Learning Local Algorithms with Global Views for Graph Combinatorial Optimization

Le Song

Georgia Institute of Technology
Joint work with

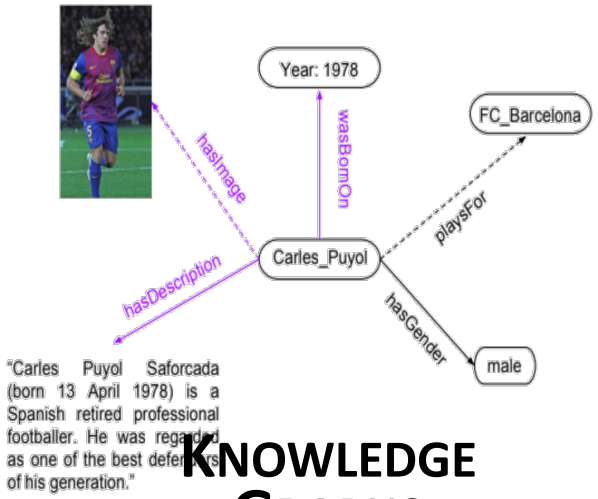
Hanjun Dai
Google Brain

Xinshi Chen
Georgia Tech

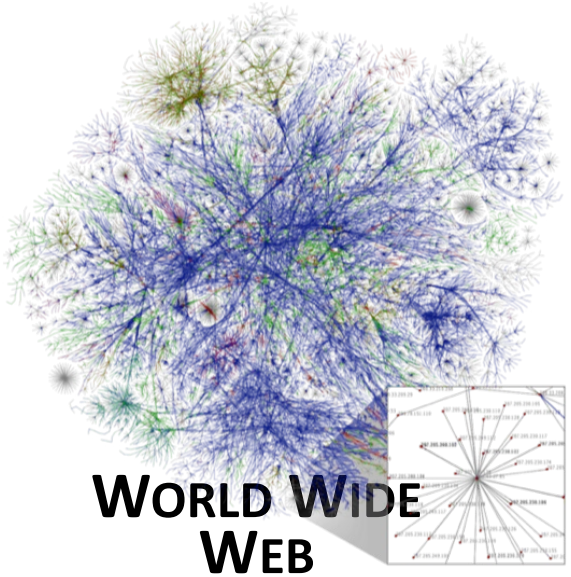
Yu Li
CUHK

Graphs are everywhere

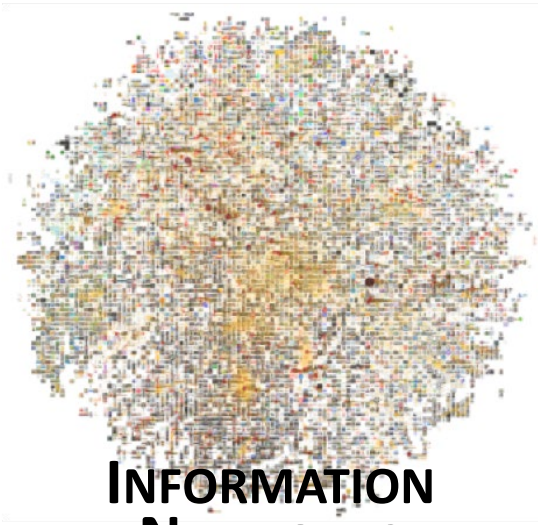
SOCIAL NETWORKS



KNOWLEDGE GRAPHS

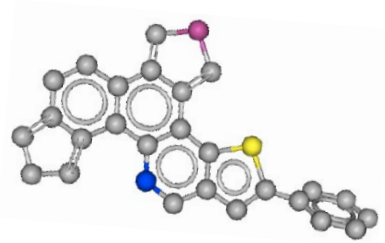


WORLD WIDE WEB

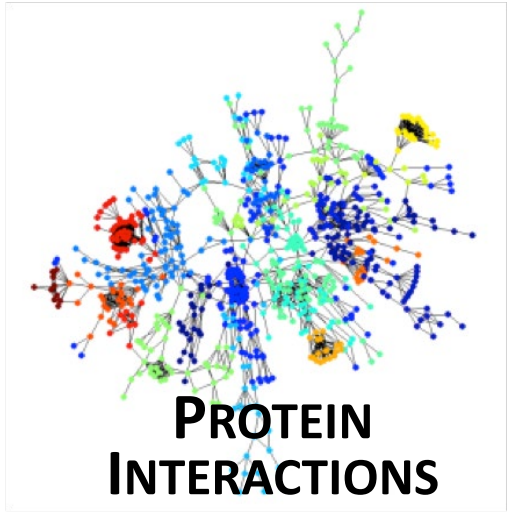


INFORMATION NETWORKS

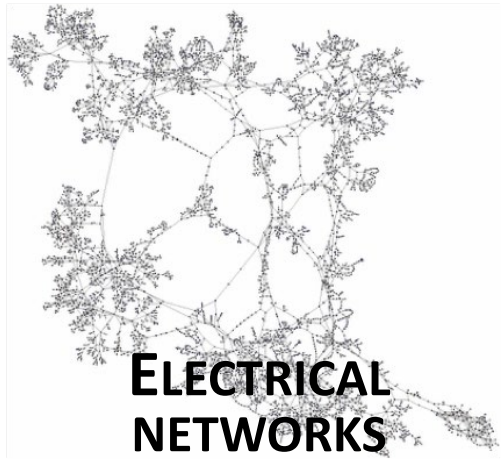
TRANSPORTATION NETWORKS



MOLECULAR GRAPHS

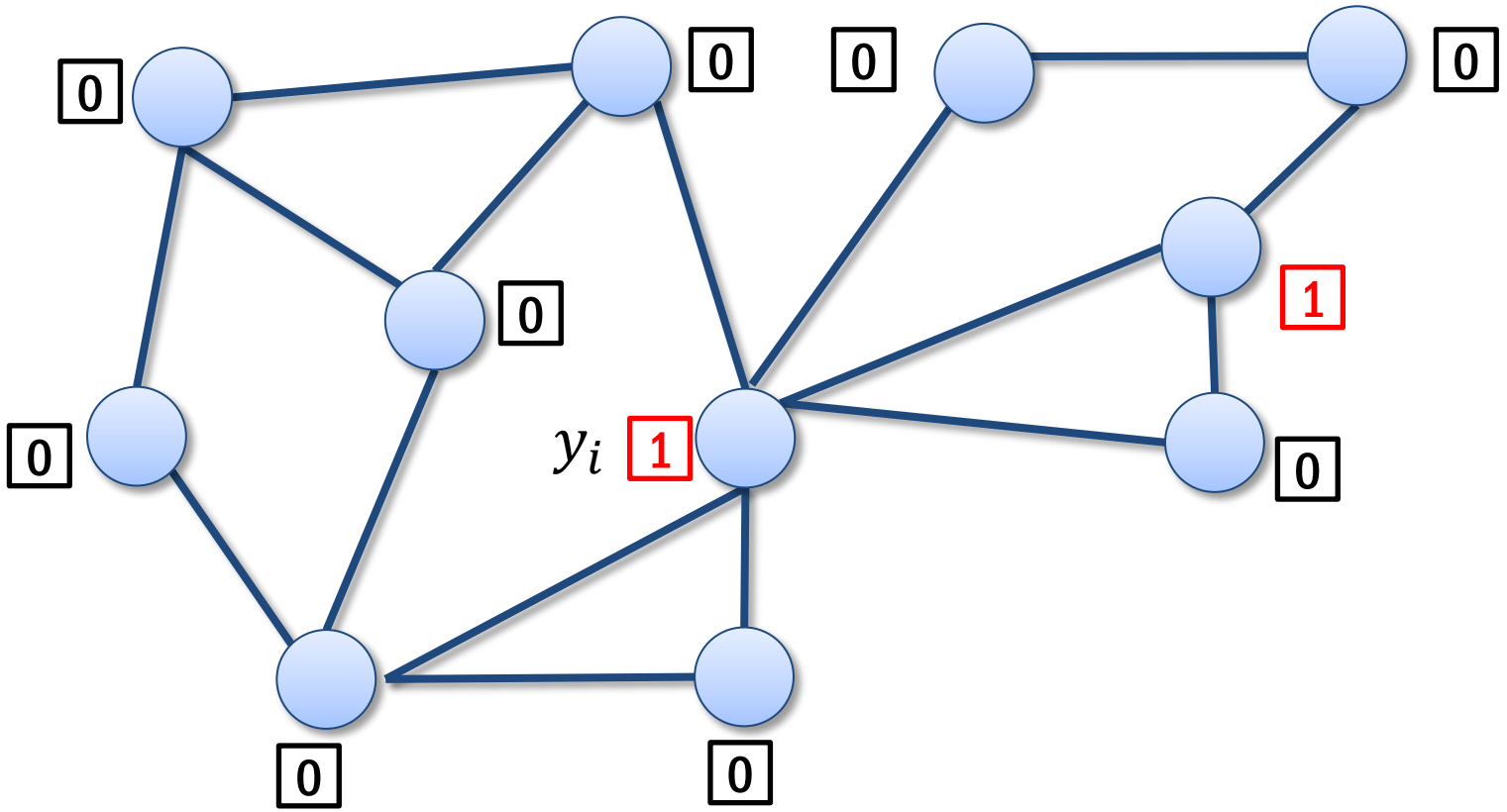


PROTEIN INTERACTIONS



ELECTRICAL NETWORKS

Combinatorial optimization over graph



Minimum vertex cover

$$\min_{y\in\{0,1\}^{|\boldsymbol{V}|}} \quad f(y;G) \coloneqq \sum_{i\in\boldsymbol{V}} y_i$$

$$\text{s.t.} \quad g(y;G) \coloneqq y_i + y_j \geq 1 \quad \forall (i,j) \in \boldsymbol{\mathcal{E}}$$

NP-hard problems

Graph neural networks (GNN/MPN/Structure2vec)

- Obtain embedding via iterative updates:
  - Initialize $h_i^{(0)} = \sigma(W_1 X_i), \forall\, i$
  - Iterate T times:

