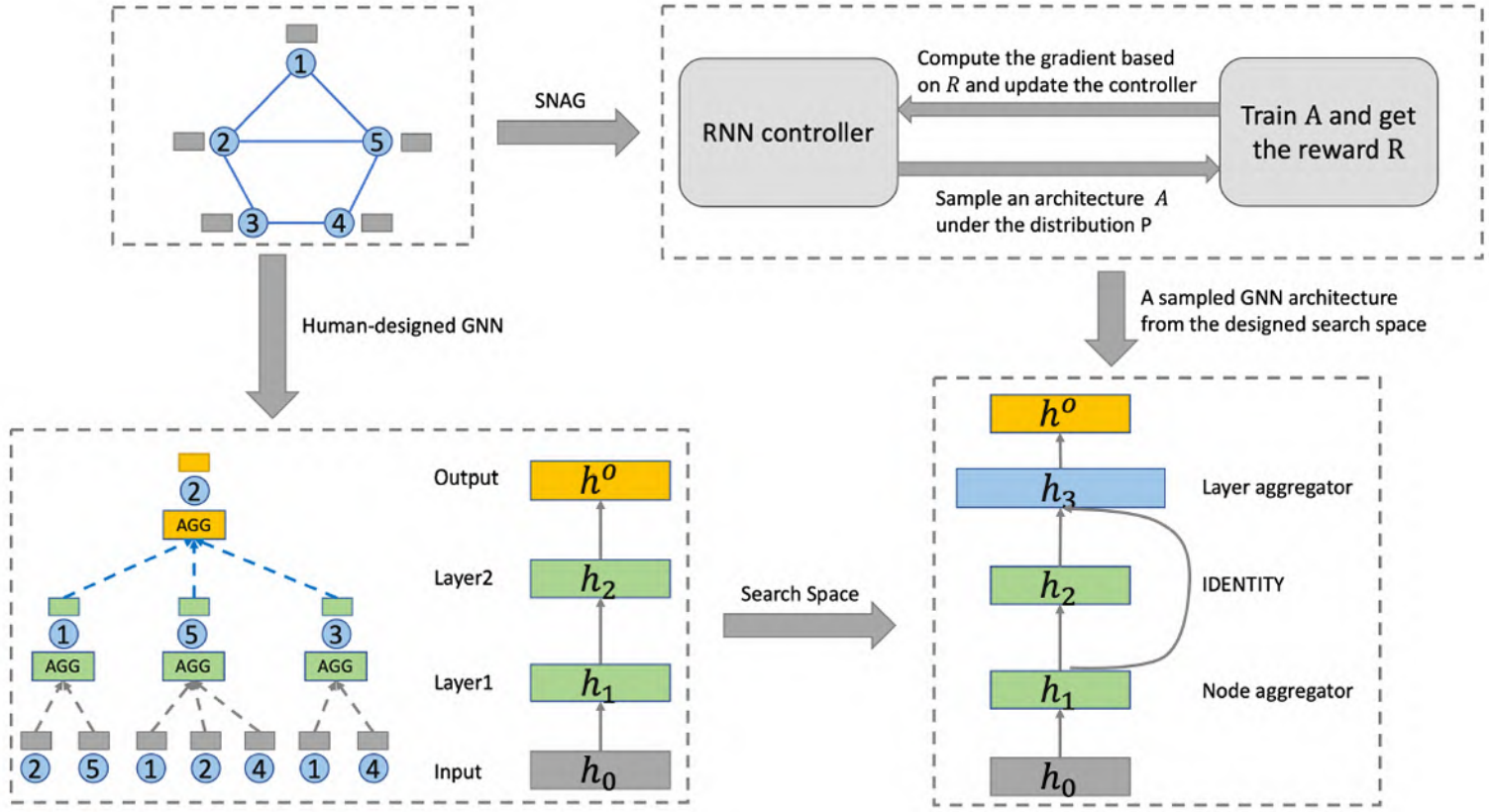
Auto-GNN Neural Architecture Search of Graph Neural Networks, *arXiv 2019*

Graph NAS Example: AutoGNN

Baseline Class	Model	#Layers	Cora		Citeseer		Pubmed	
			#Params	Accuracy	#Params	Accuracy	#Params	Accuracy
Handcrafted Architectures	Chebyshev	2	0.09M	81.2%	0.09M	69.8%	0.09M	74.4%
	GCN	2	0.02M	81.5%	0.05M	70.3%	0.02M	79.0.5%
	GAT	2	0.09M	83.0 ± 0.7%	0.23M	72.5 ± 0.7%	0.03M	79.0 ± 0.3%
	LGCN	3 ~ 4	0.06M	83.3 ± 0.5%	0.05M	73.0 ± 0.6%	0.05M	79.5 ± 0.2%
NAS Baselines	GraphNAS-w/o share	2	0.09M	82.7 ± 0.4%	0.23M	73.5 ± 1.0%	0.03M	78.8 ± 0.5%
	GraphNAS-with share	2	0.07M	83.3 ± 0.6%	1.91M	72.4 ± 1.3%	0.07M	78.1 ± 0.8%
	Random-w/o share	2	0.37M	81.4 ± 1.1%	0.95M	72.9 ± 0.2%	0.13M	77.9 ± 0.5%
	Random-with share	2	2.95M	82.3 ± 0.5%	0.95M	69.9 ± 1.7%	0.13M	77.9 ± 0.4%
AGNN	AGNN-w/o share	2	0.05M	**83.6 ± 0.3%**	0.71M	**73.8 ± 0.7%**	0.07M	**79.7 ± 0.4%**
	AGNN-with share	2	0.37M	82.7 ± 0.6%	1.90M	72.7 ± 0.4%	0.03M	79.0 ± 0.5%

Auto-GNN Neural Architecture Search of Graph Neural Networks, *arXiv 2019*

Graph NAS Example: SANG



- Search space: micro + macro
- Search strategy: RNN + RL

Simplifying Architecture Search for Graph Neural Network, *CIKM 2020 workshop*

Graph NAS Example: SANG

- A caution of search space: trade-off between effectiveness and efficiency
- Alternate the search space using domain knowledge and trial-and-errors

	Node aggregators	Layer aggregators	Others
GraphNAS/ Auto-GNN	GCN, SAGE-SUM/-MEAN/-MAX, MLP, GAT, GAT-SYM/-COS/ -LINEAR/-GEN-LINEAR,	-	Hidden Embedding Size, Attention Head, Activation Function
Ours	All above plus SAGE-LSTM and GeniePath	CONCAT, MAX, LSTM	IDENTITY, ZERO

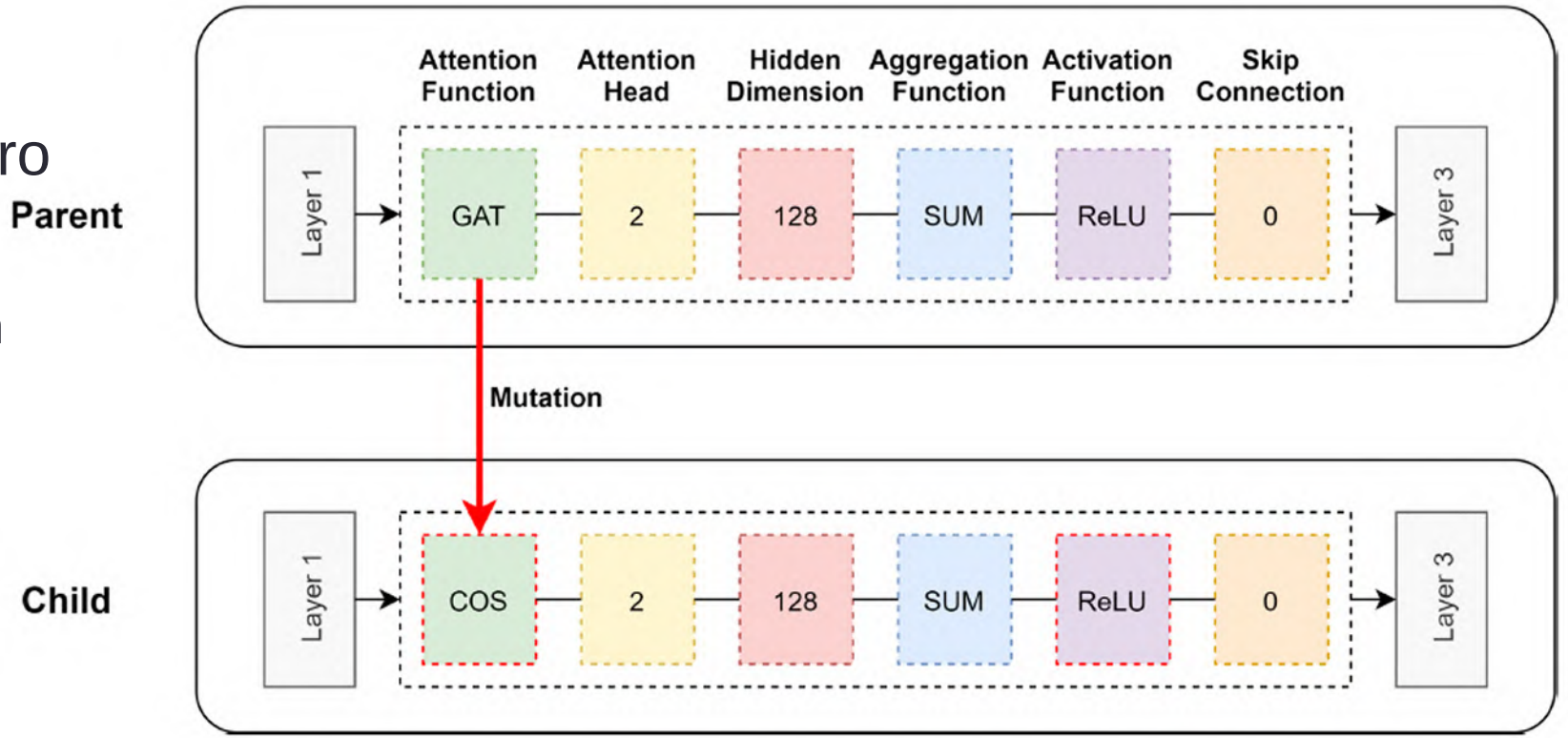
Simplifying Architecture Search for Graph Neural Network, *CIKM 2020 workshop*

Graph NAS Example: AutoGraph

- Previous RL based NAS: a fixed number of layers

- Search space: micro + macro
- Search strategy: evolution
  - A special “Layer Add” operation

  → A flexible number of layers



AutoGraph: Automated Graph Neural Network, *ICONIP 2020*

Graph NAS Example: AutoGraph

Table 2. Experiment results on Cora, Citeseer and Pubmed

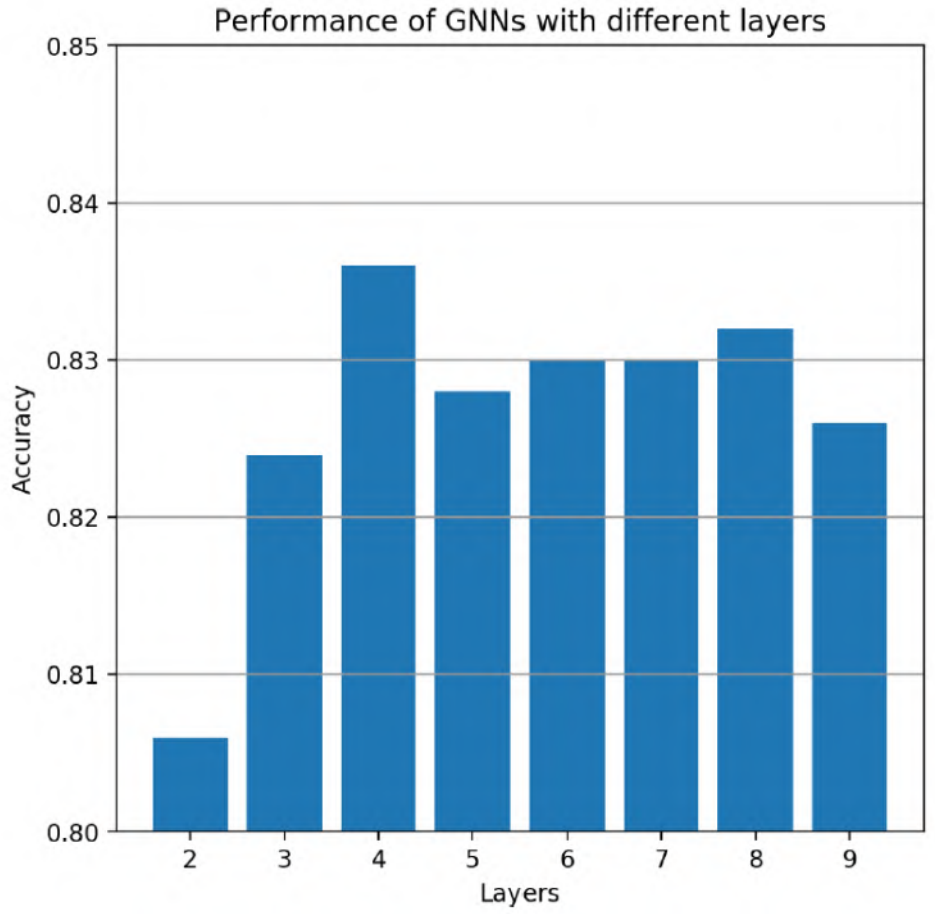
Models	Cora	Citeseer	Pubmed
Chebyshev	81.2%	69.8%	74.4%
GCN	81.5%	70.3%	79.0%
GAT	$83.0 \pm 0.7\%$	$72.5 \pm 0.7\%$	$79.0 \pm 0.3\%$
LGCN	$83.3 \pm 0.5\%$	$73.0 \pm 0.6\%$	$79.5 \pm 0.2\%$
GraphNAS	$83.3 \pm 0.6\%$	$73.5 \pm 1.0\%$	$78.8 \pm 0.5\%$
AutoGraph	$\mathbf{83.5 \pm 0.4\%}$	$\mathbf{74.4 \pm 0.4\%}$	$\mathbf{80.3 \pm 0.3\%}$

Table 3. Experiment results on PPI

Models	micro-F1
GraphSAGE (lstm)	0.612
GeniePath	0.979
GAT	0.973 ± 0.002
LGCN	0.772 ± 0.002
GraphNAS	0.985 ± 0.004
AutoGraph	$\mathbf{0.987 \pm 0.003}$

AutoGraph: Automated Graph Neural Network, *ICONIP 2020*

Graph NAS Example: AutoGraph



AutoGraph: Automated Graph Neural Network, *ICONIP 2020*

Graph NAS Example: NASGNN

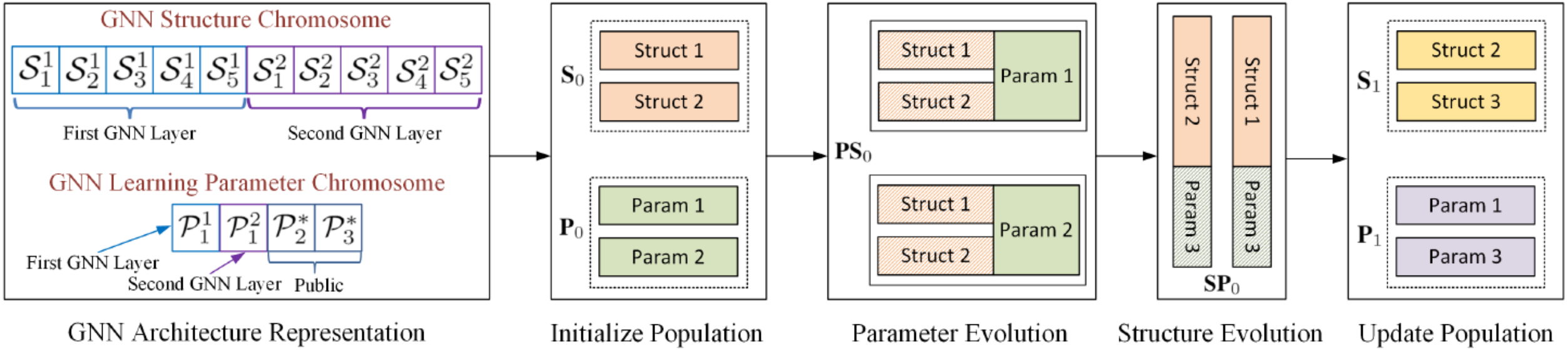
- Search space: micro + macro
- Search strategy: aging evolution (Real et al., AAAI 2019), RL, random search

		Macro		Micro	
		Accuracy	Time	Accuracy	Time
COR	EA	0.83 ± 0.007	0.75 ± 0.16	0.82 ± 0.005	1.73 ± 0.53
	RL	0.83 ± 0.003	1.45 ± 0.38	0.81 ± 0.001	2.42 ± 0.62
	RS	0.82 ± 0.003	0.96 ± 0.02	0.80 ± 0.009	1.20 ± 0.21
CIT	EA	0.75 ± 0.002	1.18 ± 0.10	0.71 ± 0.007	2.80 ± 0.72
	RL	0.73 ± 0.004	1.52 ± 0.42	0.68 ± 0.006	2.24 ± 0.08
	RS	0.73 ± 0.005	1.05 ± 0.03	0.69 ± 0.006	1.29 ± 0.04
MED	EA	0.82 ± 0.003	1.40 ± 0.37	0.82 ± 0.009	1.40 ± 0.09
	RL	0.80 ± 0.003	2.10 ± 0.14	0.76 ± 0.017	2.58 ± 0.28
	RS	0.85 ± 0.045	1.31 ± 0.02	0.80 ± 0.009	1.10 ± 0.18
CS	EA	0.98 ± 0.001	3.35 ± 0.78	0.99 ± 0.002	2.65 ± 0.48
	RL	0.95 ± 0.001	3.13 ± 0.11	0.97 ± 0.002	2.90 ± 0.34
	RS	0.97 ± 0.001	1.50 ± 0.03	0.99 ± 0.001	1.58 ± 0.05
PHY	EA	0.99 ± 0.002	4.21 ± 0.85	0.99 ± 0.000	1.53 ± 0.15
	RL	0.98 ± 0.001	3.34 ± 0.27	0.98 ± 0.001	2.01 ± 0.19
	RS	0.98 ± 0.001	2.08 ± 0.07	0.99 ± 0.001	1.11 ± 0.05
CMP	EA	0.91 ± 0.005	3.09 ± 0.49	0.93 ± 0.004	4.02 ± 1.94
	RL	0.90 ± 0.010	3.43 ± 0.21	0.92 ± 0.008	3.68 ± 0.27
	RS	0.89 ± 0.004	1.69 ± 0.07	0.92 ± 0.002	2.05 ± 0.07
PHO	EA	0.97 ± 0.002	2.48 ± 0.22	0.98 ± 0.004	1.66 ± 0.41
	RL	0.96 ± 0.005	3.65 ± 0.19	0.97 ± 0.002	1.88 ± 0.23
	RS	0.96 ± 0.002	1.82 ± 0.04	0.97 ± 0.002	1.08 ± 0.04

Neural Architecture Search in Graph Neural Networks, *BRACIS 2020*

Graph NAS Example: Genetic-GNN

- Search space: micro + hyper-parameters
- Search strategy: alternating evolution process between GNN architectures and learning hyper-parameter



Evolutionary Architecture Search for Graph Neural Networks, *arXiv 2020*

Graph NAS Example: Stacked MPNN

- Previous graph NAS: focus on node classification
- Molecular property prediction: graph-level tasks
- Search space: micro + macro + pooling
  - Choices: global pool, global gather, global attention pool, global attention sum pool, flatten
- Search strategy: regularized evolution
- Experiments:

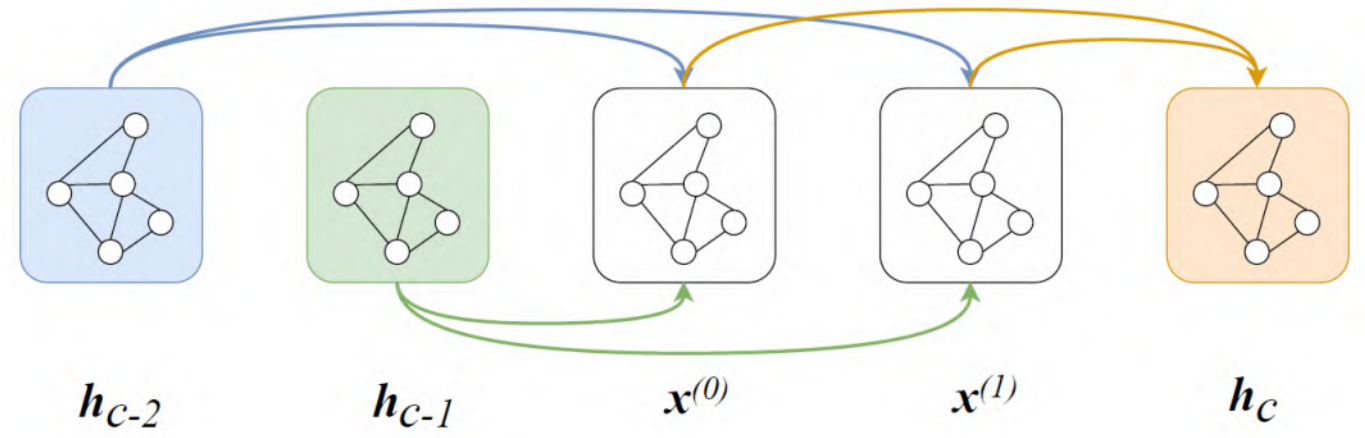
Data Set	Stacked MPNN	MoleculeNet GNN
QM7 (MAE)	**48.0±0.7**	77.9±2.1
ESOL (RMSE)	**0.54±0.01**	0.58±0.03
FreeSolv (RMSE)	1.21±0.03	**1.15±0.12**
Lipophilicity (RMSE)	**0.598±0.043**	0.715±0.035

QM8	Stacked MPNN	Molecule Net GNN	QM9	Stacked MPNN	Molecule Net GNN
E1-CC2	**0.0068±0.0003**	0.0084	mu	0.564±0.003	**0.358**
E2-CC2	**0.0079±0.0003**	0.0091	alpha	**0.69±0.01**	0.89
f1-CC2	**0.0129±0.0002**	0.0151	HOMO	**0.00560±0.00004**	0.00541
f2-CC2	**0.0291±0.0005**	0.0314	LUMO	**0.00602±0.00002**	0.00623
E1-PBE0	**0.0064±0.0001**	0.0083	gap	**0.0080±0.0000**	0.0082
E2-PBE0	**0.0072±0.0001**	0.0086	gap	**0.0080±0.0000**	0.0082
f1-PBE0	**0.0104±0.0002**	0.0123	gap	**0.0080±0.0000**	0.0082
f2-PBE0	**0.0216±0.0005**	0.0236	gap	**0.0080±0.0000**	0.0082
E1-CAM	**0.0063±0.0001**	0.0079	R2	41.3±0.6	**28.5**
E2-CAM	**0.0069±0.0002**	0.0082	ZPVE	**0.00130±0.00014**	0.00216
f1-CAM	**0.0117±0.0002**	0.0134	ZPVE	**0.00130±0.00014**	0.00216
f2-CAM	**0.0236±0.0001**	0.0258	U0	**0.65±0.06**	2.05
			U	**0.62±0.05**	2.00
			H	**0.68±0.11**	2.02
			G	**0.66±0.05**	2.02
			Cv	**0.35±0.01**	0.42

Graph Neural Network Architecture Search for Molecular Property Prediction, *IEEE BigData 2020*

Graph NAS Example: DSS

- Search space: micro
- Search strategy: differentiable



$$\boldsymbol{I}^{(j)} = \sum o^{(i,j)}\left(x^{(i)}\right), 2 \leq j \leq N-2.$$

$$\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}_{i,j}} \frac{\exp\left(\boldsymbol{\alpha}_o^{(i,j)}\right)}{\sum_{o' \in \mathcal{O}_{i,j}} \exp\left(\boldsymbol{\alpha}_{o'}^{(i,j)}\right)} o(x).$$

One-shot Graph Neural Architecture Search with Dynamic Search Space, *AAAI 2021*

Graph NAS Example: DSS

- Dynamic search space: only the top-K operations are kept after an iteration
  - Basic idea: if an operation is ranked lower than the other in a subset, it is ranked lower in the universe.

Algorithm 1 Search with dynamic search space

Input: a set of operation candidates $\mathcal{O}$, the number of nodes N in a cell, the maximum size M of the candidates subset, the maximum epoch max_epoch of inner-loop to optimize a hyper-network, the number K of top candidates to be remained after each iteration

Output: architecture parameter $\boldsymbol{\alpha}$

Random sample $\mathcal{O}_{i,j} \in \mathcal{O}$ of size M
while $\mathcal{O} \neq \varnothing$ **do**
  Let $\mathcal{O} \leftarrow \mathcal{O} - \mathcal{O}_{i,j}$
  Random initialize $\boldsymbol{\alpha}^{(i,j)}$
  for $epoch \in \{1, \ldots, max\_epoch\}$ **do**
    Update weights $\boldsymbol{w}$ with $\mathcal{L}_{train}(\boldsymbol{w}, \boldsymbol{\alpha})$
    Update architecture $\boldsymbol{\alpha}$ with $\mathcal{L}_{valid}(\boldsymbol{w}, \boldsymbol{\alpha})$
  end for
  Select top K operations $\mathcal{O}^*_{i,j} \in \mathcal{O}_{i,j}$
  Random sample $\mathcal{O}'_{i,j} \in \mathcal{O}$ of size $M - K$
  Let $\mathcal{O}_{i,j} \leftarrow \mathcal{O}^*_{i,j} \cup \mathcal{O}'_{i,j}$
end while

One-shot Graph Neural Architecture Search with Dynamic Search Space, *AAAI 2021*

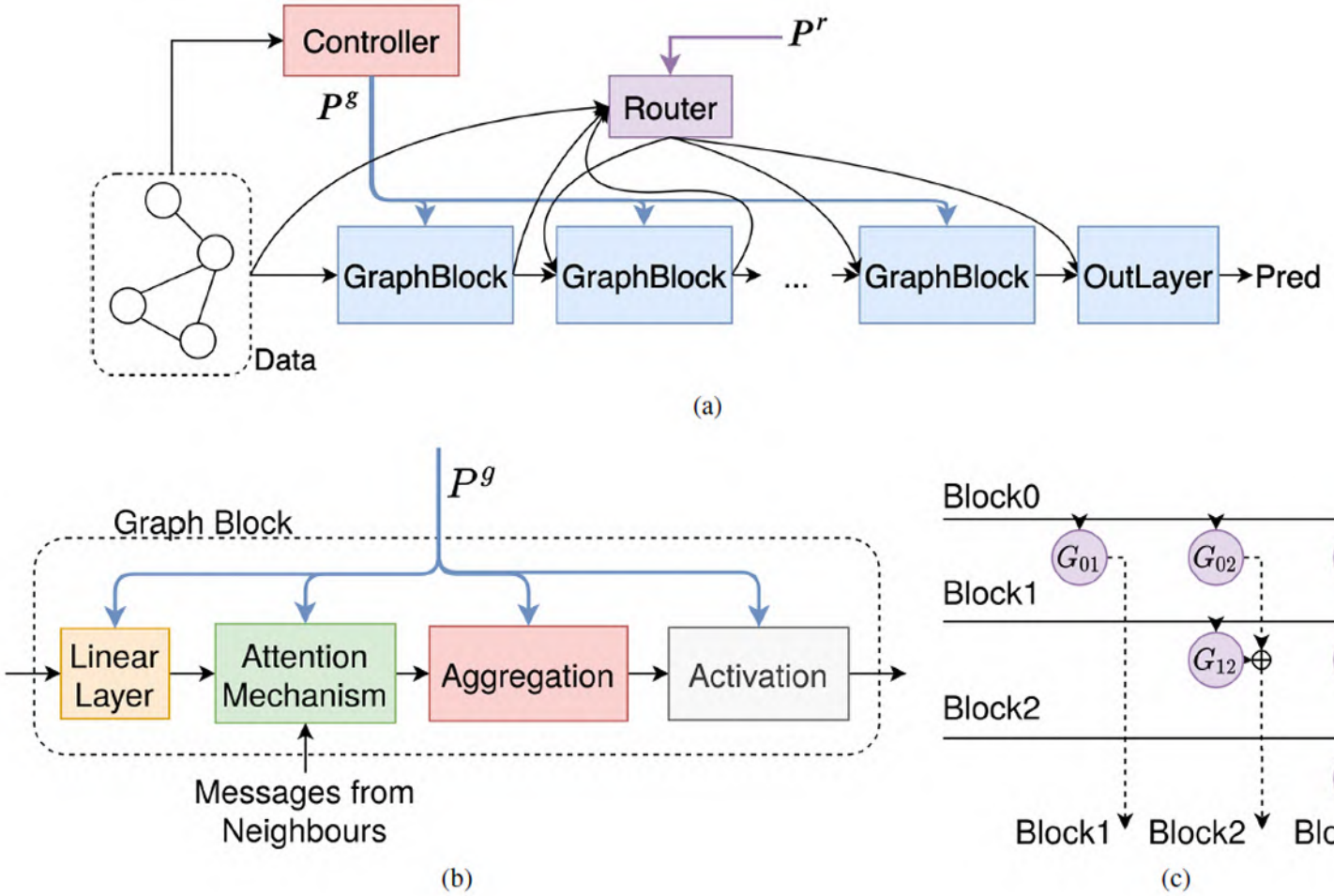
Graph NAS Example: DSS

Type	Model	Cora		Citeseer		Pubmed	
		semi	full	semi	full	semi	full
Manually Crafted	GCN (Kipf and Welling 2016)	$81.4 \pm 0.5\%$	$90.2 \pm 0.0\%$	$70.9 \pm 0.5\%$	$80.0 \pm 0.3\%$	$79.0 \pm 0.4\%$	$87.8 \pm 0.2\%$
	GAT (Veličković et al. 2017)	$83.0 \pm 0.7\%$	$89.5 \pm 0.3\%$	$72.5 \pm 0.7\%$	$78.6 \pm 0.3\%$	$79.0 \pm 0.3\%$	$86.5 \pm 0.6\%$
	ARMA (Bianchi et al. 2019)	$82.8 \pm 0.6\%$	$89.8 \pm 0.1\%$	$72.3 \pm 1.1\%$	$79.9 \pm 0.6\%$	$78.8 \pm 0.3\%$	$88.1 \pm 0.2\%$
	APPNP (Klicpera et al. 2018)	$83.3 \pm 0.1\%$	$90.4 \pm 0.2\%$	$71.8 \pm 0.4\%$	$79.2 \pm 0.4\%$	$80.2 \pm 0.2\%$	$87.4 \pm 0.3\%$
	H-GCN (Hu et al. 2019)	$79.8 \pm 1.2\%$	$89.7 \pm 0.4\%$	$70.0 \pm 1.3\%$	$79.2 \pm 0.5\%$	$78.4 \pm 0.6\%$	$88.0 \pm 0.5\%$
Macro NAS	AGNN (Zhou et al. 2019)	$83.6 \pm 0.3\%$	-	$73.8 \pm 0.7\%$	-	$79.7 \pm 0.4\%$	-
	GraphNAS (Gao et al. 2020)	$83.7 \pm 0.4\%$ (2 GPU Hrs)	-	$73.5 \pm 0.3\%$ (2 GPU Hrs)	-	$80.5 \pm 0.3\%$ (9 GPU Hrs)	-
Micro NAS	GraphNAS (Gao et al. 2020)	-	$90.6 \pm 0.3\%$ (6 GPU Hrs)	-	$81.2 \pm 0.5\%$ (6 GPU Hrs)	-	$91.2 \pm 0.3\%$ (12 GPU Hrs)
	DSS (Ours)	$83.9 \pm 0.3\%$	$91.0 \pm 0.2\%$ (0.9 GPU Hrs)	$73.3 \pm 0.3\%$	$81.4 \pm 0.4\%$ (0.8 GPU Hrs)	$80.3 \pm 0.2\%$	$88.2 \pm 0.4\%$ (0.9 GPU Hrs)