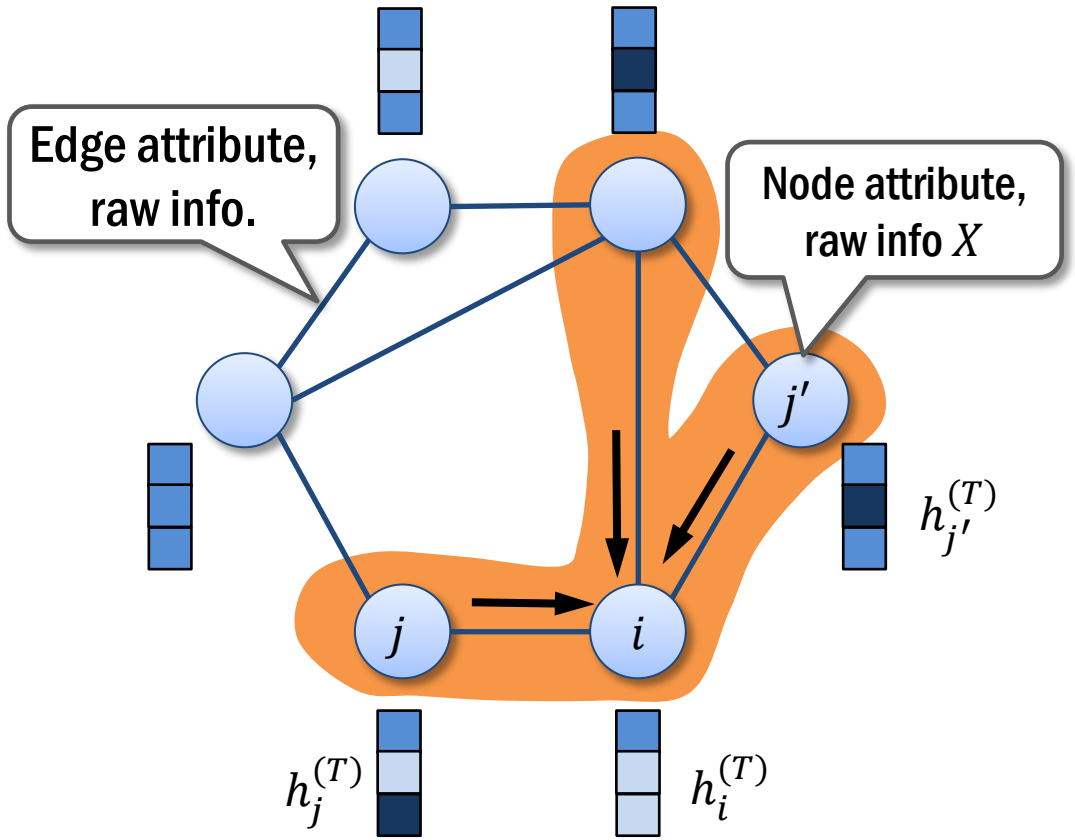
neural network

$$h_i^{(t)} \leftarrow \sigma\left(W_1 h_i^{(t-1)} + W_2 \sum_{j \in \mathcal{N}(i)} h_j^{(t-1)}\right)$$

⇩

Node selection probability

$$p_i = \text{Output}\left(h_i^{(T)}\right)$$

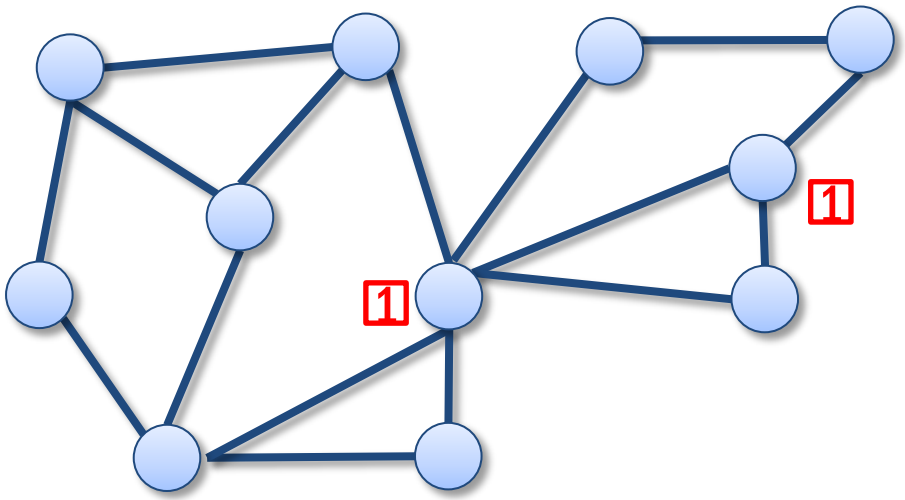


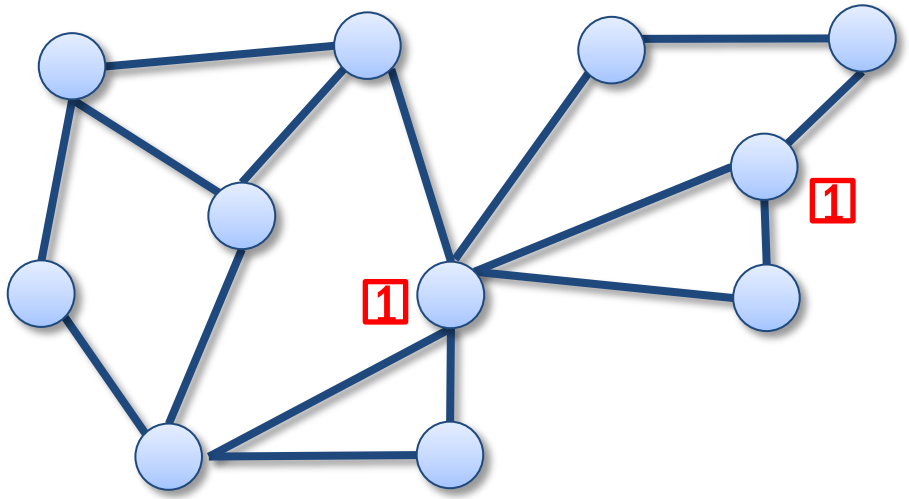
Optimize GNN parameters =
Search over the space of graph algorithms

[Dai, et al. ICML 16 and many others]

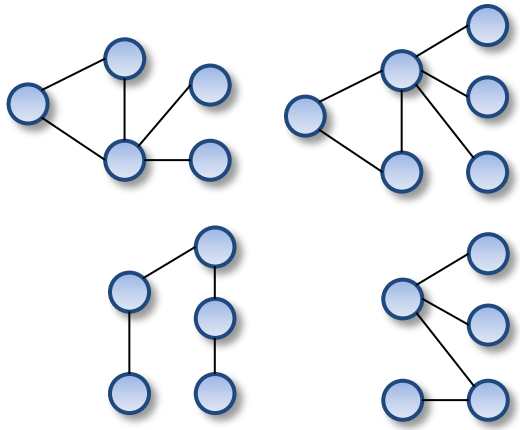
Sequential vs distributed local algorithms

- Sequential algorithms
  - Nodes are selected one by one
  - Graph may change after each selection
- Distributed local algorithms
  - Nodes are selected simultaneously
  - Decision are made based on local graph structure

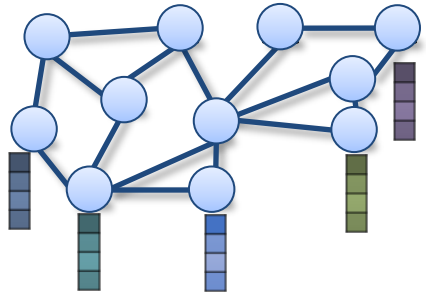




Learning algorithms with reinforcement learning



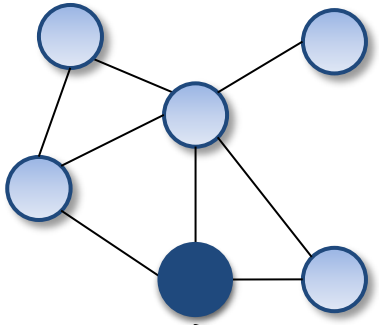
1. Training graphs



2. GNN node embeddings

$\{h_i^T\}_{i \in V}$

5. Inference on test graph



Greedy action

$i^* = \text{argmax}_i \, q(i|S)$

3. State-action value function

$q(i|S) = \theta_1 \sigma(\theta_2 \sum_{j \in V} h_j + \theta_3 h_i)$

4. Train

Reward -1 for each step

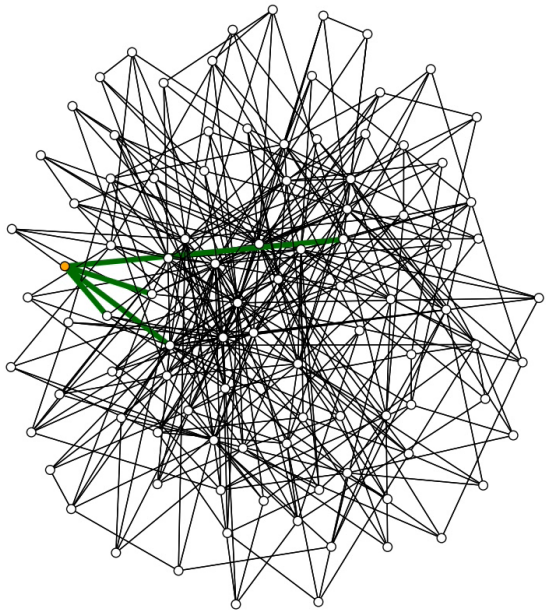
Reinforcement Learning

[Learning combinatorial optimization over graphs. *NIPS 2017*.]

Example of learned sequential algorithms

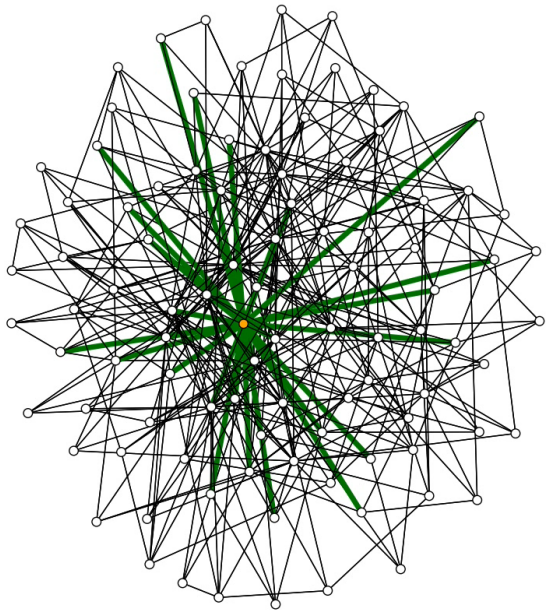
- Learned GNN algorithm balances between
  - Degree of the picked node and
  - Connectivity of the graph