- Competitive performance to existing GNN NAS approaches with up to 10x speedup

One-shot Graph Neural Architecture Search with Dynamic Search Space, *AAAI 2021*

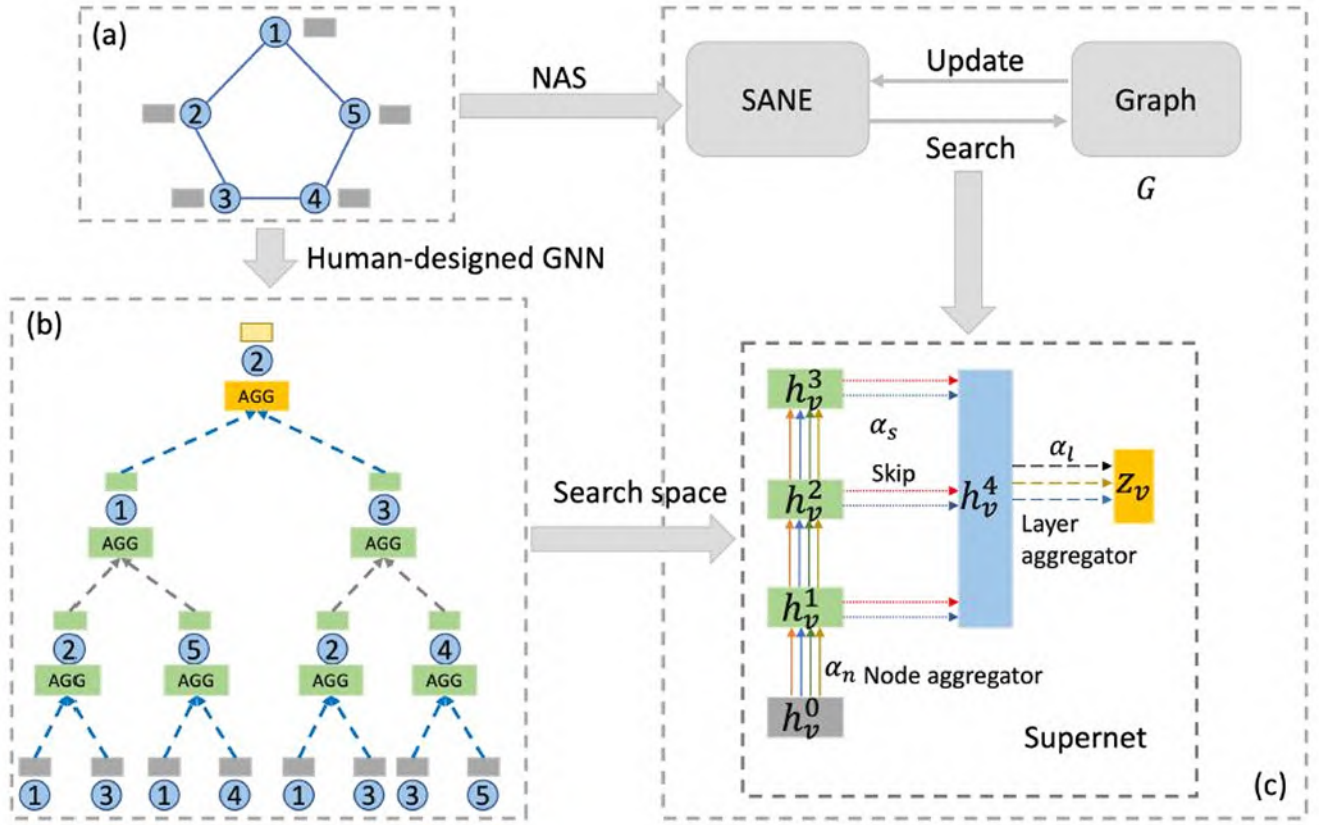
Graph NAS Example: PDNAS



- Search space: micro + macro
- Search algorithm: differentiable

Probabilistic Dual Network Architecture Search on Graphs, *arXiv 2021*

Graph NAS Example: EGAN



- Search space: micro + macro
- Search strategy: one-shot differentiable
- Large-scale graphs: sample subgraphs as proxies (similar to AutoNE and eAutoGR)

Efficient Graph Neural Architecture Search, *OpenReview 2020*

Graph NAS Example: SANE

- Node aggregator: similar to the micro space
  - Search how to aggregate neighborhoods

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

- Layer aggregator: similar to the macro space
  - Search how to aggregate different layers

$$\mathbf{z}_v = \mathrm{AGG}_{\mathrm{layer}}(\mathbf{h}_v^1, \cdots, \mathbf{h}_v^K)$$

- Candidate operations

	Operations
$\mathcal{O}_n$	SAGE-SUM, SAGE-MEAN, SAGE-MAX, GCN, GAT,GAT-SYM, GAT-COS, GAT-LINEAR, GAT-GEN-LINEAR, GIN, GeniePath
$\mathcal{O}_l$	CONCAT, MAX, LSTM
$\mathcal{O}_s$	IDENTITY, ZERO

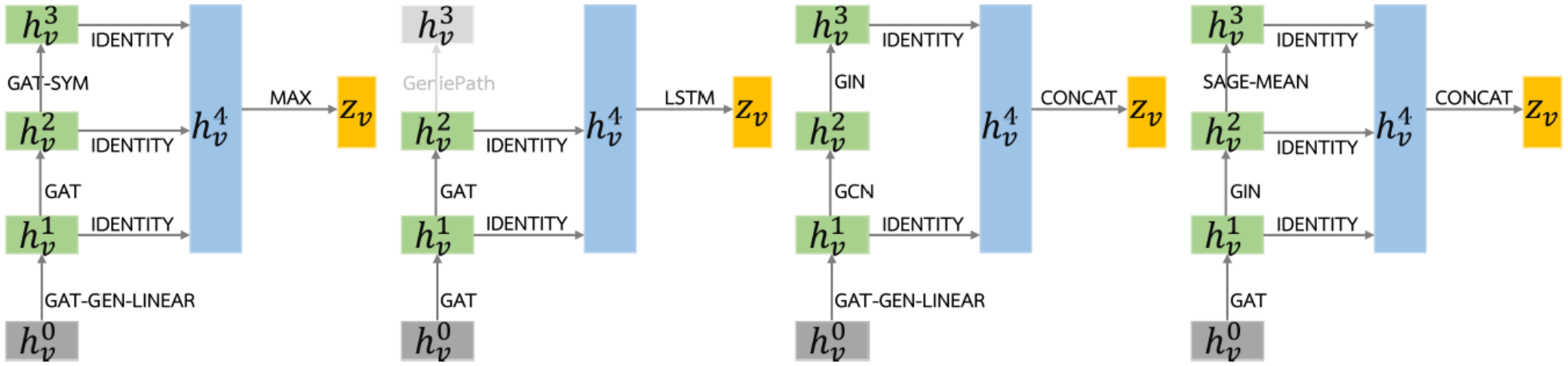
Search to aggregate neighborhood for graph neural network, *ICDE 2021*

Graph NAS Example: SANE

		Transductive			Inductive
	Methods	Cora	CiteSeer	PubMed	PPI
Human-designed architectures	GCN	0.8811 (0.0101)	0.7666 (0.0202)	0.8858 (0.0030)	0.6500 (0.0000)
	GCN-JK	0.8820 (0.0118)	0.7763 (0.0136)	0.8927 (0.0037)	0.8078(0.0000)
	GraphSAGE	0.8741 (0.0159)	0.7599 (0.0094)	0.8834 (0.0044)	0.6504 (0.0000)
	GraphSAGE-JK	0.8841 (0.0015)	0.7654 (0.0054)	0.8942 (0.0066)	0.8019 (0.0000)
	GAT	0.8719 (0.0163)	0.7518 (0.0145)	0.8573 (0.0066)	0.9414 (0.0000)
	GAT-JK	0.8726 (0.0086)	0.7527 (0.0128)	0.8674 (0.0055)	0.9749 (0.0000)
	GIN	0.8600 (0.0083)	0.7340 (0.0139)	0.8799 (0.0046)	0.8724 (0.0002)
	GIN-JK	0.8699 (0.0103)	0.7651 (0.0133)	0.8878 (0.0054)	0.9467 (0.0000)
	GeniePath	0.8670 (0.0123)	0.7594 (0.0137)	0.8846 (0.0039)	0.7138 (0.0000)
	GeniePath-JK	0.8776 (0.0117)	0.7591 (0.0116)	0.8868 (0.0037)	0.9694 (0.0000)
	LGCN	0.8687 (0.0075)	0.7543 (0.0221)	0.8753 (0.0012)	0.7720 (0.0020)
NAS approaches	Random	0.8594 (0.0072)	0.7062 (0.0042)	0.8866(0.0010)	0.9517 (0.0032)
	Bayesian	0.8835 (0.0072)	0.7335 (0.0006)	0.8801(0.0033)	0.9583 (0.0082)
	GraphNAS	0.8840 (0.0071)	0.7762 (0.0061)	0.8896 (0.0024)	0.9692 (0.0128)
	GraphNAS-WS	0.8808 (0.0101)	0.7613 (0.0156)	0.8842 (0.0103)	0.9584 (0.0415)
one-shot NAS	SANE	**0.8926 (0.0123)**	**0.7859 (0.0108)**	**0.9047 (0.0091)**	**0.9856 (0.0120)**

Search to aggregate neighborhood for graph neural network, *ICDE 2021*

Graph NAS Example: SANE

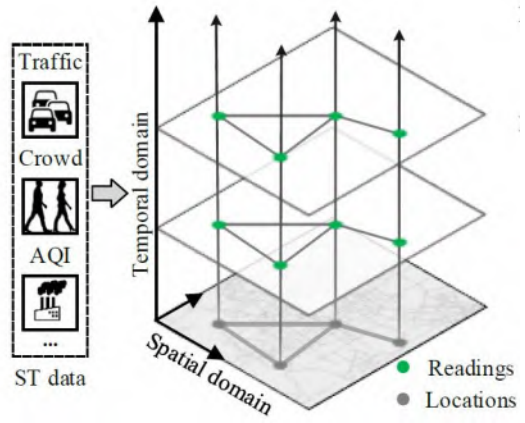


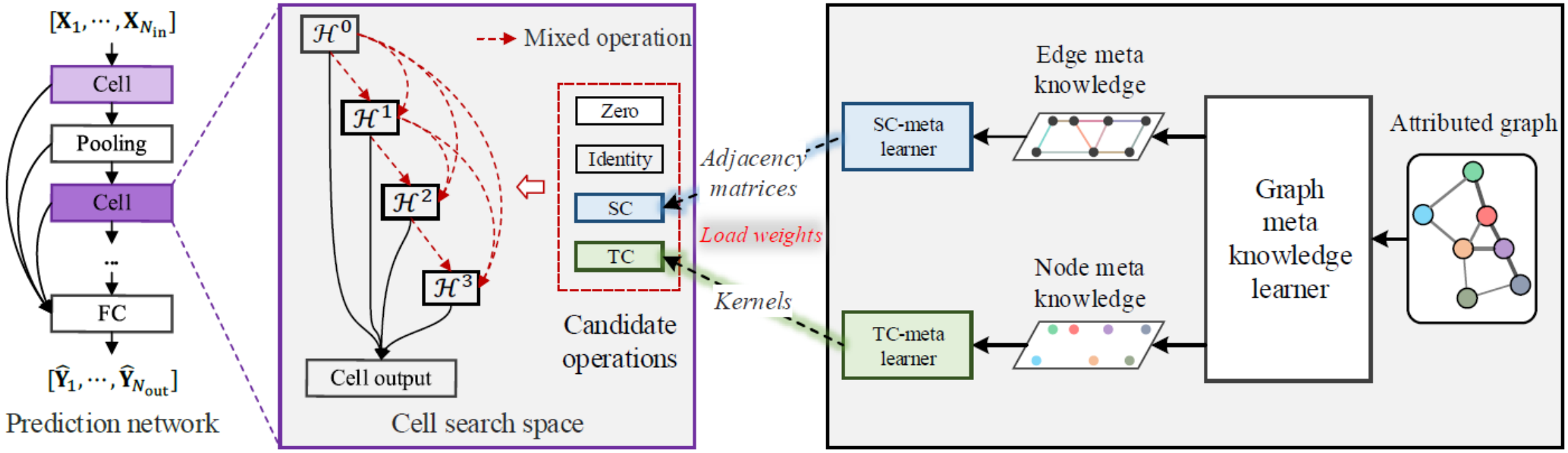
(a) Cora. (b) CiteSeer. (c) PubMed. (d) PPI.

Search to aggregate neighborhood for graph neural network, *ICDE 2021*

Graph NAS Example: AutoSTG

- Tasks: NAS for spatial-temporal graphs
  - Typical example: traffic prediction, time-series prediction, etc.





(a) Architecture search space

(b) Meta learning for network weight parameters

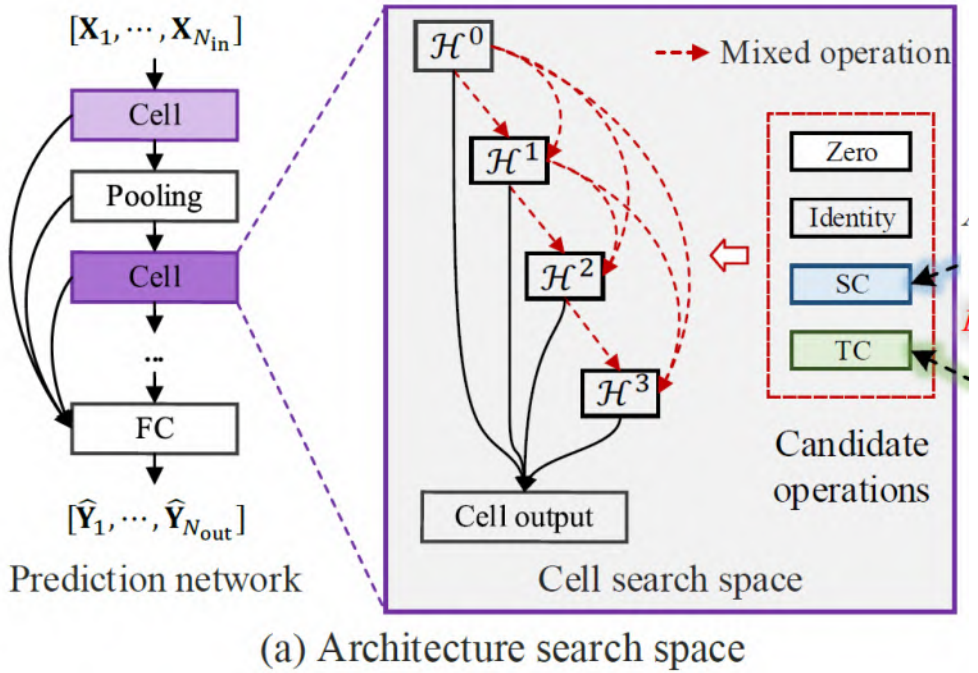
AutoSTG Neural Architecture Search for Predictions of Spatio-Temporal Graphs, *WWW 2021*

Graph NAS Example: AutoSTG

- Search space: spatial convolution (SC) and temporal convolution (TC)
  - Spatial convolution: diffusion convolution
  - Temporal convolution: $\mathbf{H}'_i = \mathbf{H}_i \star \mathcal{K}_i,$

$$\mathrm{DC}(\mathbf{H}, \mathcal{A}, \mathbb{W}) = \sum_{k=1}^{K} \sum_{p=1}^{P} (\mathbf{A}_k)^p \mathbf{H} \mathbf{W}_{kp},$$

  - Zero: not connection
  - Identity: for residual connections

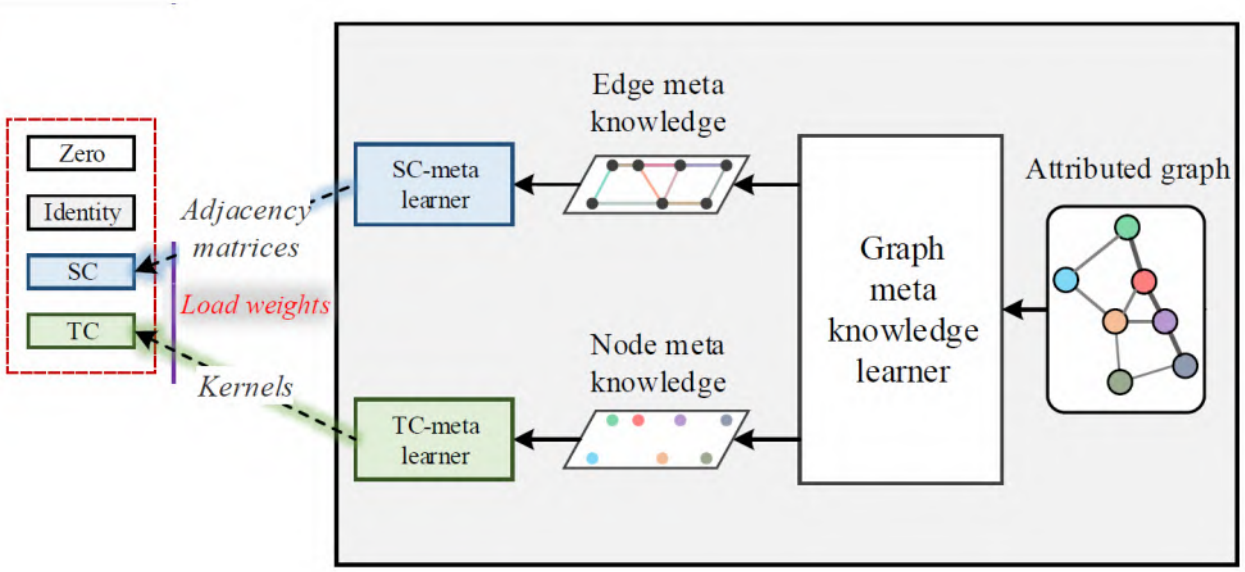


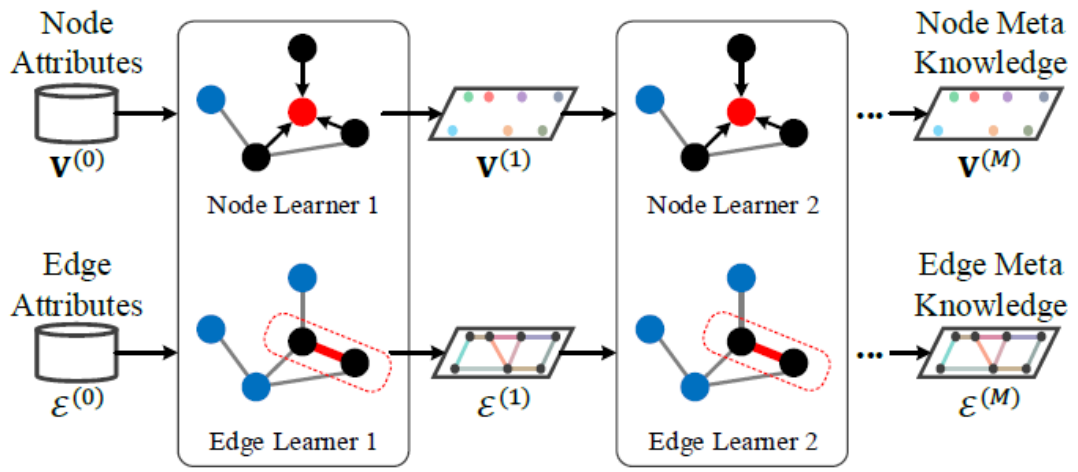
(a) Architecture search space

AutoSTG Neural Architecture Search for Predictions of Spatio-Temporal Graphs, *WWW 2021*

Graph NAS Example: AutoSTG

- Search strategy: differentiable + meta learning to generate weight parameters of SC and TC





- Graph Meta Knowledge Learner: improves upon a previous work (Pan et al., KDD 2019)

AutoSTG Neural Architecture Search for Predictions of Spatio-Temporal Graphs, *WWW 2021*

Graph NAS Example: AutoSTG

Table 3: Predictive performance on PEMS-BAY and METR-LA datasets.