S2V-DQN



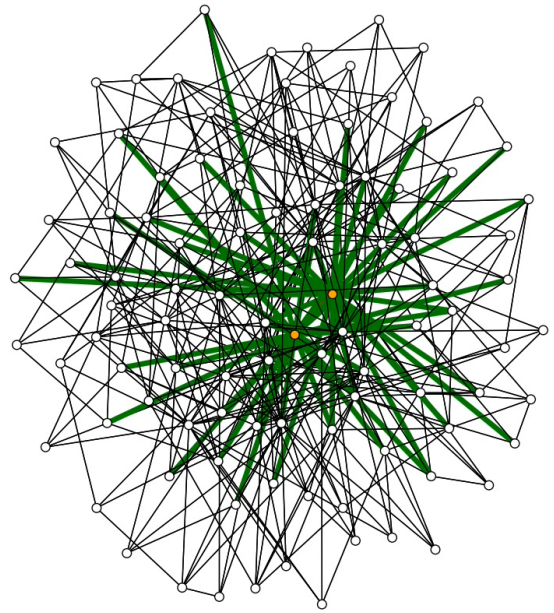
GNN algorithm

Greedy-Node



Node greedy

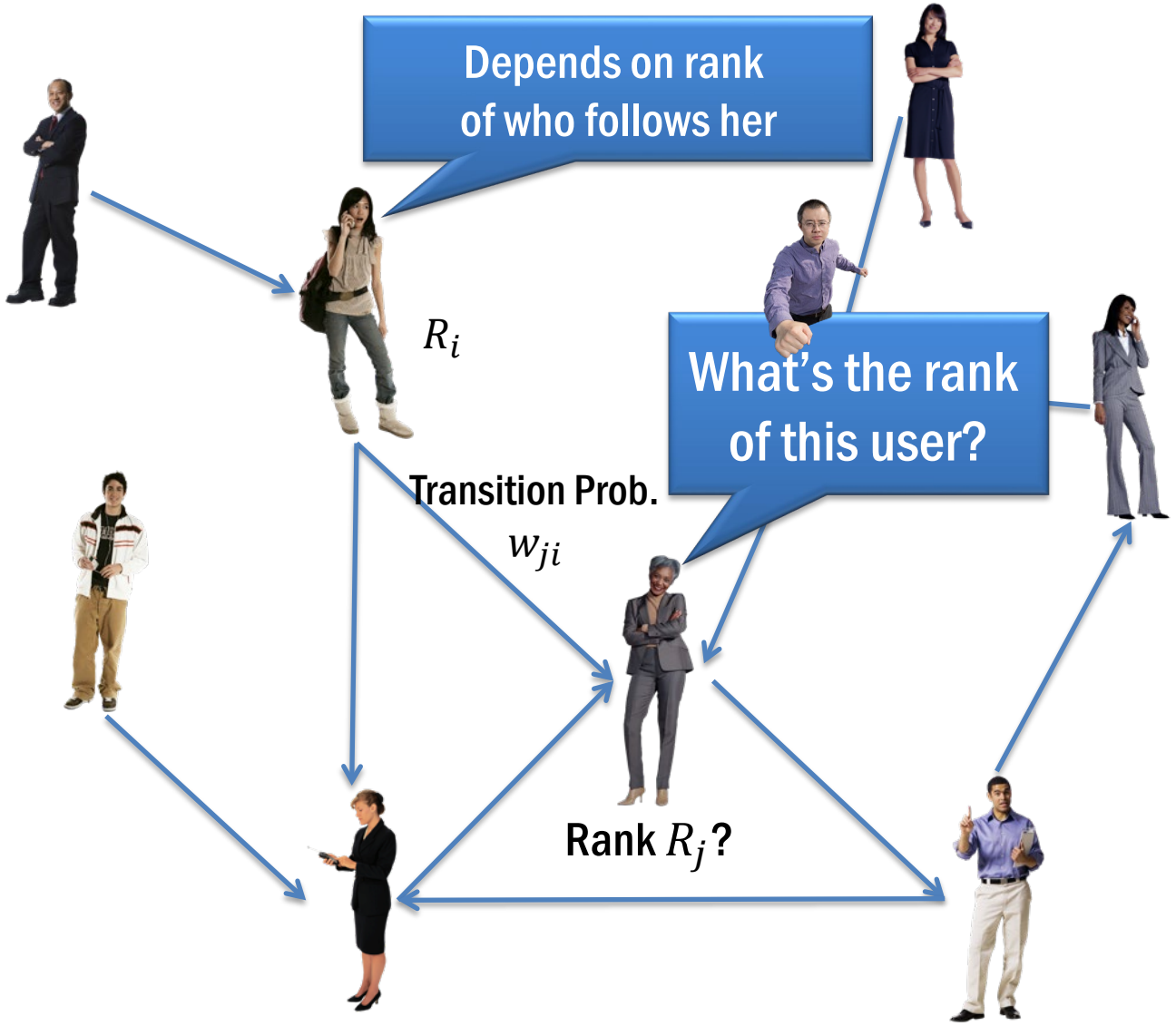
Greedy-Edge



Edge greedy

[Dai et al. NIPS 2017]

Example of distributed local algorithms: PageRank



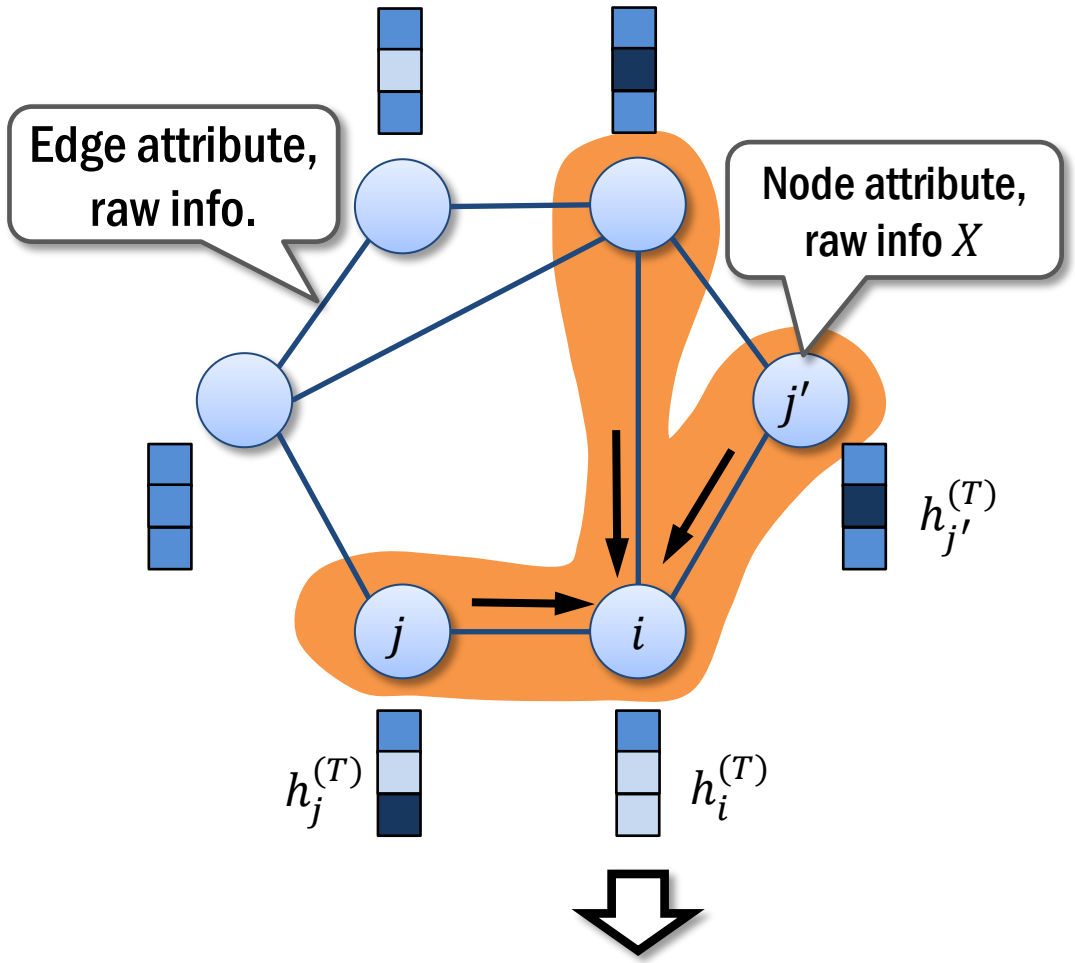
Iterate until convergence:

$$R_j \leftarrow \sum_{(j,i)\in E} w_{ji} R_i$$

Manually designed. Can we learn from data?

GNN = Parametrized distributed local graph algorithm

- Expressiveness of GNN:
  - Can express distributed local algorithms (Weisfeler-Lehman for graph isomorphism)
- Limitation of GNN:
  - Only capture information in T-hop neighborhood
  - When T is large, it is hard to train GNN.
  - Without global information, not expressive enough



$$p_i = \text{Output}\left(h_i^{(T)}\right)$$

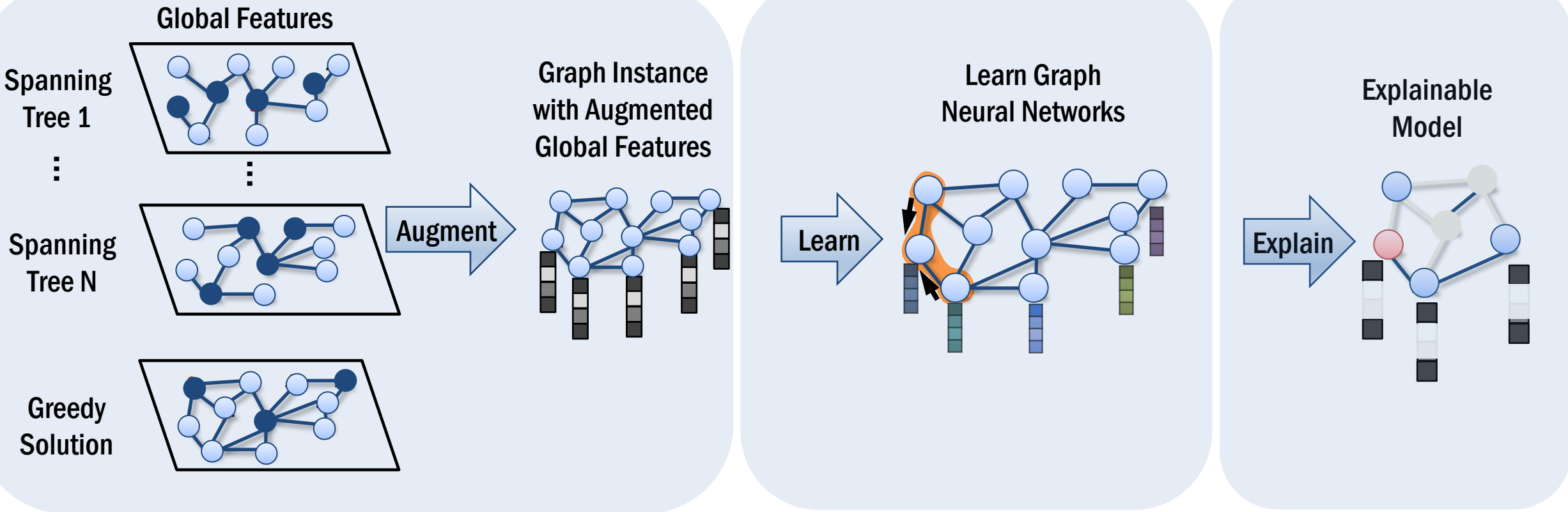
[Sato et al. NIPS 2019]