	MAE (↓)				RMSE (↓)			
PEMS-BAY	Overall	15 min	30 min	60 min	Overall	15 min	30 min	60 min
HA	3.84±0.00	3.84±0.00	3.84±0.00	3.84±0.00	7.16±0.00	7.16±0.00	7.16±0.00	7.16±0.00
GBRT	1.96±0.02	1.49±0.01	1.99±0.02	2.61±0.04	4.48±0.00	3.21±0.00	4.51±0.01	5.76±0.02
GAT-Seq2Seq	1.74±0.00	1.38±0.01	1.79±0.00	2.26±0.01	4.08±0.01	2.94±0.01	4.10±0.01	5.22±0.04
DCRNN	1.59±0.00	1.31±0.00	1.65±0.01	1.97±0.00	3.70±0.02	2.76±0.01	3.78±0.02	4.60±0.02
Graph WaveNet	1.59±0.00	1.31±0.01	1.65±0.01	1.98±0.03	3.66±0.04	**2.75±0.01**	3.73±0.04	4.56±0.06
ST-MetaNet$^+$	1.60±0.01	1.31±0.00	1.66±0.06	1.99±0.01	3.72±0.02	2.78±0.01	3.81±0.01	4.62±0.04
AutoSTG	**1.56±0.01**	**1.31±0.00**	**1.63±0.01**	**1.92±0.01**	**3.57±0.02**	2.76±0.01	**3.67±0.02**	**4.38±0.03**
METR-LA	Overall	15 min	30 min	60 min	Overall	15 min	30 min	60 min
HA	4.79±0.00	4.79±0.00	4.79±0.00	4.79±0.00	8.72±0.00	8.72±0.00	8.72±0.00	8.72±0.00
GBRT	3.86±0.01	3.16±0.00	3.85±0.00	4.86±0.01	7.49±0.01	6.05±0.00	7.50±0.00	9.10±0.02
GAT-Seq2Seq	3.28±0.00	2.83±0.01	3.31±0.00	3.93±0.01	6.66±0.01	5.47±0.01	6.68±0.00	8.03±0.02
DCRNN	3.04±0.01	2.67±0.00	3.08±0.01	3.56±0.01	6.27±0.03	5.18±0.01	6.20±0.03	7.53±0.04
Graph WaveNet	3.05±0.01	2.70±0.01	3.08±0.01	3.55±0.12	6.16±0.03	5.16±0.01	6.20±0.03	7.35±0.05
ST-MetaNet$^+$	**3.00±0.01**	**2.65±0.01**	**3.04±0.01**	3.48±0.02	6.16±0.02	**5.11±0.01**	**6.16±0.02**	7.37±0.04
AutoSTG	3.02±0.00	2.70±0.01	3.06±0.00	**3.47±0.01**	**6.10±0.01**	5.16±0.01	6.17±0.01	**7.27±0.01**

AutoSTG Neural Architecture Search for Predictions of Spatio-Temporal Graphs, *WWW 2021*

Graph NAS Example: AutoSTG

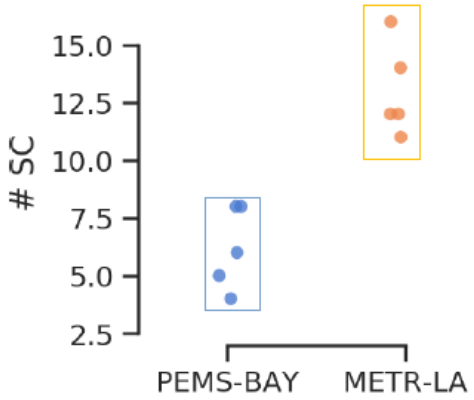
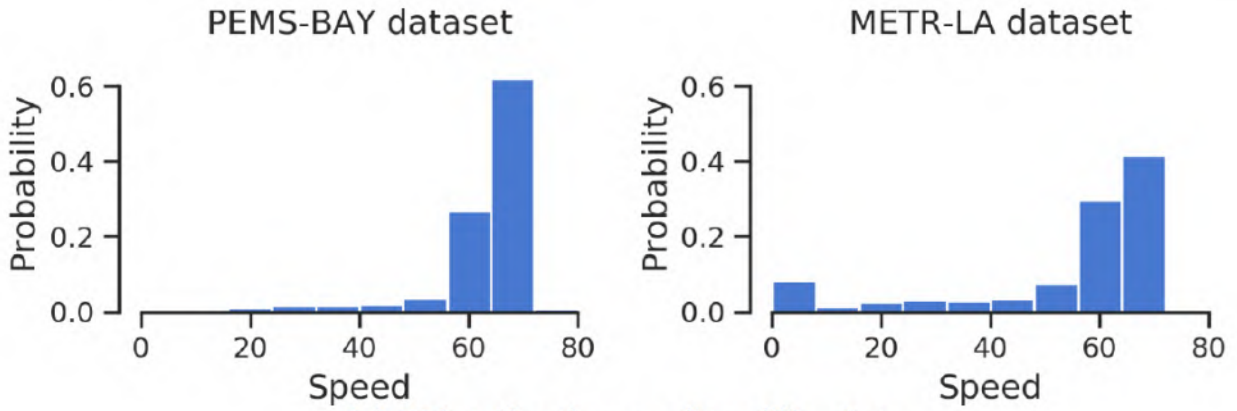
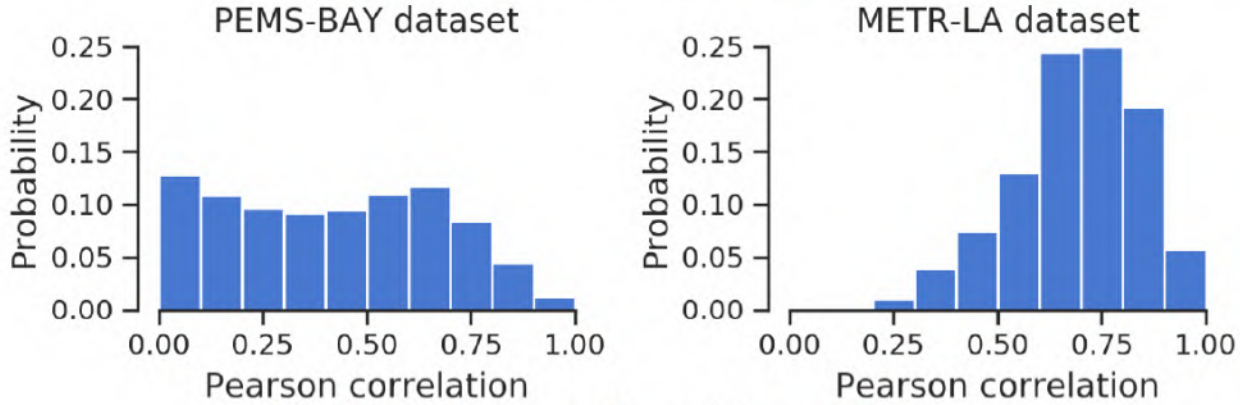


Figure 10: The number of SC in the learned architectures.



(a) Distributions of traffic data



(b) Distributions of spatial correlations

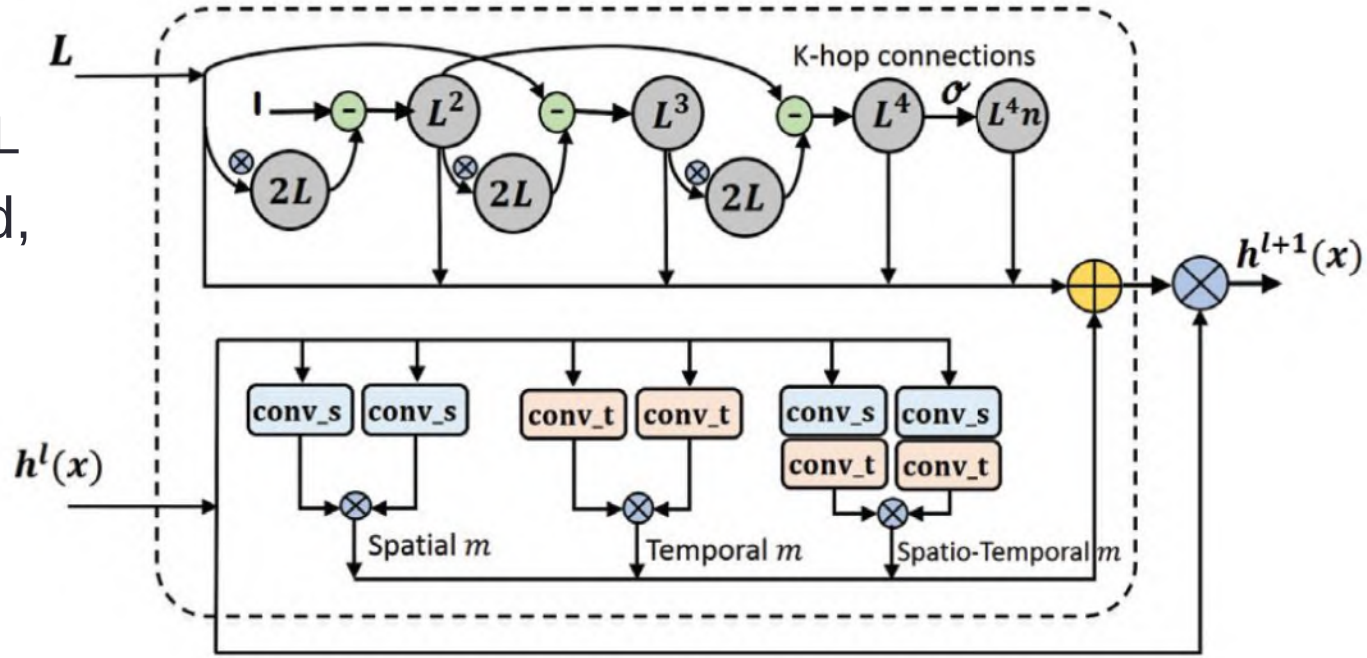
AutoSTG Neural Architecture Search for Predictions of Spatio-Temporal Graphs, *WWW 2021*

Graph NAS Example: Skeleton-based Action Recognition

- GNNs are widely used in skeleton-based action recognition
  - However, all the existing methods are manually designed
  - A general framework

$$\forall i,j \in \mathcal{V}, A_D(i,j) = \frac{e^{\phi(h(x_i)) \otimes \psi(h(x_j))}}{\sum_{j=1}^{n} e^{\phi(h(x_i)) \otimes \psi(h(x_j))}}$$

- Search space:
  - Spatial convolution: ϕ, Ψ as channel-wise convolution filters
  - Temporal convolution: ϕ, Ψ as temporal convolution filters
- Search algorithm: modified from CEM-RL (Pourchot and Sigaud, ICLR 2019)



Learning Graph Convolutional Network for Skeleton-based Human Action Recognition by Neural Searching, *AAAI 2020*

Graph NAS Example: GNAS

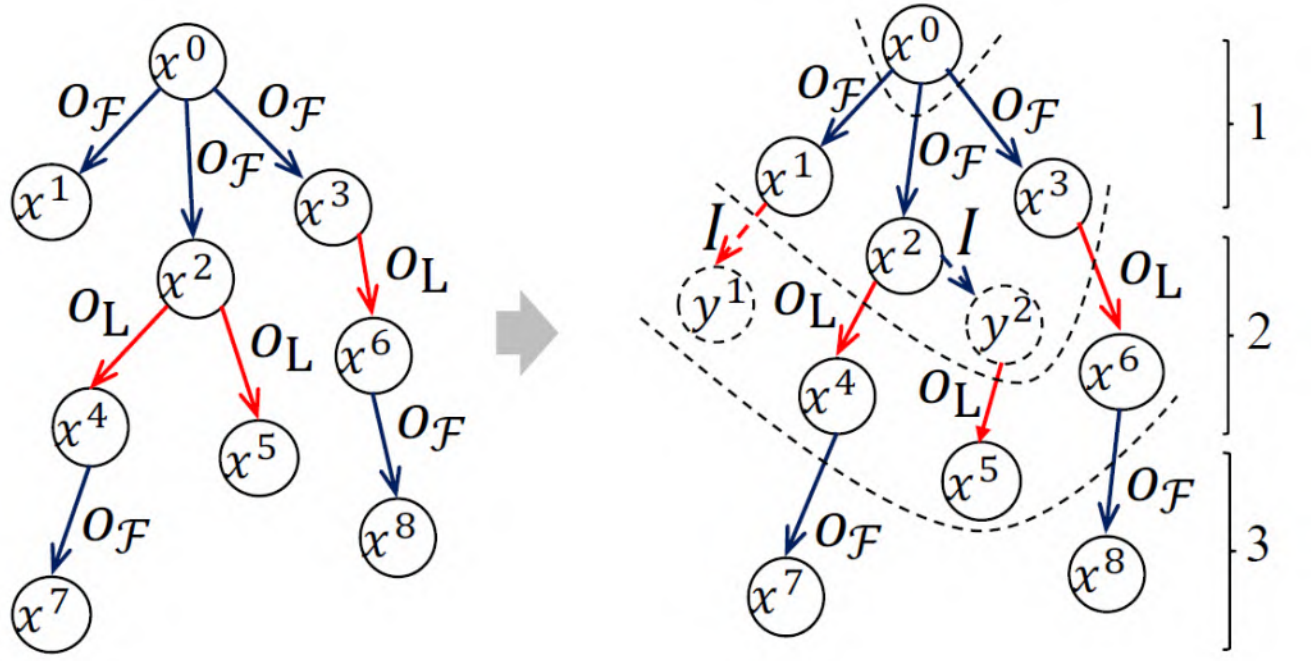
- A new GNN paradigm: feature filtering + neighbor aggregation
  - Feature filtering: gating mechanism to control the information flow
    - Sparse filter: $\mathcal{F}_s(H) = QH, Q = diag\left(\mathcal{M}_Q([H, H_{in}])\right)$
    - Dense filter: $\mathcal{F}_d(H) = Z \odot H, Z = \mathcal{M}_Z([H, H_{in}])$
    - Identity filter: $\mathcal{I}(H) = H$
  - Neighborhood aggregation: mean, max, sum

GNNs	Approximation Formula
GCN	$\mathbf{H}_{out} \approx \mathcal{M}(L_{mean}(\mathbf{H}_{in}))$
GIN	$\mathbf{H}_{out} \approx \mathcal{M}([\mathcal{I}(\mathbf{H}_{in})\|\|\mathcal{F}_s(\mathbf{H}_{in})\|\|L_{sum}(\mathbf{H}_{in})])$
GraphSage	$\mathbf{H}_{out} \approx \mathcal{M}([\mathcal{I}(\mathbf{H}_{in}) \parallel L_{mean}(\mathbf{H}_{in})])$
GAT	$\mathbf{H}_{out} \approx \mathcal{M}(\mathcal{F}_s(L_{sum}(\mathcal{F}_s(\mathbf{H}_{in}))))$
GatedGCN	$\mathbf{H}_{out} \approx \mathcal{M}([\mathcal{I}(\mathbf{H}_{in}) \parallel \mathcal{F}_d(L_{sum}(\mathcal{F}_d(\mathbf{H}_{in})))])$