Challenges for learning new algorithms

1. How to design the search space?

2. How to learn these algorithms?
Supervised, Unsupervised

3. Can we interpret what's discovered?



Differentiable Algorithm Discovery (DAD) framework

Search Space Design for GNN

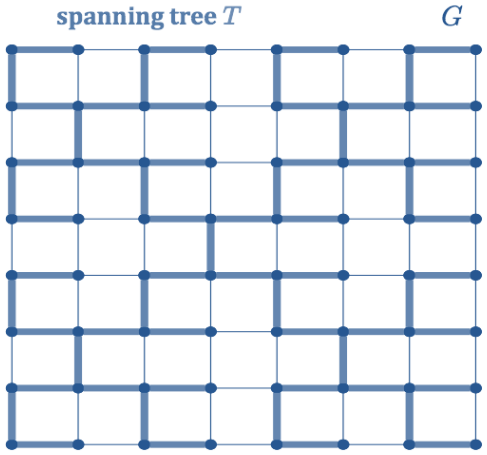
Motivating example

- The best known algorithm for solving a general linear system takes time $\mathcal{O}(n^{2.373})$
- Kelner et al. (2013) proposed an algorithm for solving Laplacian system:

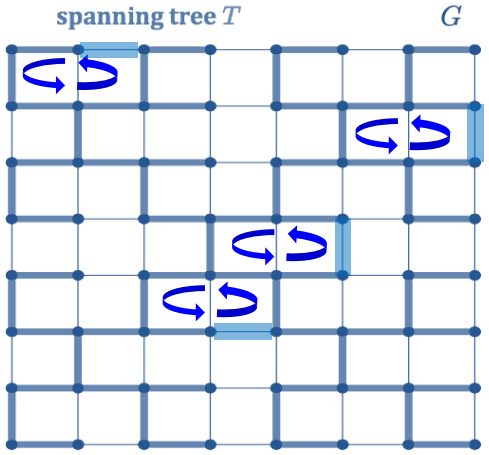
$$Lx = b, \text{ where } L \text{ is Laplacian matrix}$$

in nearly-linear time.

Step 1: Find a low-stretch spanning tree and obtain an initial solution on the tree.

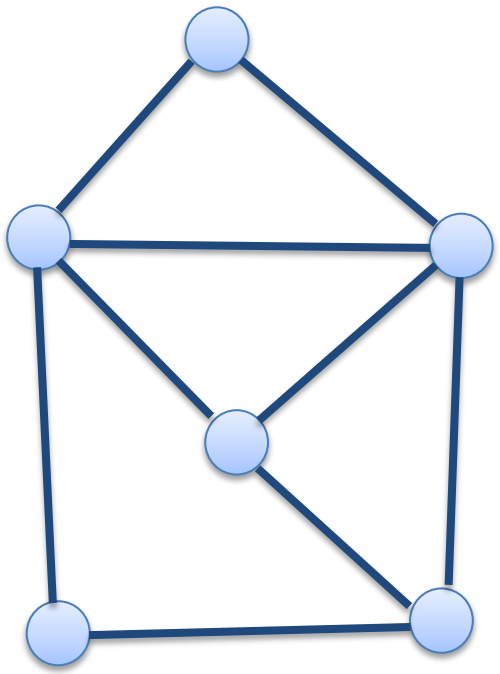


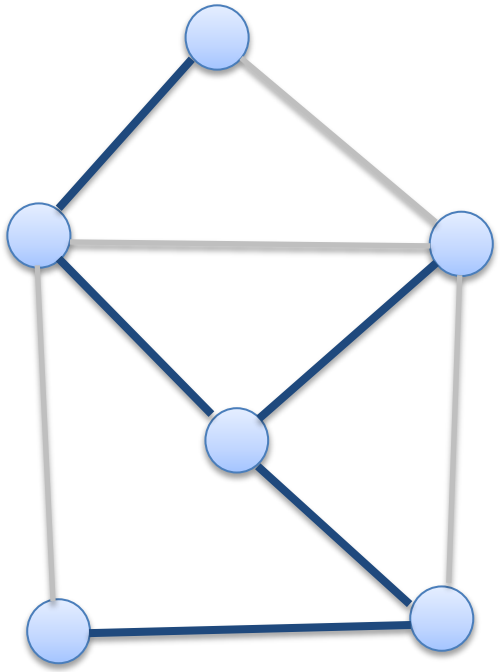
Step 2: Refine the initialized solution by iteratively operating on local cycles in the original graph.