Rethinking Graph Neural Architecture Search from Message-passing, *CVPR 2021*

Graph NAS Example: GNAS

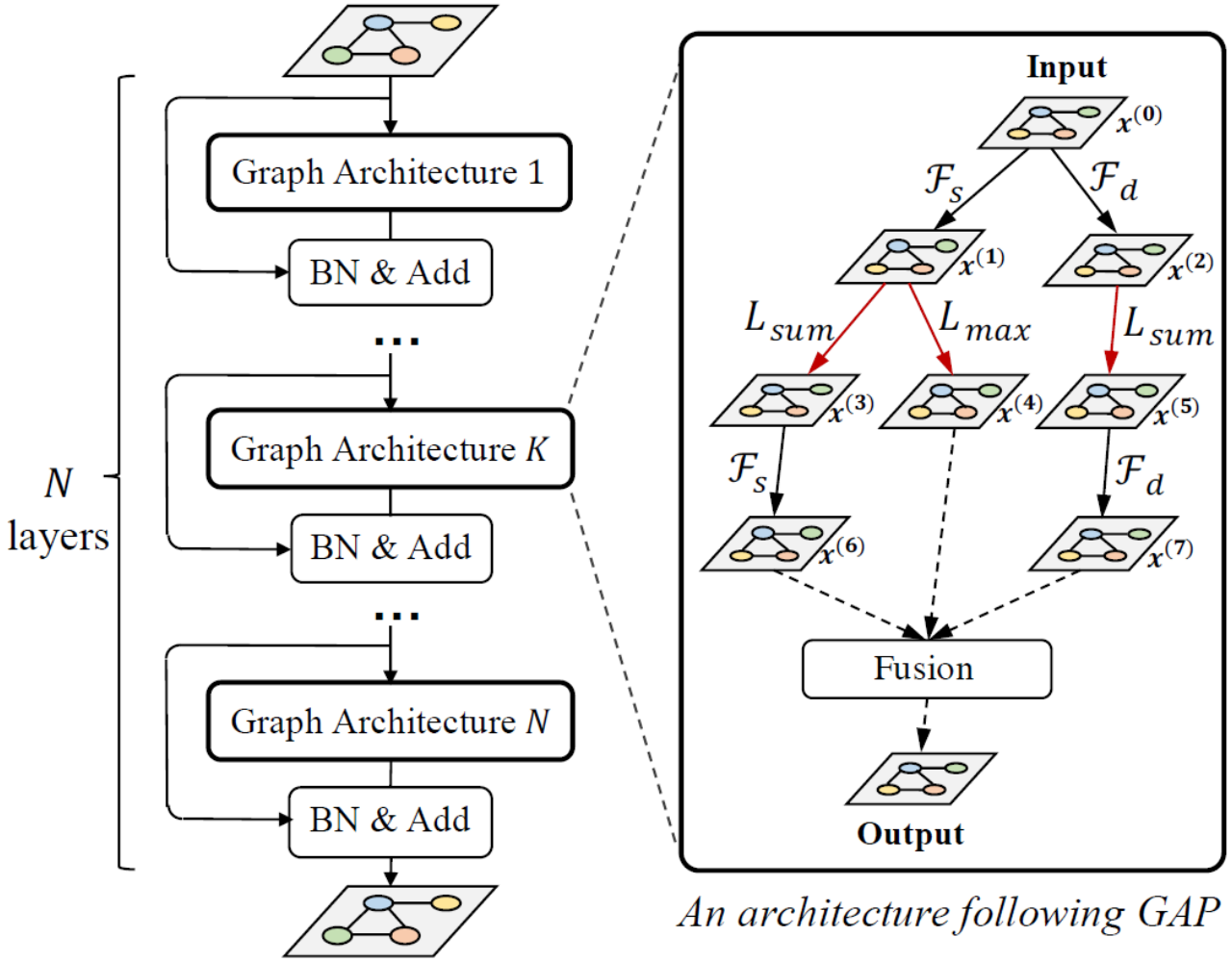
- Search space:
  - Atomic operations: feature filtering and neighbor aggregation
  - Cell architecture: DAG + only one neighbor aggregation per path
    - Nodes only exchange information with first-order neighborhoods
    - Three-level search space: DAG + neighborhood aggregation + DAG



Rethinking Graph Neural Architecture Search from Message-passing, *CVPR 2021*

Graph NAS Example: GNAS

- Overall framework



Rethinking Graph Neural Architecture Search from Message-passing, *CVPR 2021*

Graph NAS Example: GNAS

- Algorithm: adaptively select depth

Algorithm 1 Search Efficient GNN with Optimal message-passing Depth

Input: dataset $\mathcal{S}$
Output: graph neural network $\mathcal{N}$

1: Initialize $\mathcal{D}_o$ as half of average graph diameter of $\mathcal{S}$
2: **repeat**
3: Initialize $\mathcal{N}_s$ as a search network with $\mathcal{D}_o$-layer graph architectures
4: Optimize the architectures of $\mathcal{N}_s$ with GNAS on $\mathcal{S}$
5: Derive a discrete sub-network of $\mathcal{N}_d$ from $\mathcal{N}_s$
6: $\mathcal{D}_i = \mathcal{D}_o$
7: Update $\mathcal{D}_o$ as the number of graph architectures with at least one neighbor aggregation in $\mathcal{N}_d$
8: **until** $\mathcal{D}_i = \mathcal{D}_o$
9: **return** $\mathcal{N}_d$

Rethinking Graph Neural Architecture Search from Message-passing, *CVPR 2021*

Graph NAS Example: LPGNAS

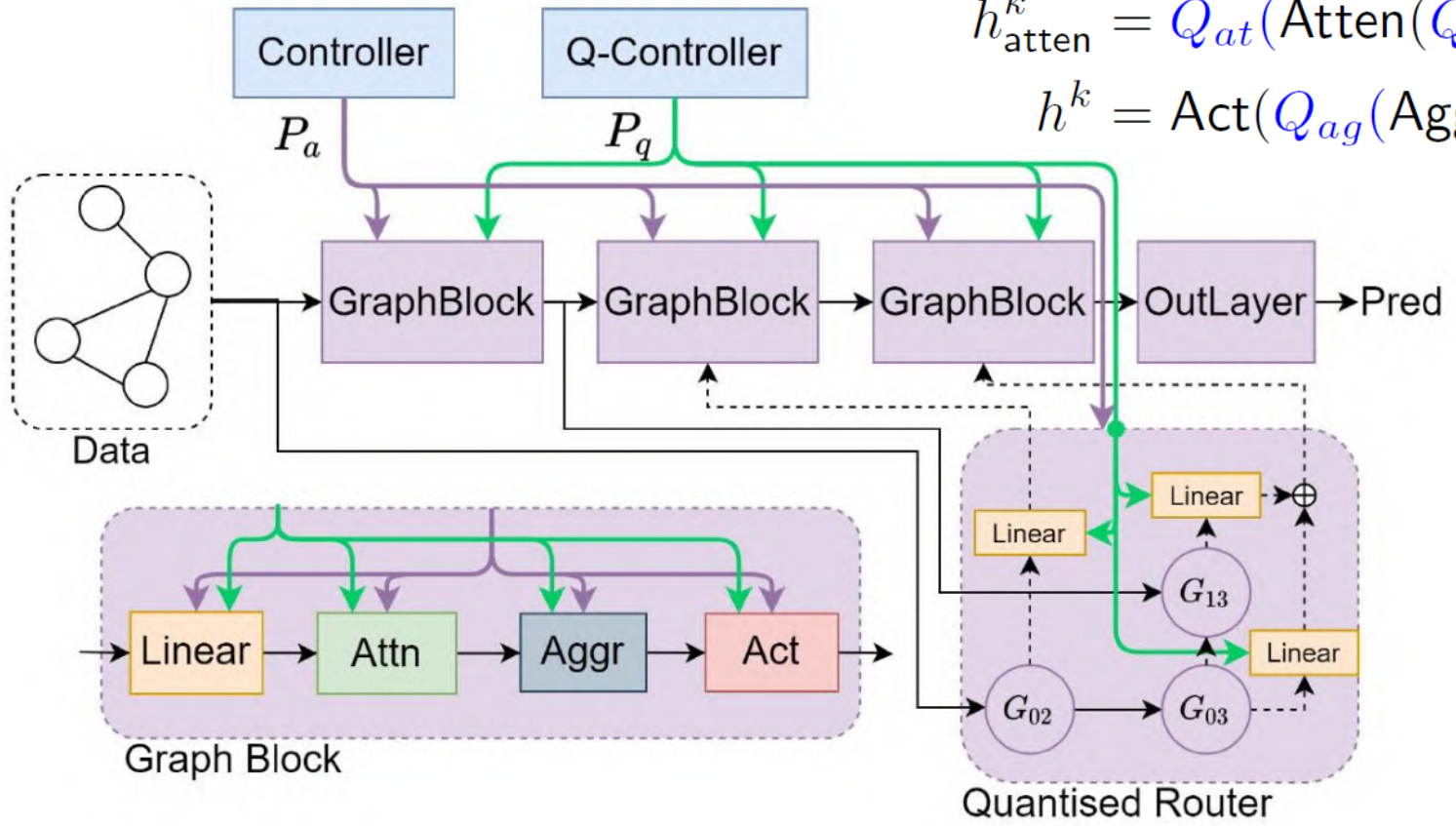
- Consider quantisation in GNNs
  - Quantisation: reduce computation and memory cost

$$h^k = \text{Act}(\text{Aggr}(\text{Atten}(a^k, \text{Linear}(w^k, h^{k-1}))))$$

$$h^k_{\text{linear}} = Q_l(\text{Linear}(Q_w(w^k), Q_h(h^{k-1})))$$

$$h^k_{\text{atten}} = Q_{at}(\text{Atten}(Q_a(a^k), h^k_{\text{linear}}))$$

$$h^k = \text{Act}(Q_{ag}(\text{Aggr}(h^k_{\text{atten}})))$$



Learned low precision graph neural networks, 2021 *EuroMLSys workshop*

Graph NAS Example: LPGNAS

- Search space: micro + macro + quantisation

Weights			Activations		
Quantisation	Frac Bits	Total Bits	Quantisation	Frac Bits	Total Bits
BINARY	0	1	FIX2.2	2	4
BINARY	0	1	FIX4.4	4	8
TERNARY	0	2	FIX2.2	2	4
TERNARY	0	2	FIX4.4	4	8
TERNARY	0	2	FIX4.8	4	12
FIX1.3	3	4	FIX4.4	4	8
FIX2.2	2	4	FIX4.4	4	8
FIX1.5	5	6	FIX4.4	4	8
FIX3.3	3	6	FIX4.4	4	8
FIX2.4	4	6	FIX4.4	4	8
FIX4.4	4	8	FIX4.4	4	8
FIX4.4	4	8	FIX4.8	8	12
FIX4.4	4	8	FIX8.8	8	16
FIX4.8	8	12	FIX4.8	8	12
FIX4.12	12	16	FIX4.4	4	8
FIX4.12	12	16	FIX4.8	8	12
FIX4.12	12	16	FIX8.8	8	16

- Search strategy: differentiable

Learned low precision graph neural networks, 2021 *EuroMLSys workshop*

Graph NAS Example: LPGNAS

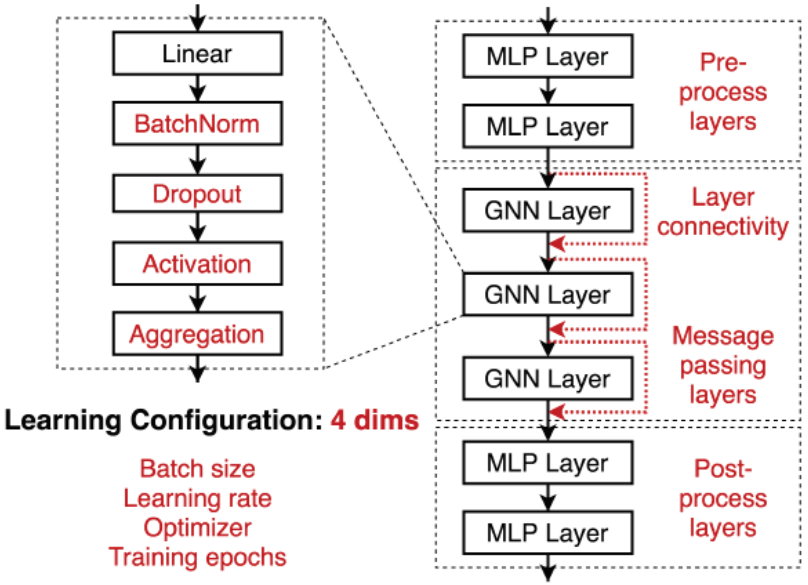
Method	Quan	Cora		CiteSeer		PubMed	
		Accuracy	Size	Accuracy	Size	Accuracy	Size
GraphSage	float	$74.5 \pm 0.0\%$	92.3KB	$75.3 \pm 0.0\%$	237.5KB	$85.3 \pm 0.1\%$	32.2KB
GraphSage	w10a12	$74.3 \pm 0.1\%$	28.8KB	$75.1 \pm 0.1\%$	74.2KB	$85.0 \pm 0.0\%$	10.1KB
GAT	float	$88.9 \pm 0.0\%$	369.5KB	$75.9 \pm 0.0\%$	950.3KB	$86.1 \pm 0.0\%$	129.6KB
GAT	w4a8	$88.8 \pm 0.1\%$	46.2KB	$68.0 \pm 0.1\%$	118.8KB	$82.0 \pm 0.0\%$	16.2KB
JKNet	float	$88.7 \pm 0.0\%$	214.9KB	$75.5 \pm 0.0\%$	505.2KB	$87.6 \pm 0.0\%$	94.5KB
JKNet	w6a8	$88.7 \pm 0.1\%$	**40.3KB**	$73.2 \pm 0.1\%$	94.7KB	$86.1 \pm 0.1\%$	17.7KB
PDNAS-2	float	$89.3 \pm 0.1\%$	192.2KB	$\mathbf{76.3 \pm 0.3\%}$	478.6KB	$89.1 \pm 0.2\%$	72.8KB
PDNAS-3	float	$89.3 \pm 0.1\%$	200.0KB	$75.5 \pm 0.3\%$	494.4KB	$89.1 \pm 0.2\%$	81.4KB
PDNAS-4	float	$89.8 \pm 0.3\%$	205.0KB	$75.6 \pm 0.2\%$	500.0KB	$89.2 \pm 0.1\%$	102.7KB
LPGNAS	mixed	$\mathbf{89.8 \pm 0.0\%}$	67.3KB	$76.2 \pm 0.1\%$	**56.5KB**	$\mathbf{89.6 \pm 0.1\%}$	45.6KB

Learned low precision graph neural networks, 2021 *EuroMLSys workshop*

Graph NAS Example: Design Space

- A systematic study of GNN design space and task space

(a) GNN Design Space

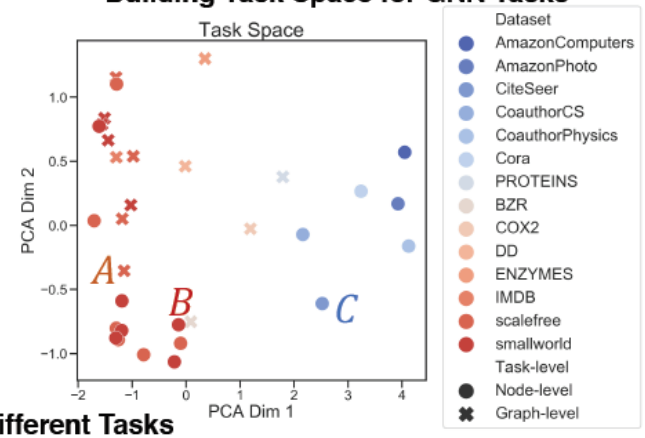


(b) GNN Task Space

Task Similarity Metric

	Anchor Model Performance ranking					Similarity to Task A
Task A	M_1	M_2	M_3	M_4	M_5	1.0
Task B	M_1	M_3	M_2	M_4	M_5	0.8
Task C	M_5	M_1	M_4	M_3	M_2	-0.4

Building Task Space for GNN Tasks



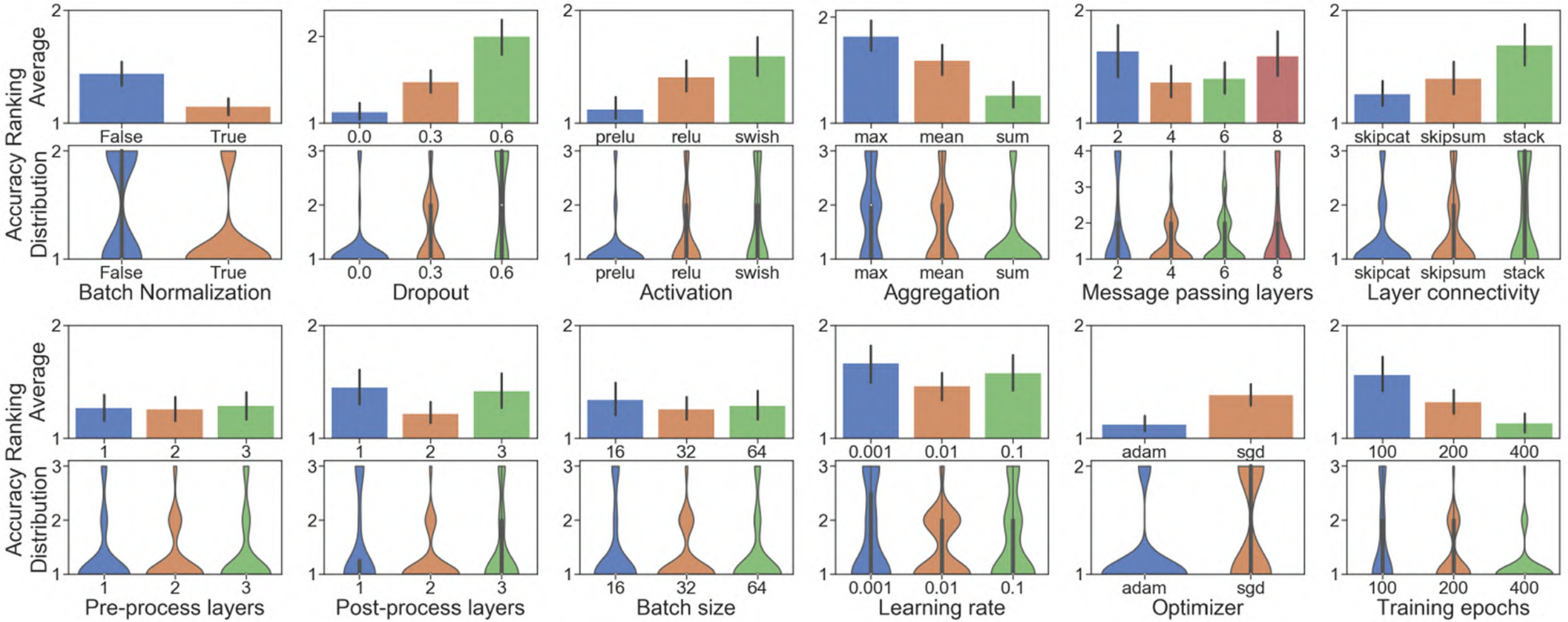
(c) Best GNN Designs Found in Different Tasks