Spanning tree solution as cheap global feature

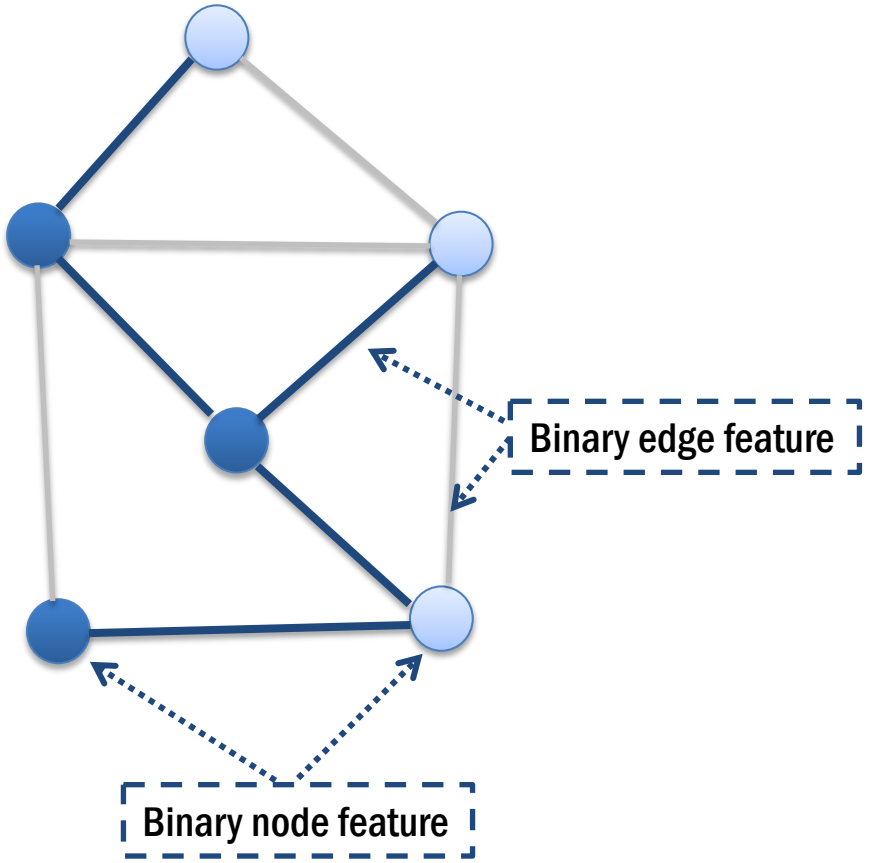
input graph G





spanning tree

optimal solutions y over tree
with dynamic programming

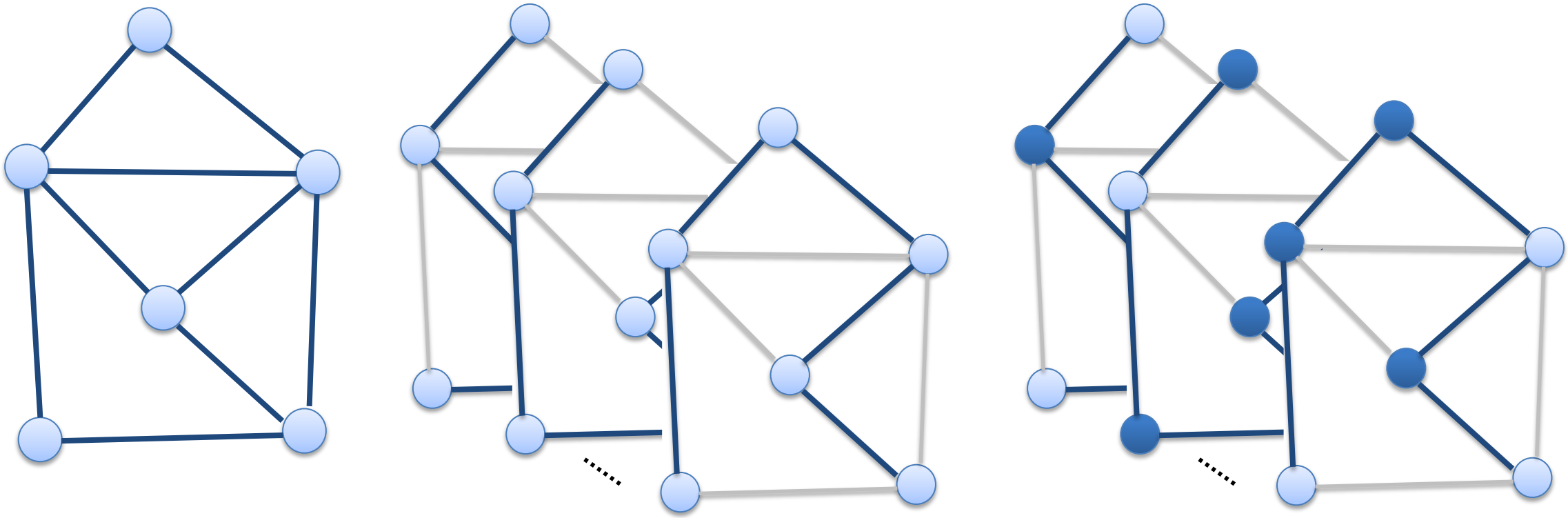


Multiple spanning trees to multiple features

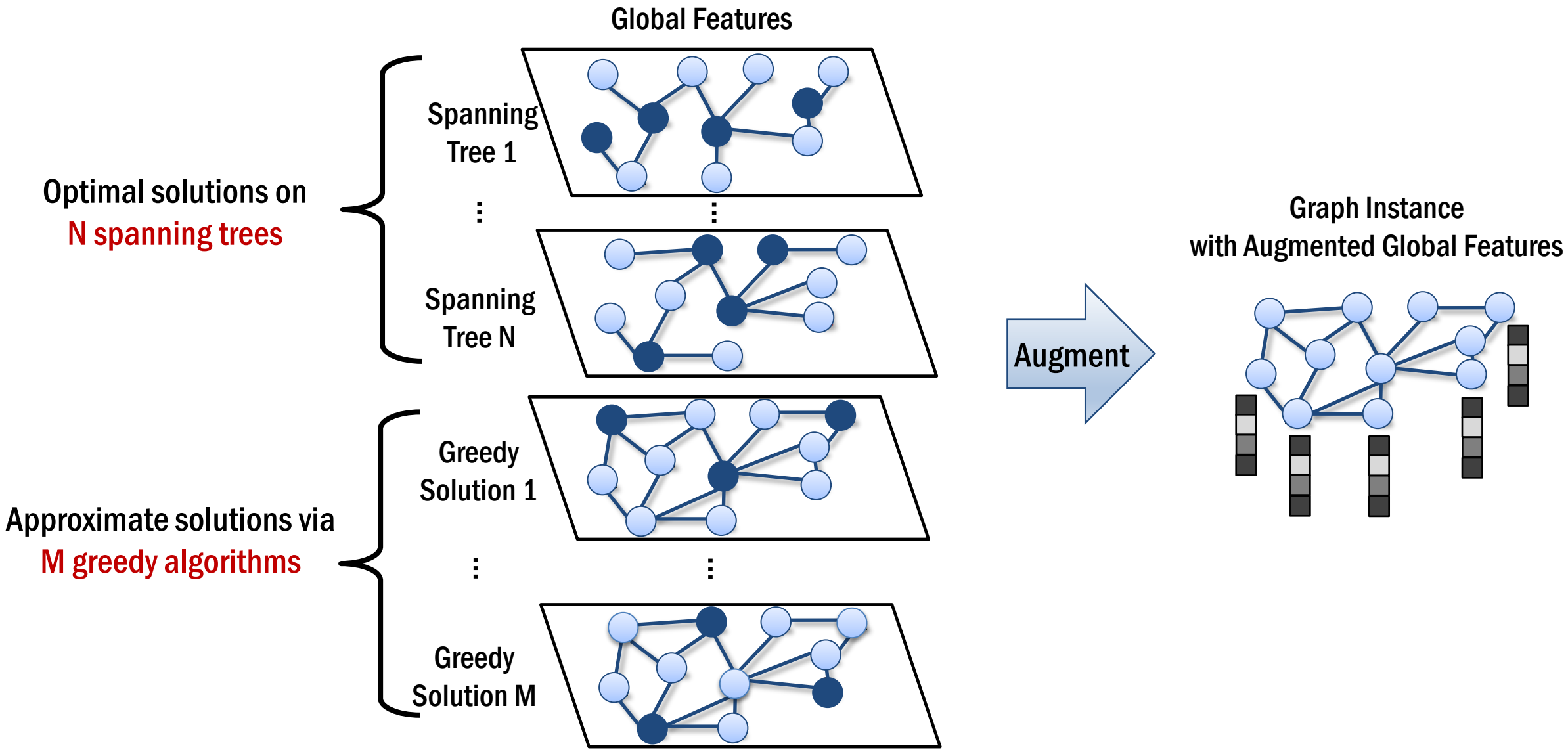
Input graph G

Spanning trees $T(1), \ldots, T(n)$

Optimal solutions over trees $y^{T(1)}, \ldots, y^{T(n)}$



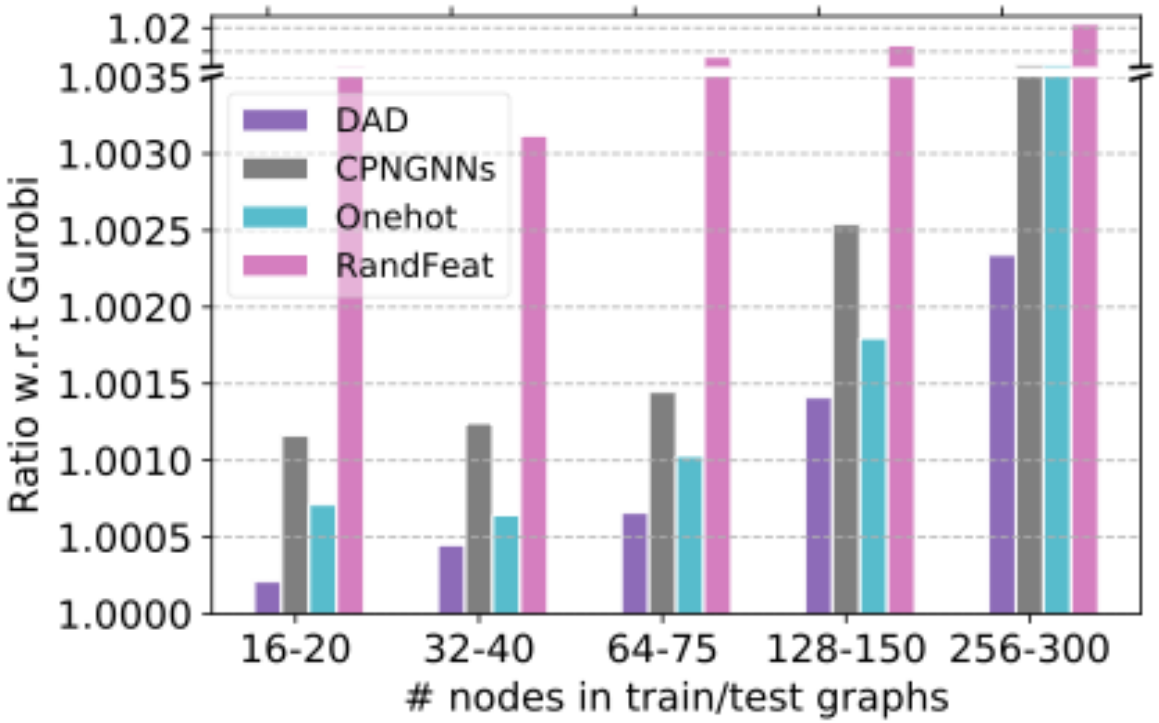
Overall search space design



Better learned algorithms with global information

- Comparison of our features to other features
  - Random features
  - Random one-hot encoding
  - Port Numbering + Weak 2-coloring

Our approach is consistently better

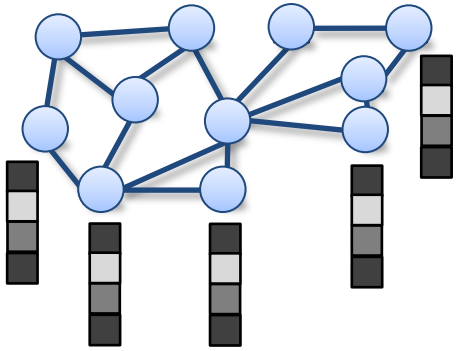


Learn Distributed Local Algorithms with GNN

Supervised

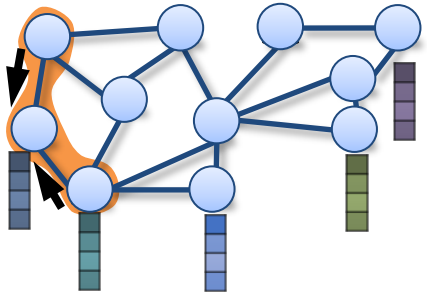
- For each graph G, a solution y^* is obtained by running expensive solver
- Learn GNN-based algorithm which can imitate y^* but runs much faster

Graph Instance G
with Global Node Features X
and Global Edge Features Z



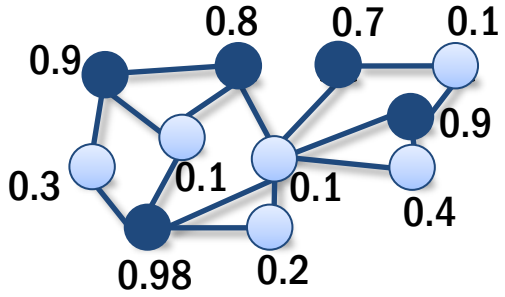
T-hop GNN

Node embeddings
$\{h_i^T\}_{i \in V}$



output layer

Node selection probability
$p_i = \text{Output}\left(h_i^{(T)}\right)$

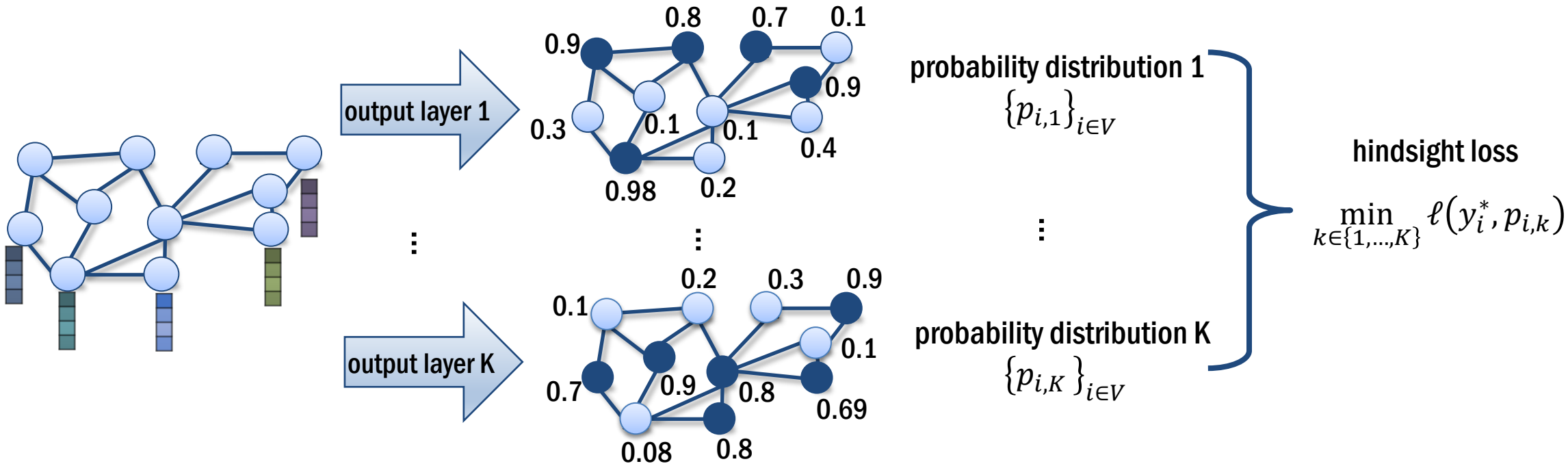


Selection by y^*

Supervised

- Challenge: "mode-averaging" problem.
  - There exists symmetries in the solutions.
  - For example, y^* and $1 - y^*$ are representing the same cut in the Max-Cut problem.