	Pre-process layers	Message passing layers	Post-process layers	Layer connectivity	Aggregation
Task A: graph-IMDB	2	8	2	skip-sum	sum
Task B: node-smallworld	1	8	2	skip-sum	sum
Task C: node-CiteSeer	2	6	2	skip-cat	mean

Design Space for Graph Neural Networks, NeurIPS 2020

Graph NAS Example: Design Space

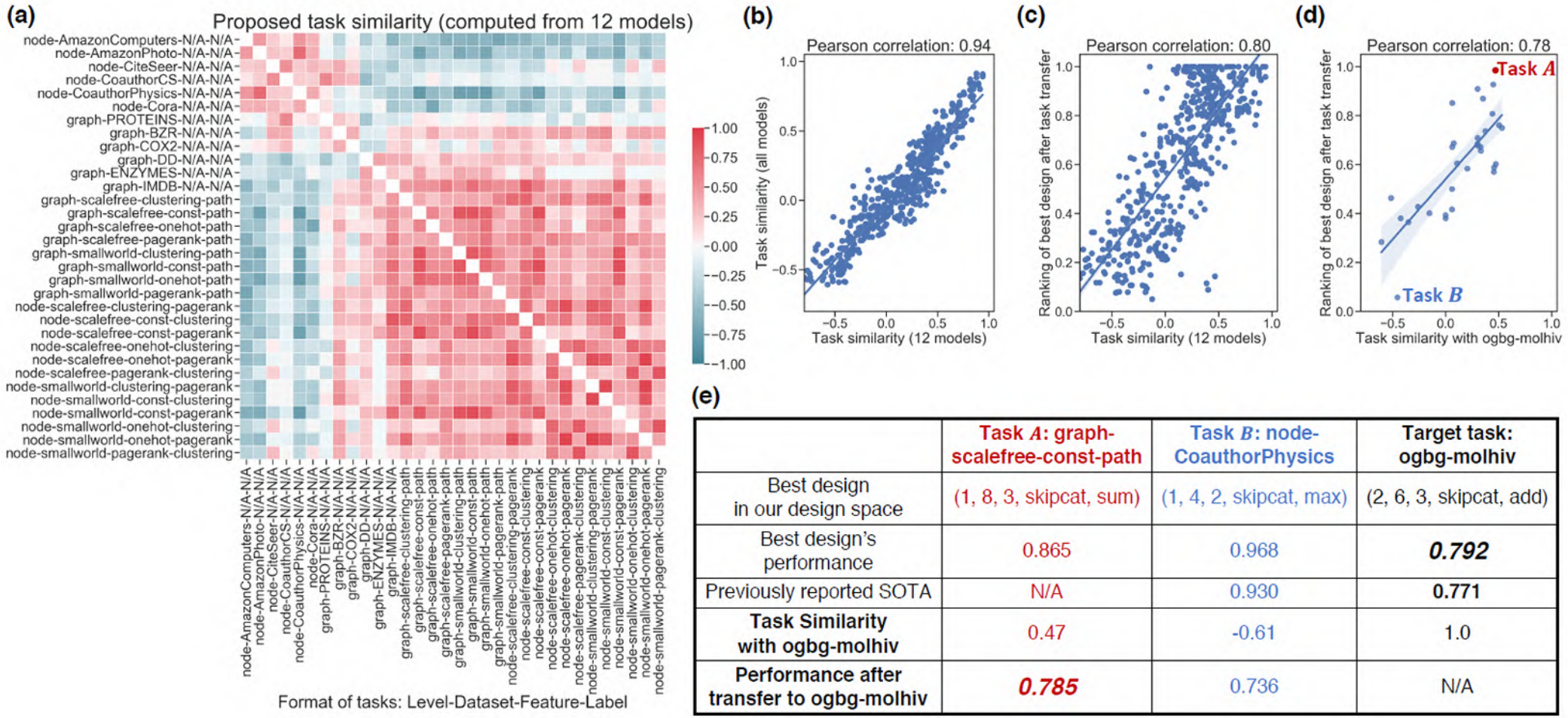
- Key results:



Design Space for Graph Neural Networks, NeurIPS 2020

Graph NAS Example: Design Space

- Key results:

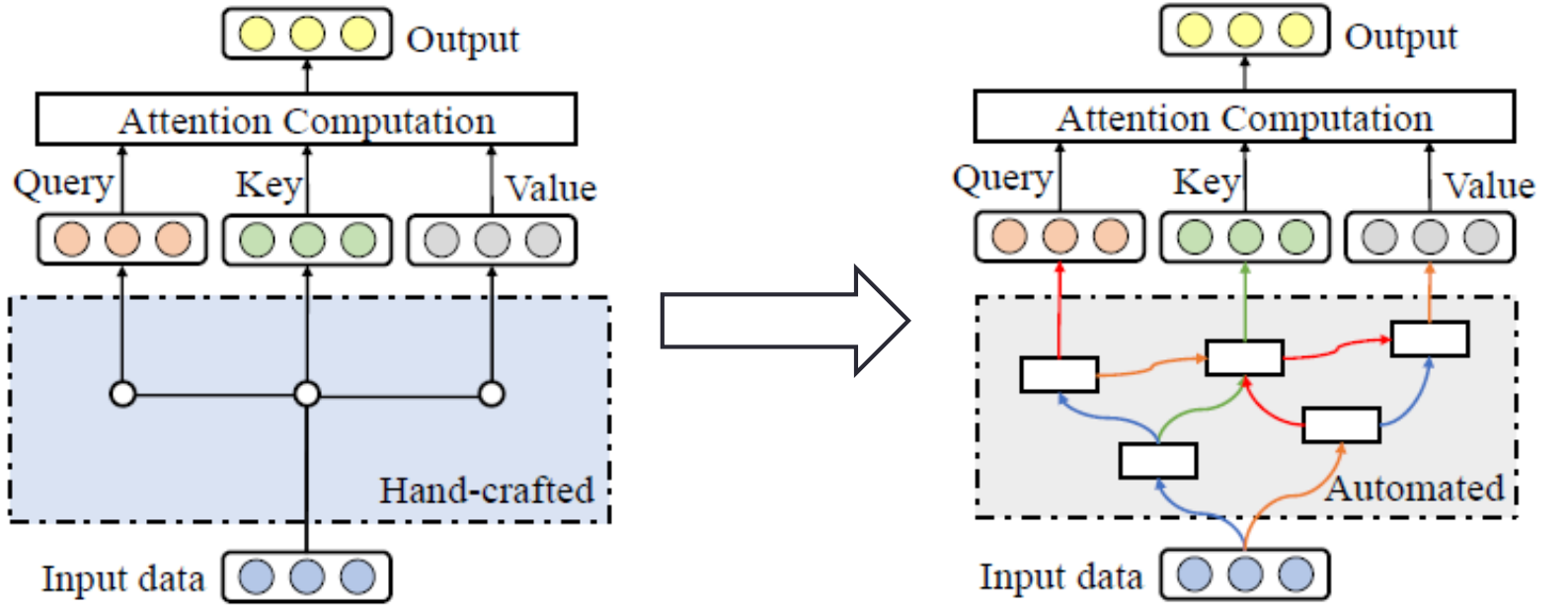


(e)

	Task *A*: graph-scalefree-const-path	Task *B*: node-CoauthorPhysics	Target task: ogbg-molhiv
Best design in our design space	(1, 8, 3, skipcat, sum)	(1, 4, 2, skipcat, max)	(2, 6, 3, skipcat, add)
Best design's performance	0.865	0.968	***0.792***
Previously reported SOTA	N/A	0.930	**0.771**
Task Similarity with ogbg-molhiv	0.47	-0.61	1.0
Performance after transfer to ogbg-molhiv	***0.785***	0.736	N/A

Design Space for Graph Neural Networks, NeurIPS 2020

AutoAttend: Overview



(a) hand-crafted attention representation

(b) automated attention representation

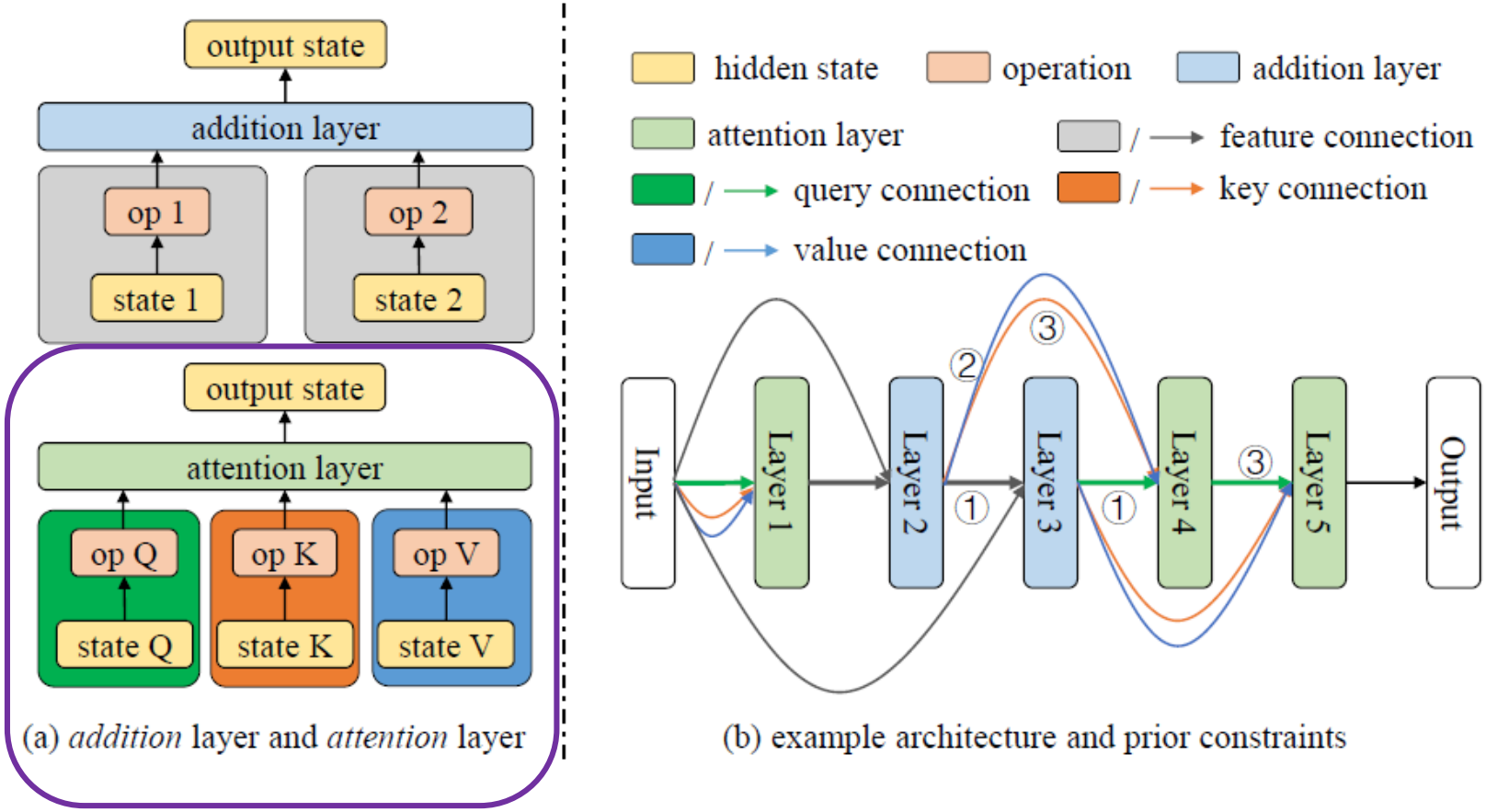
Design self-attention models automatically by searching for the best attention representations

Guan Chaoyu, Xin Wang and Wenwu Zhu.
AutoAttend: Automated Attention Representation Search. ***ICML 2021***.

AutoAttend: Challenges

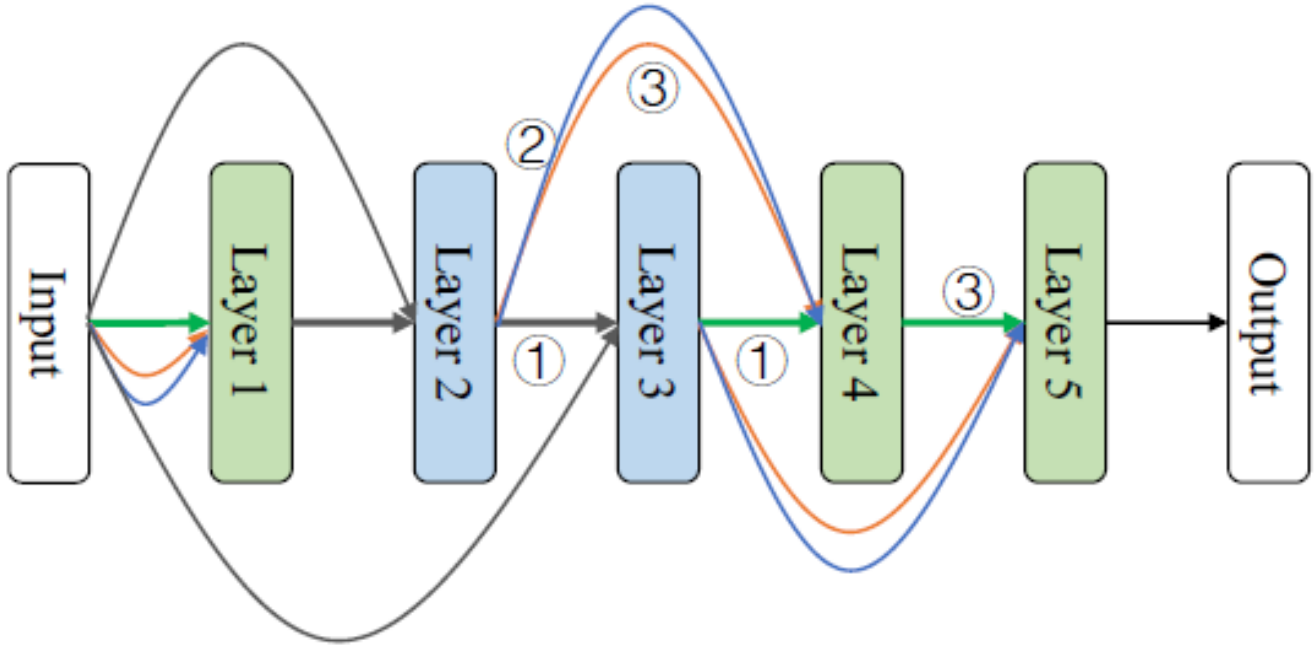
- How to define the most suitable search space?
  - Joint optimization of attention representation and other functional components
  - The search space should be flexible and expressive
  - Relatively low complexity and high feasible architecture density
- How to consider the special characteristics of each sub-architecture in parameter sharing?
  - Parameters of key, query, value, and common feature extraction operations have different functionalities

AutoAttend: Search Space Design



- Attention layer is defined to allow model to have attention aggregation
- A set of layers with optional connections between any two layers

AutoAttend: Improve Density



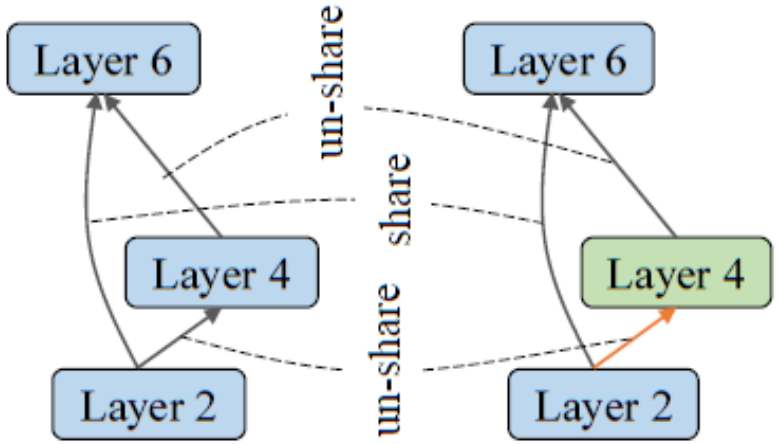
- Skeleton constraint: each layer must have one connection to its previous layer
- Key-Value Constraint: the key and value should have the same input layer
- Non-Zero Constraint: the important connections should not be zero