- Solution: generate K distributions and apply hindsight loss



Unsupervised

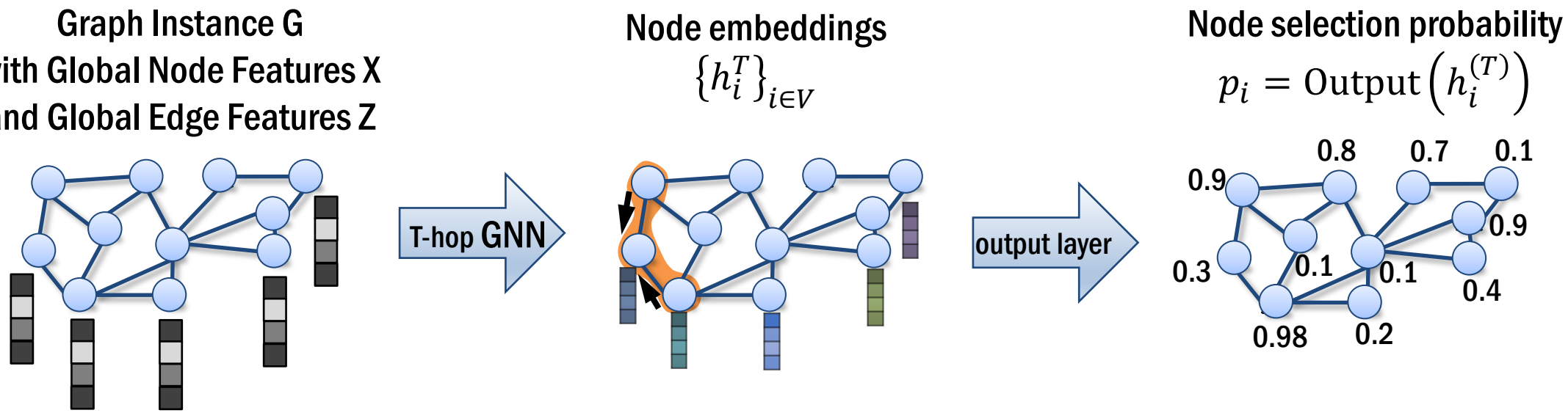
- Many graph problems can be formulated as integer programming (IP) problems:

$$\min_{y \in \{0,1\}^{|V|}} f(Y; G) \quad \text{subject to} \quad g_i(y; G) \leq 0 \quad \text{for } i = 1, \ldots, l$$

- Construct unsupervised training loss based on optimization objective f and constraints g

$$L_U(p, G) \coloneqq E[f(Y; G)] + \beta \cdot P[g_i(Y; G) \leq 0]$$

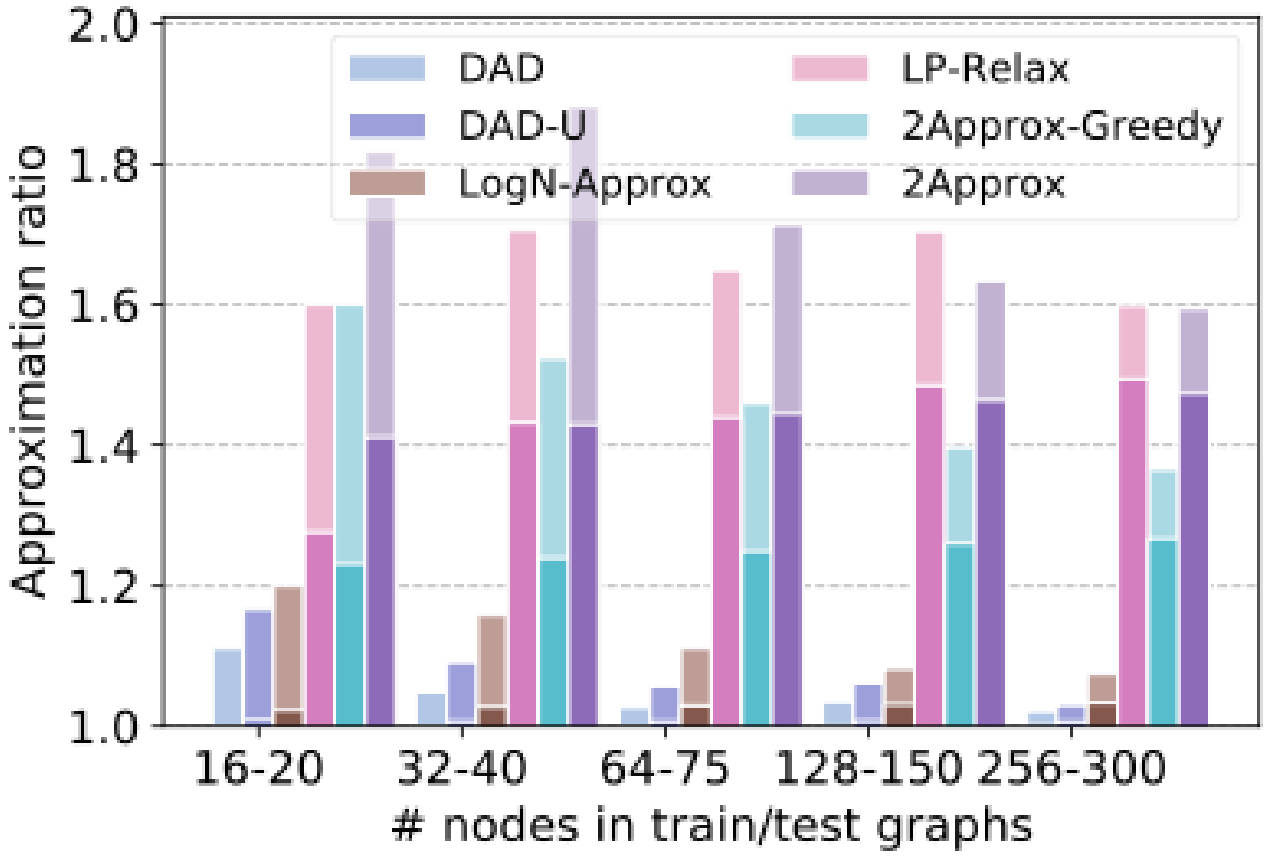
where $Y \sim \text{Bernoulli}(p)$



Better approximation than classical algorithms

- Average and Max approximation ratio
  - Train/test on Barabasi Albert random graphs, with up to 300 nodes
  - Compare to polynomial algorithms, as well as LP/SDP
  - Use either supervised or Unsupervised learning

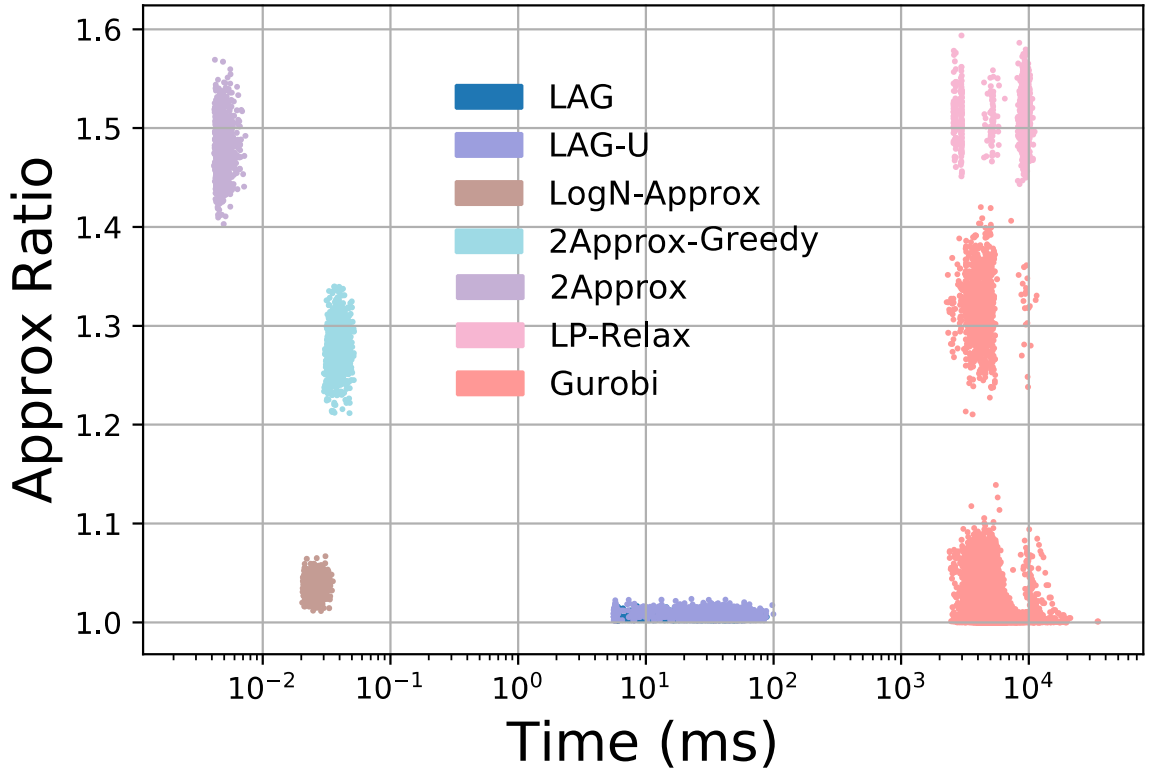
Minimum Vertex Cover



Better time-solution trade-off

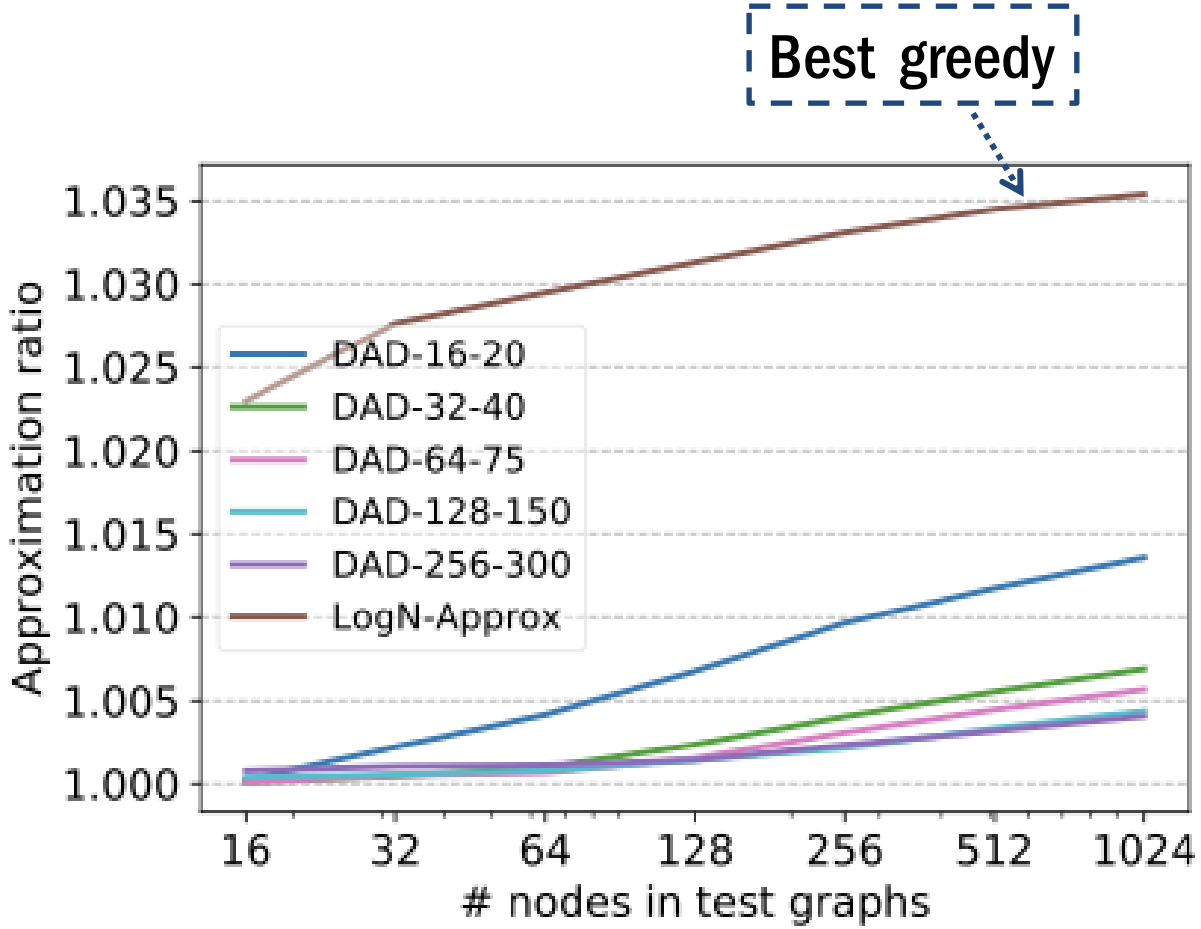
- How much time X is needed to get Y approximation ratio
  - Each dot is a (time, solution quality) pair
  - Graphs with 512 nodes

Minimum Vertex Cover



Extrapolation

- train on small graphs
- test on graphs up to 1024 nodes

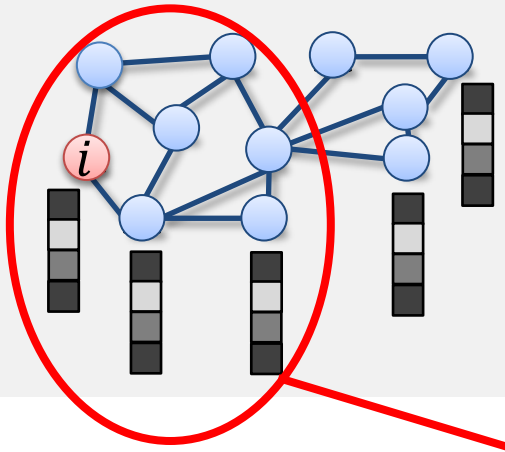


Minimum Vertex Cover on Barabasi Albert random graphs

Explain the Learned GNN Algorithms

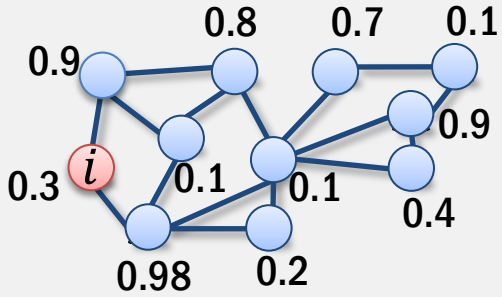
Explainer

Learned Algorithm:



T-hop subgraph

GNN → Output Layer



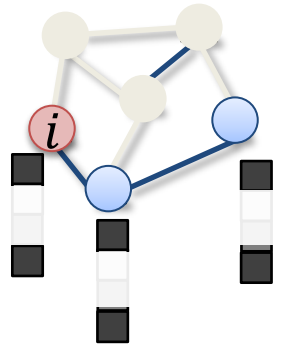
Node selection probability

$$p_i = \text{Output}\left(h_i^{(T)}\right)$$

$$Y_i \sim \text{Bernoulli}(p_i)$$

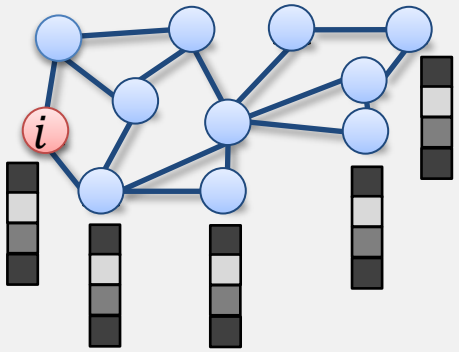
(V_S^i, E_S^i, S_S^i)

Selected Graph Structure & Features



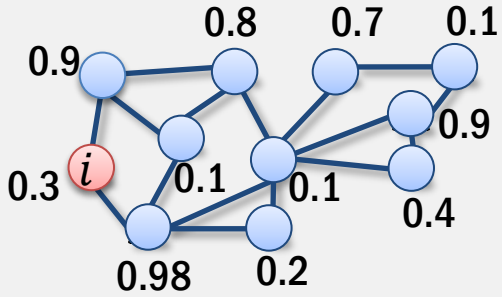
Explainer architecture

Learned Algorithm:



GNN

Output Layer



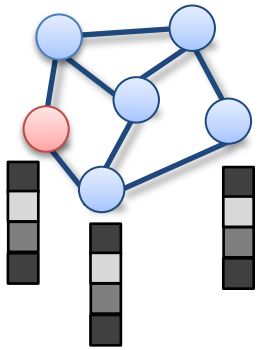
Node selection probability

$$p_i = \text{Output}\left(h_i^{(T)}\right)$$

$$Y_i \sim \text{Bernoulli}(p(i))$$

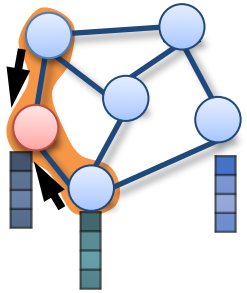
Explainer:

T-hop subgraph



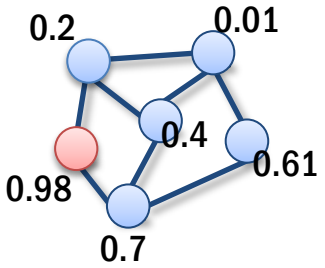
GNN

Node embeddings



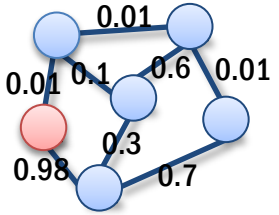
Node/Edge/Feature Scores

MLP1



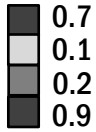
Top k1

MLP2



Top k2

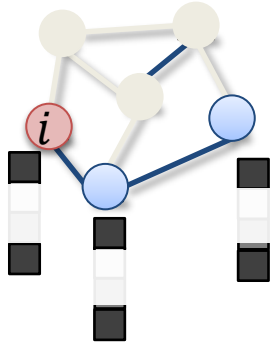
MLP3



Top k3

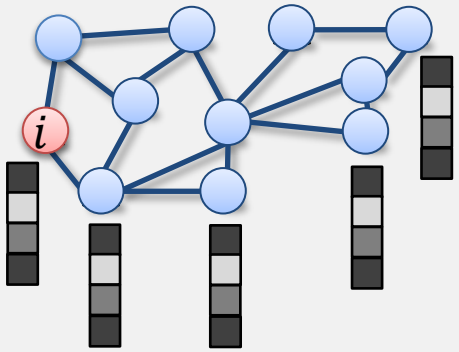
(V_S^i, E_S^i, S_S^i)

Selected Graph Structure & Features



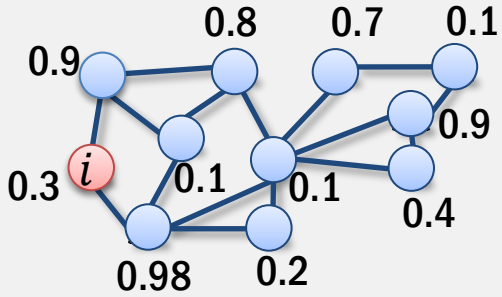
Information theoretic learning

Learned Algorithm:



GNN

Output Layer



Node selection probability

$$p_i = \text{Output}\left(h_i^{(T)}\right)$$

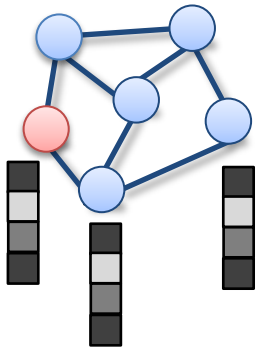
$$Y_i \sim \text{Bernoulli}(p(i))$$

maximize mutual information

$$MI\left(Y_i; (V_S^i, E_S^i, S_S^i)\right)$$

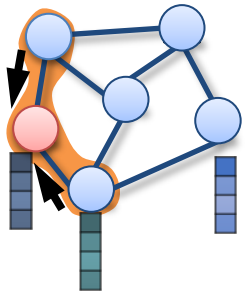
Explainer:

T-hop subgraph



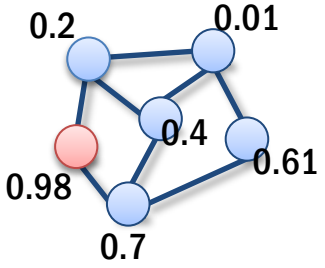
GNN

Node embeddings



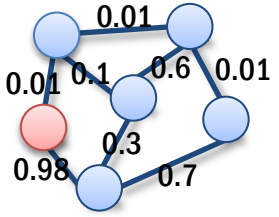
Node/Edge/Feature Scores

MLP1



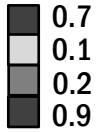
Top k1

MLP2



Top k2

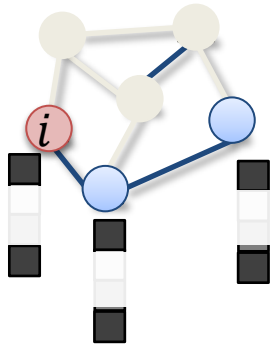
MLP3



Top k3

(V_S^i, E_S^i, S_S^i)