AutoAttend: Context-aware Parameter Sharing

- One-shot super-net based optimization relaxation

$$a^* = \text{argmin}_{a \in A} L_{val}(a, \mathbf{w}^*),$$
$$\text{s.t. } \mathbf{w}^* = \text{argmin}_{\mathbf{w} \in \mathbf{W}} \mathbb{E}_{a \sim \Gamma(A)} L_{train}(a, \mathbf{w}),$$

- Share parameters according to their contexts



- Evolutionary search for best architectures

AutoAttend: Experiments

MODEL	Search	Transfer						
	SST	SST-B	AG	DBP	YELP-B	YELP	YAHOO	AMZB
GUMBEL-LSTM (CHOI ET AL., 2018)	**53.70**	90.70	-	-	-	-	-	-
CAS-LSTM (CHOI ET AL., 2019)	53.60	**91.30**	-	-	-	-	-	-
DNC+CUW (LE ET AL., 2019)	-	-	93.90	99.00	96.40	65.60	74.30	-
DAGRN (LIU ET AL., 2020)	-	-	**94.93**	**99.16**	**97.34**	**70.14**	-	-
DRNN (WANG, 2018)	-	-	92.90	98.90	96.30	66.40	74.30	95.60
GELE (NIU ET AL., 2019)	-	-	93.20	99.00	96.70	67.00	**75.00**	**96.00**
24-LAYER TRANSFORMER	49.37	86.66	92.17	98.77	94.07	61.22	72.67	95.59
ENAS (PHAM ET AL., 2018)	51.55	88.90	92.39	99.01	96.07	64.60	73.16	95.80
DARTS (LIU ET AL., 2019B)	51.65	87.12	92.24	98.90	95.84	65.12	73.12	95.48
SMASH (BROCK ET AL., 2018)	46.65	85.94	90.88	98.86	95.62	65.26	73.63	95.58
ONE-SHOT (BENDER ET AL., 2018)	50.37	87.08	92.06	98.89	95.78	64.78	73.20	95.20
RANDOM (LI & TALWALKAR, 2019)	49.20	87.15	92.54	98.98	96.00	65.23	72.47	94.87
TEXTNAS (WANG ET AL., 2020B)	52.51	90.33	93.14	99.01	96.41	66.56	73.97	95.94
OURS	**53.71**	**90.50**	**93.53**	**99.08**	**96.62**	**66.82**	**74.48**	**96.04**

MODEL	Transductive			Inductive
	CORA	CITESEER	PUBMED	PPI
GCN	81.5	70.3	79.5	97.7
GAT	83.1	**72.5**	79.0	97.5
ARMA	**83.4**	**72.5**	78.9	**98.5**
APPNP	83.3	71.8	**80.2**	97.8
GRAPHNAS†	80.4	73.0	80.0	98.5
AGNN	83.6	**73.8**	79.7	99.2
OURS-PS	**83.9**	72.7	79.6	98.9
OURS	**83.9**	73.0	**80.6**	**99.3**

- Considerable improvement for natural language processing and graph representation learning tasks

AutoAttend: Ablation Studies

- Ablation studies on the attention layer and context-aware parameter sharing strategies

SPACE	SST	CORA	CITESEER	PUBMED
BASELINE	81.15	81.80	72.18	**81.04**
FULL	**81.68**	**82.96**	**72.90**	**81.04**

CONTEXT	SST	CORA	CITESEER	PUBMED
NC	68.68	77.09	63.68	72.72
SC	68.96	77.81	63.62	73.31
TC	69.38	78.50	64.22	77.54
FC	**69.40**	**78.61**	**64.23**	**77.72**

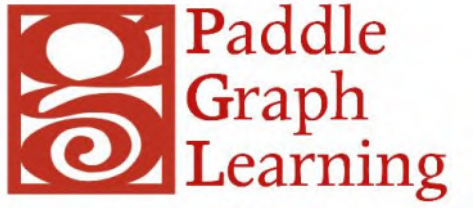
- Using attention layer and considering both the input layer and output layer can increase performance

Outline

- Graph Hyper-parameter Optimization
- Graph Neural Architecture Search
- **Automated Graph Learning Libraries**

AutoML library on Graph

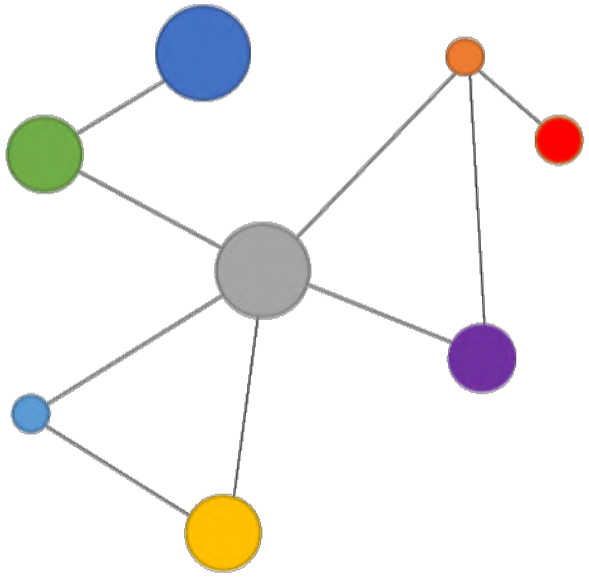
- Graph related



- AutoML related

Introduction – AutoGL

- We design the first autoML framework & toolkit for machine learning on graphs



AutoGL

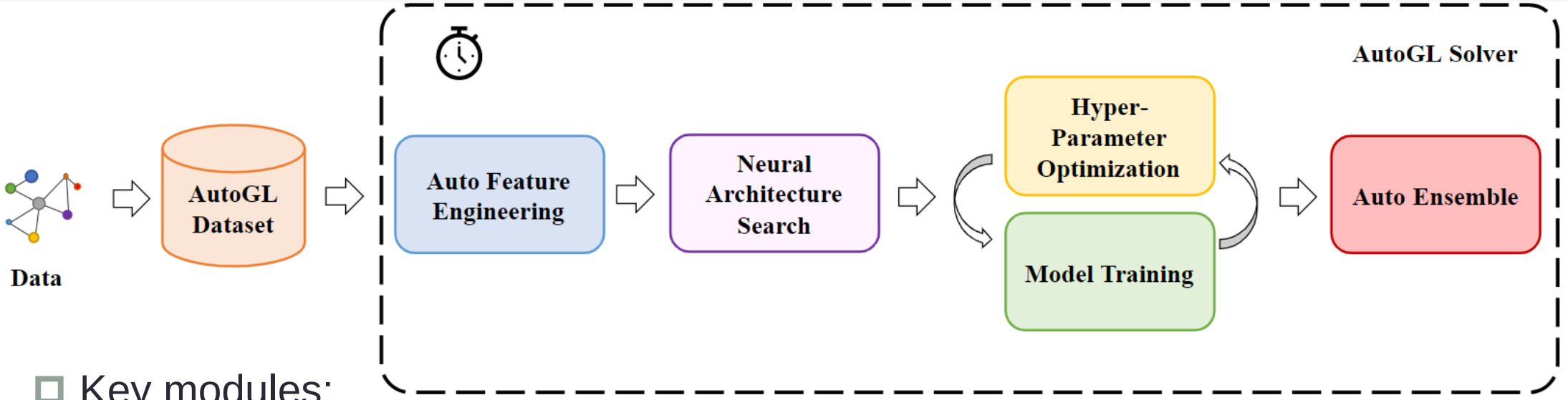
Open source

Easy to use

Flexible to be extended

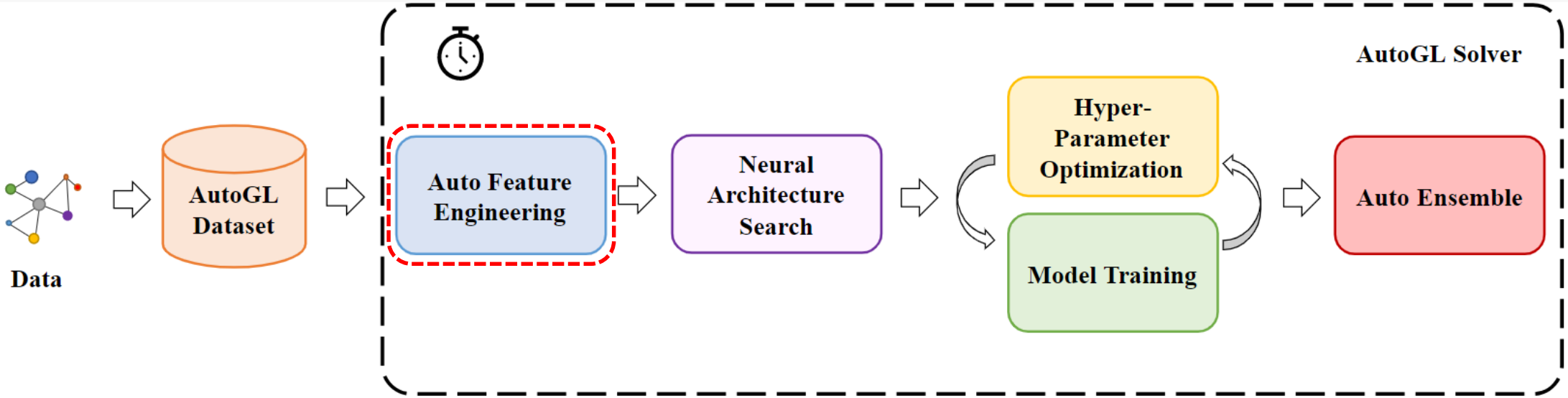
https://mn.cs.Tsinghua.edu.cn/AutoGL
https://github.com/THUMNLab/AutoGL

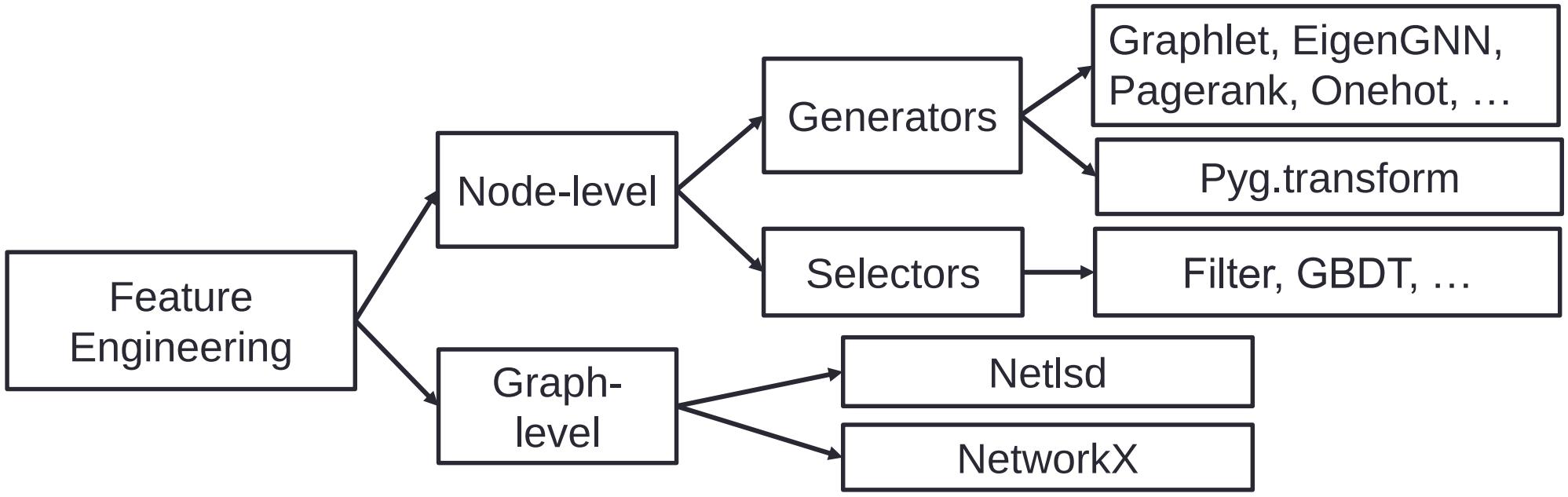
Modular Design



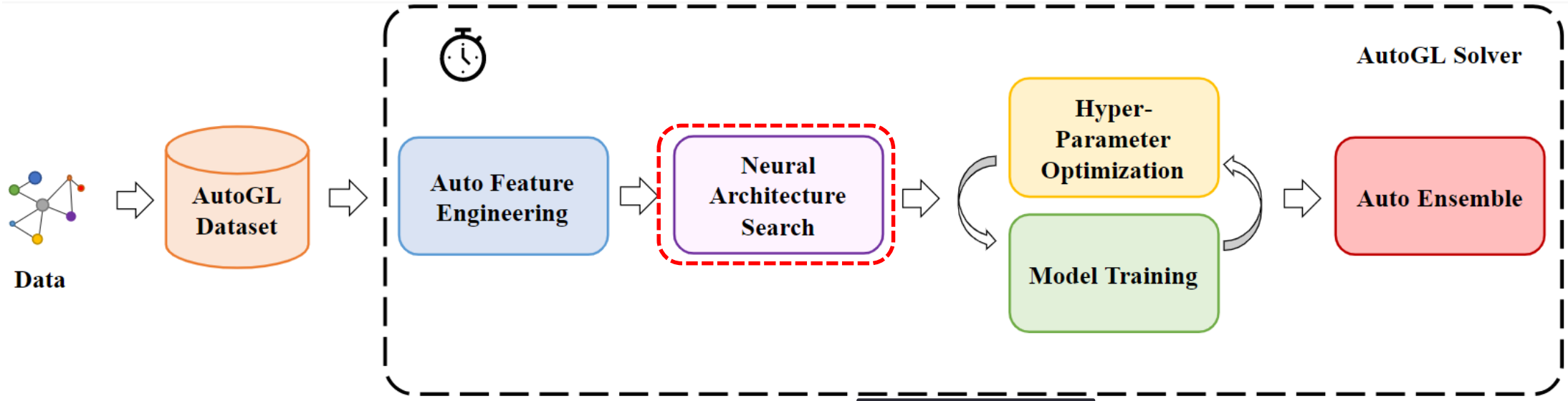
- Key modules:
  - AutoGL Dataset: manage graph datasets
  - AutoGL Solver: a high-level API to control the overall pipeline
  - Five functional modules:
    - Auto Feature Engineering,
    - Neural Architecture Search,
    - Hyper-parameter Optimization
    - Model Training
    - Auto Ensemble

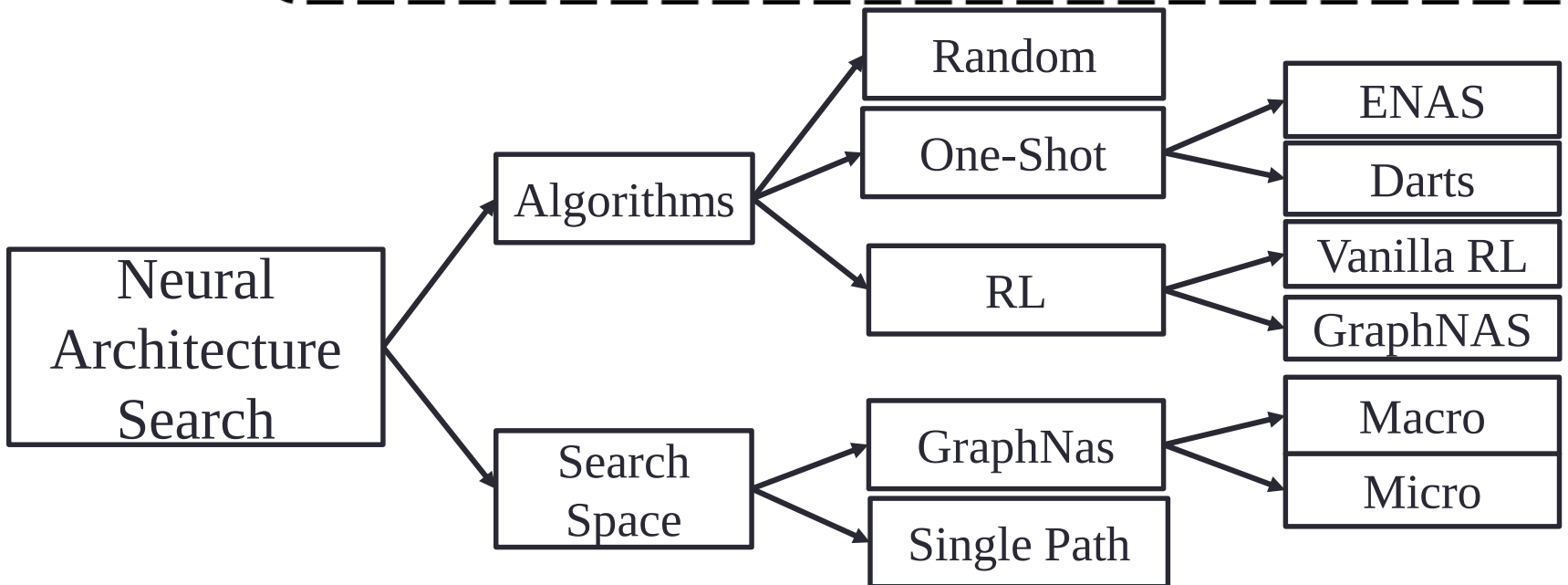
Feature Engineering



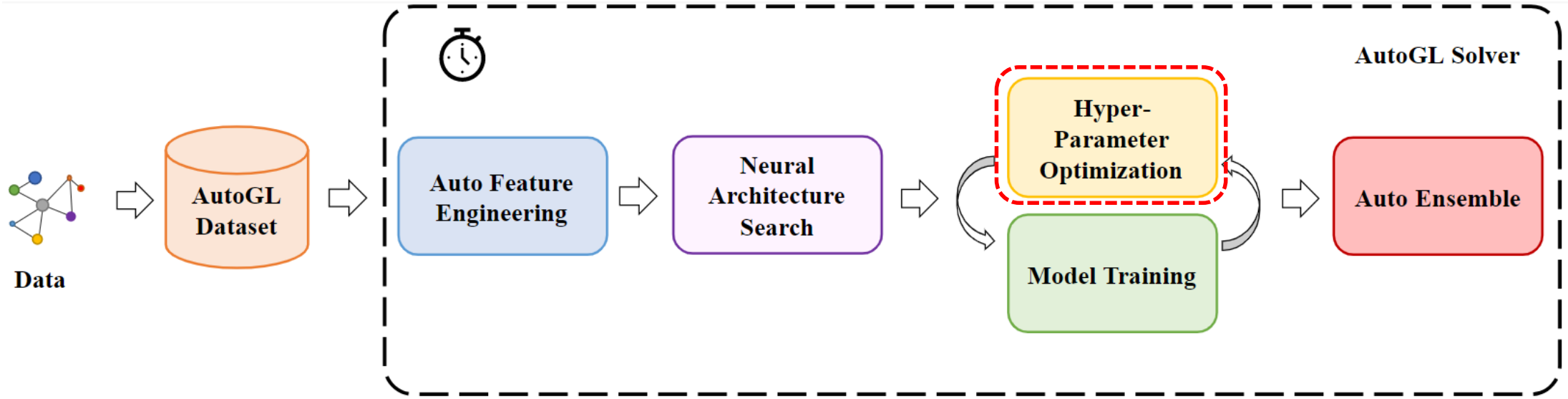


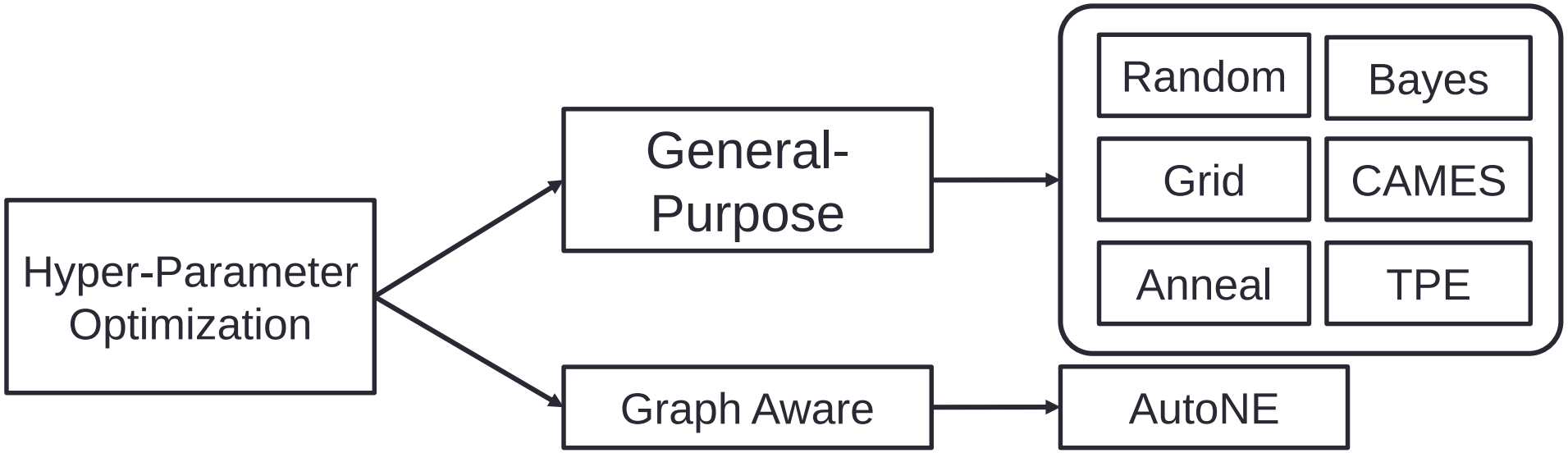
Neural Architecture Search



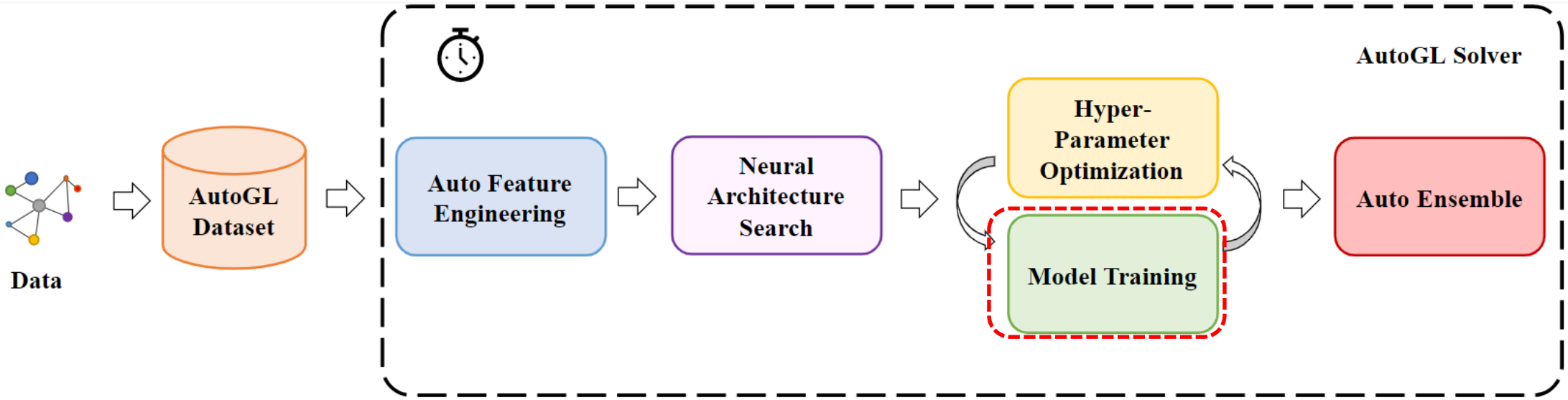


Hyper-Parameter Optimization





Model Training



Trainer

- Learning rate
- Epochs
- Optimizer
- Loss
- Early Stopping

…

Model

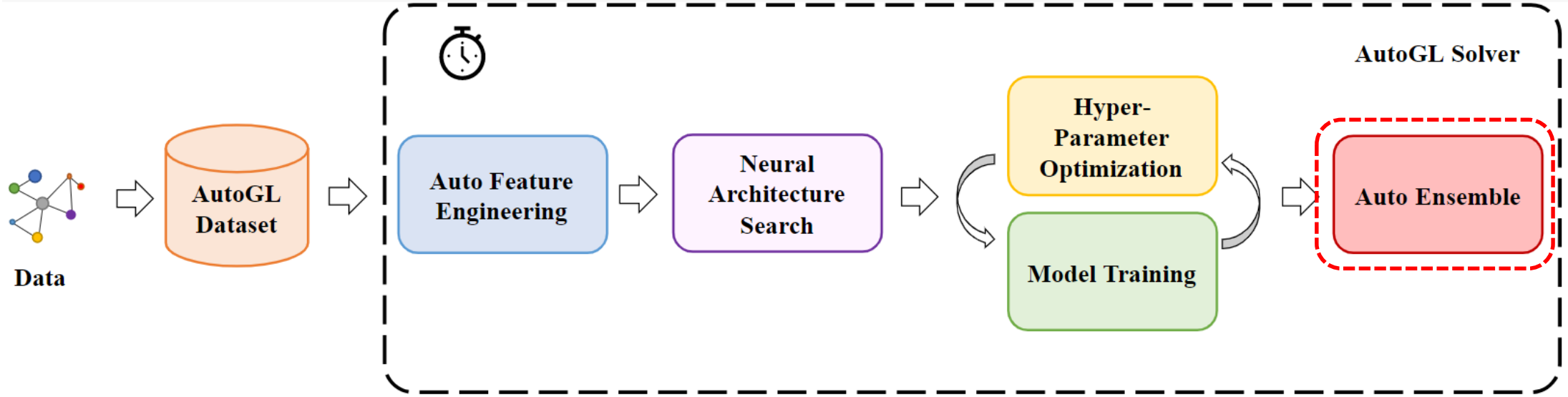
- Forward
- Ops & Architectures
- Dropout & Hidden

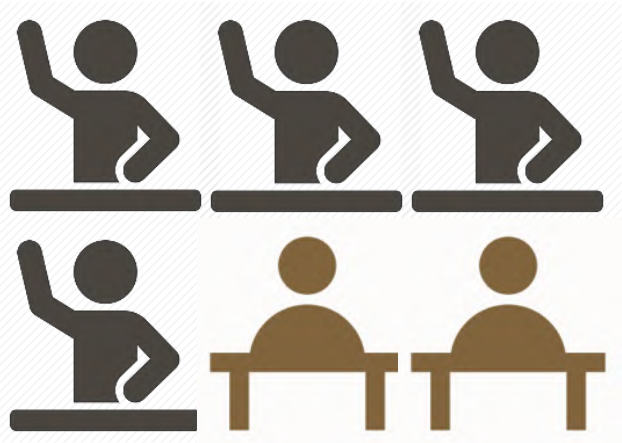
…

Currently supported models

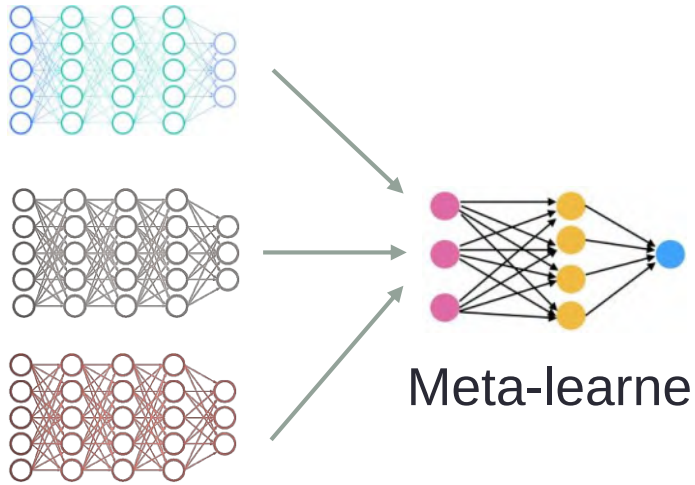
- ☐ Node classification
- ☐ Link Prediction
- ☐ Graph classification

Ensemble





voting



stacking

Example Results

Table 1: The results of node classification

Model	Cora	CiteSeer	PubMed
GCN	80.9 ± 0.7	70.9 ± 0.7	78.7 ± 0.6
GAT	82.3 ± 0.7	71.9 ± 0.6	77.9 ± 0.4
GraphSAGE	74.5 ± 1.8	67.2 ± 0.9	76.8 ± 0.6
AutoGL	$\mathbf{83.2 \pm 0.6}$	$\mathbf{72.4 \pm 0.6}$	$\mathbf{79.3 \pm 0.4}$

Table 2: The results of graph classification

Model	MUTAG	PROTEINS	IMDB-B
Top-K Pooling	80.8 ± 7.1	69.5 ± 4.4	71.0 ± 5.5
GIN	82.7 ± 6.9	66.5 ± 3.9	69.1 ± 3.7
AutoGL	$\mathbf{87.6 \pm 6.0}$	$\mathbf{73.3 \pm 4.4}$	$\mathbf{72.1 \pm 5.0}$

Table 3: The results of different HPO methods for node classification

Method	Trials	Cora		CiteSeer		PubMed	
		GCN	GAT	GCN	GAT	GCN	GAT
None		80.9 ± 0.7	82.3 ± 0.7	70.9 ± 0.7	71.9 ± 0.6	78.7 ± 0.6	77.9 ± 0.4
random	1	81.0 ± 0.6	81.4 ± 1.1	70.4 ± 0.7	70.1 ± 1.1	78.3 ± 0.8	76.9 ± 0.8
	10	82.0 ± 0.6	82.5 ± 0.7	71.5 ± 0.6	$\mathbf{72.2 \pm 0.7}$	79.1 ± 0.3	78.2 ± 0.3
	50	81.8 ± 1.1	$\mathbf{83.2 \pm 0.7}$	71.1 ± 1.0	72.1 ± 1.0	$\mathbf{79.2 \pm 0.4}$	78.2 ± 0.4
TPE	1	81.8 ± 0.6	81.9 ± 1.0	70.1 ± 1.2	71.0 ± 1.2	78.7 ± 0.6	77.7 ± 0.6
	10	82.0 ± 0.7	82.3 ± 1.2	71.2 ± 0.6	72.1 ± 0.7	79.0 ± 0.4	$\mathbf{78.3 \pm 0.4}$
	50	$\mathbf{82.1 \pm 1.0}$	83.2 ± 0.8	$\mathbf{72.4 \pm 0.6}$	71.6 ± 0.8	79.1 ± 0.6	78.1 ± 0.4

AutoGL Plans

Incoming new features:

- DGL backend
- More large-scale graph support
  - E.g., sampling, distributed, etc.
- More graph tasks
  - E.g., heterogenous graphs, spatial-temporal graphs, etc.

Warmly welcome all feedbacks and suggestions!

Contact: autogl@tsinghua.edu.cn

Section Summary

- Graph Hyper-parameter Optimization
- Graph Neural Architecture Search
- Automated Graph Learning Librawries

- Open Problems:
  - Graph models for AutoML
    - E.g., regard NN as Directed Acyclic Graph (DAG)
    - E.g., using GNNs as surrogate models
  - Robustness and explainability
  - Hardware-aware models
  - Comprehensive evaluation protocols

Automated Graph Learning Survey

Automated Machine Learning on Graphs: A Survey

Ziwei Zhang* , **Xin Wang*** and **Wenwu Zhu†**
Tsinghua University, Beijing, China
zw-zhang16@mails.tsinghua.edu.cn, {xin_wang,wwzhu}@tsinghua.edu.cn

Abstract

Machine learning on graphs has been extensively studied in both academic and industry. However, as the literature on graph learning booms with a vast number of emerging methods and techniques, it becomes increasingly difficult to manually design the optimal machine learning algorithm for different graph-related tasks. To solve this critical challenge, automated machine learning (AutoML) on graphs which combines the strength of graph machine learning and AutoML together, is gaining attention from the research community. Therefore, we comprehensively survey AutoML on graphs in this paper, primarily focusing on hyper-parameter optimization (HPO) and neural architecture search

Zitnik and Leskovec, 2017], physical simulation [Kipf *et al.*, 2018], traffic forecasting [Li *et al.*, 2018b; Yu *et al.*, 2018], knowledge representation [Wang *et al.*, 2017], drug re-purposing [Ioannidis *et al.*, 2020; Gysi *et al.*, 2020] and pandemic prediction [Kapoor *et al.*, 2020] for Covid-19.

Despite the popularity of graph machine learning algorithms, the existing literature heavily relies on manual hyper-parameter or architecture design to achieve the best performance, resulting in costly human efforts when a vast number of models emerge for various graph tasks. Take GNNs as an example. At least one hundred new general-purpose architectures have been published in top-tier machine learning and data mining conferences in the year 2020 alone, not to mention cross-disciplinary researches of task-specific designs. More and more human efforts are inevitably needed if

Ziwei Zhang, Xin Wang, Wenwu Zhu.
Automated Machine Learning on Graphs: A Survey. *IJCAI 2021.*

Paper collection: https://github.com/THUMNLab/awesome-auto-graph-learning

Summary

- Learn vectorized representation of nodes/graphs
  - Preserve structures and properties
- End-to-end learning paradigms on graphs
  - Balance structures and attributes/features

Network Embedding

GNN

Automated Graph Machine Learning

- The automation of designing learning algorithms on graphs
  - Handle large-scale and complicated graph structures

Thanks!

Xin Wang
xin_wang@tsinghua.edu.cn
http://mn.cs.tsinghua.edu.cn/xinwang/

Ziwei Zhang
zwzhang@tsinghua.edu.cn
https://zw-zhang.github.io/

Wenwu Zhu
wwzhu@tsinghua.edu.cn