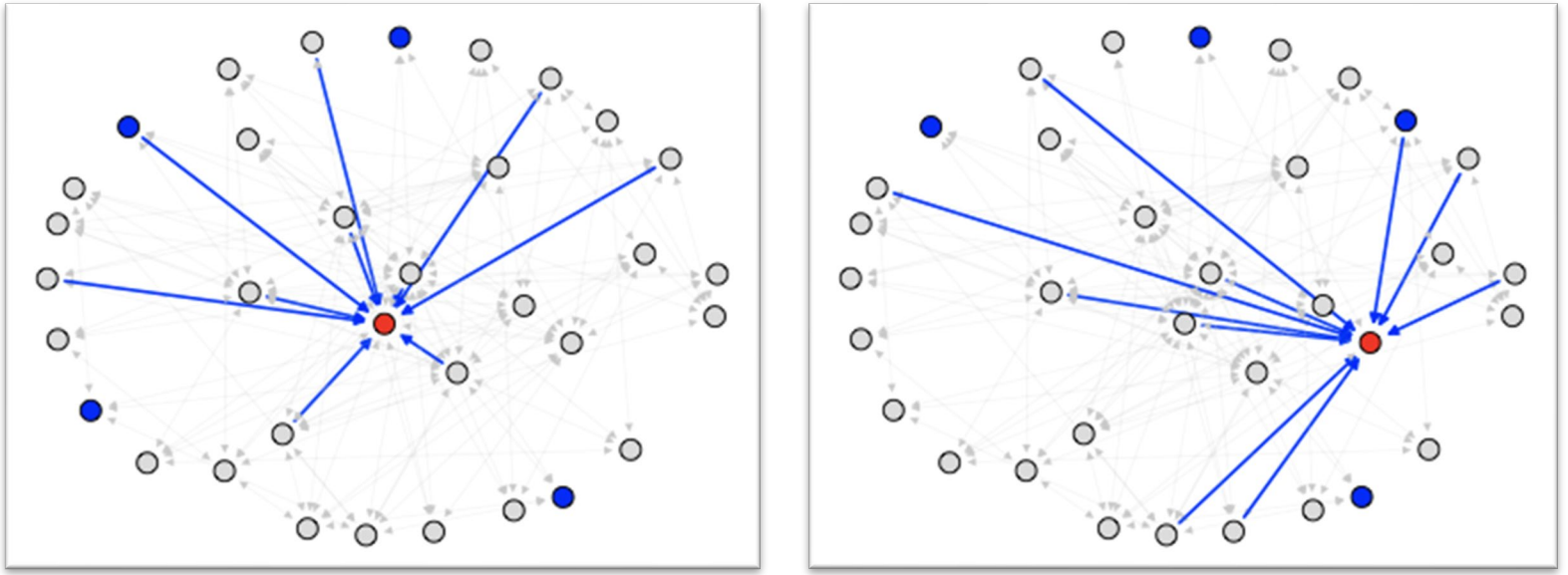
Selected Graph Structure & Features



Discovery of greedy-like behavior

- Explanation setting:
  - limit to 5 nodes and 10 edges to explain each target node

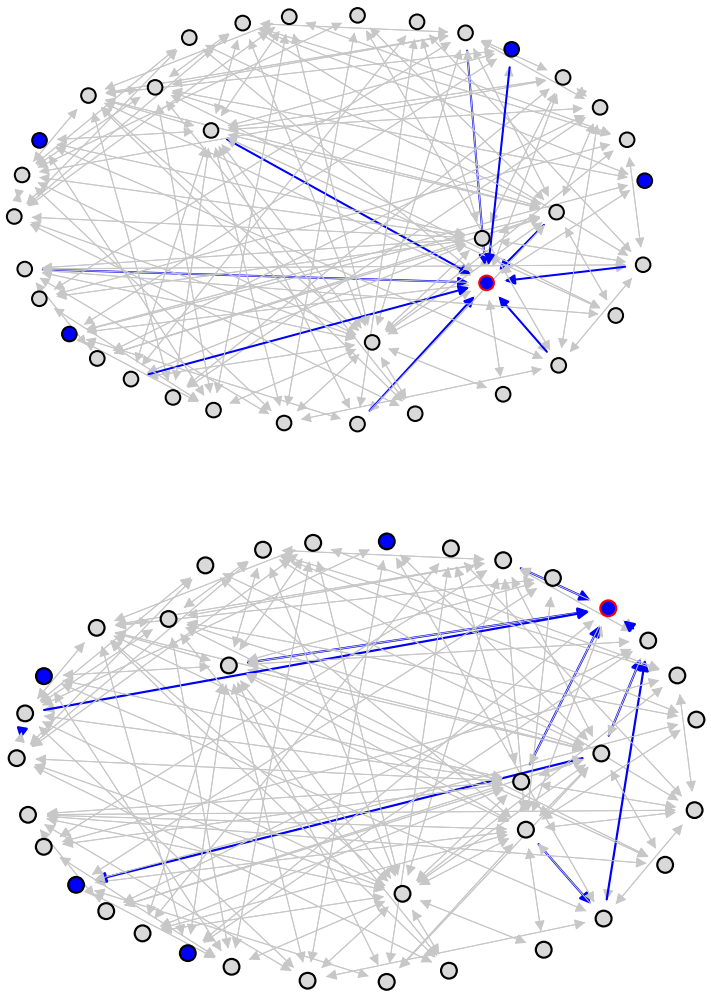
Minimum Vertex Cover

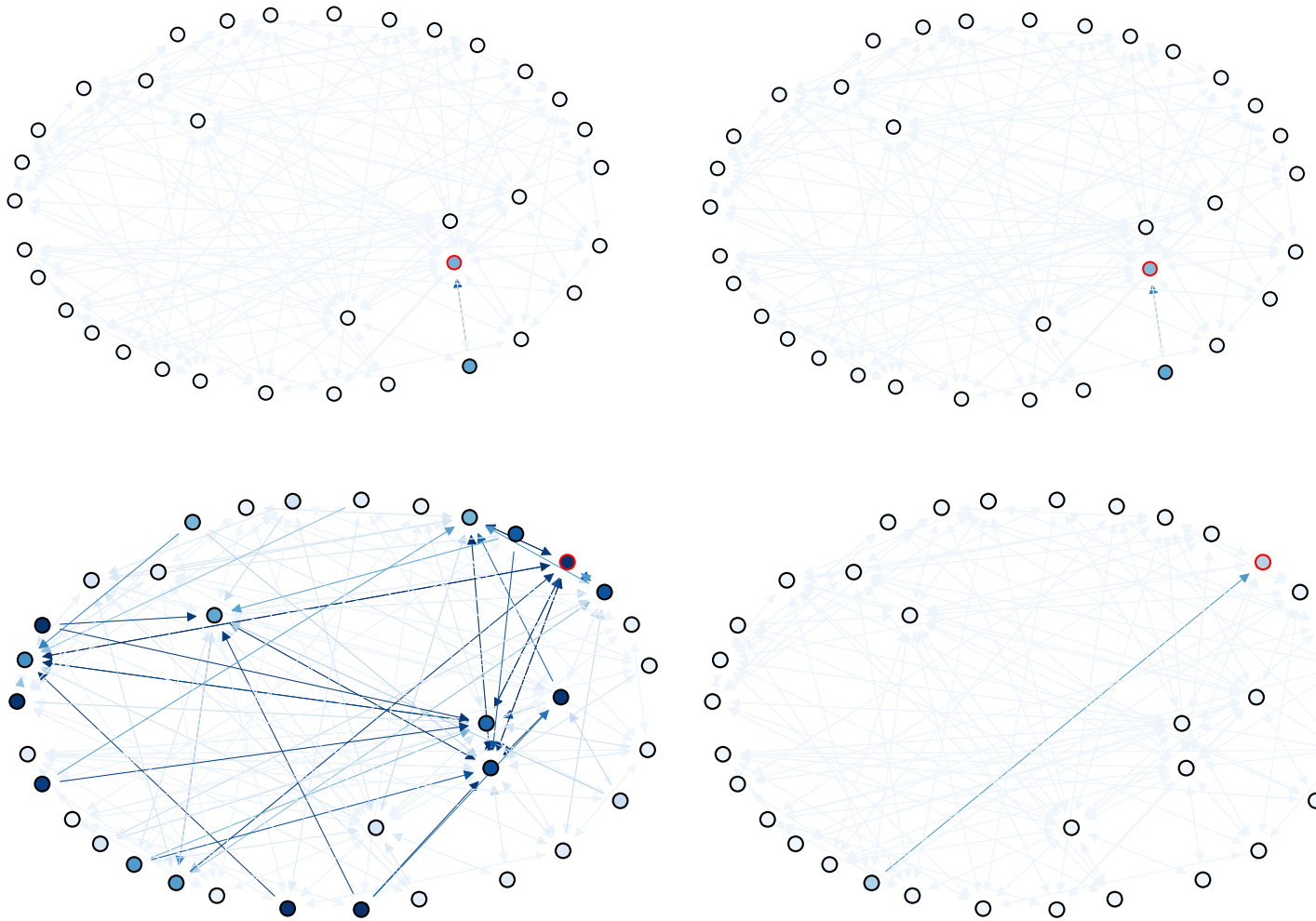


Greedy like behavior on some targets!

- Takeaway:
  - Greedy heuristics are the best performing ones on these tasks
  - GNN understands and learns the meaning of greedy algorithm features

Compare with Continuous Space Explainers



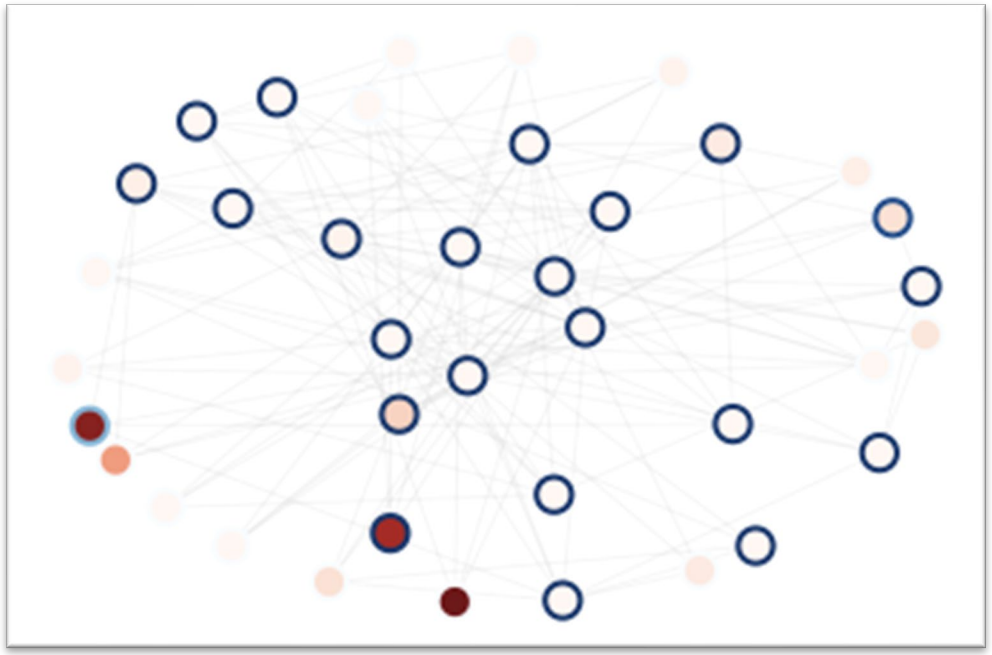


GNNExplainer with different regularization coefficients

Anchor nodes of explanation

- Node color: the darker, the more frequent of being selected for explanation;

Minimum Vertex Cover



- Observation
  - There exists a set of "anchor nodes"
  - Anchor nodes tends to be diverse, regarding their solution probability

- Hypothesis
  - Anchor nodes are like "landmarks" in the graph
  - GNN compares the target node with anchor nodes to make prediction

- Connections: GNN with anchor nodes: Position/distance aware GNNs (You et.al, 2018; Li et.al, 2020)

Comparing feature quality

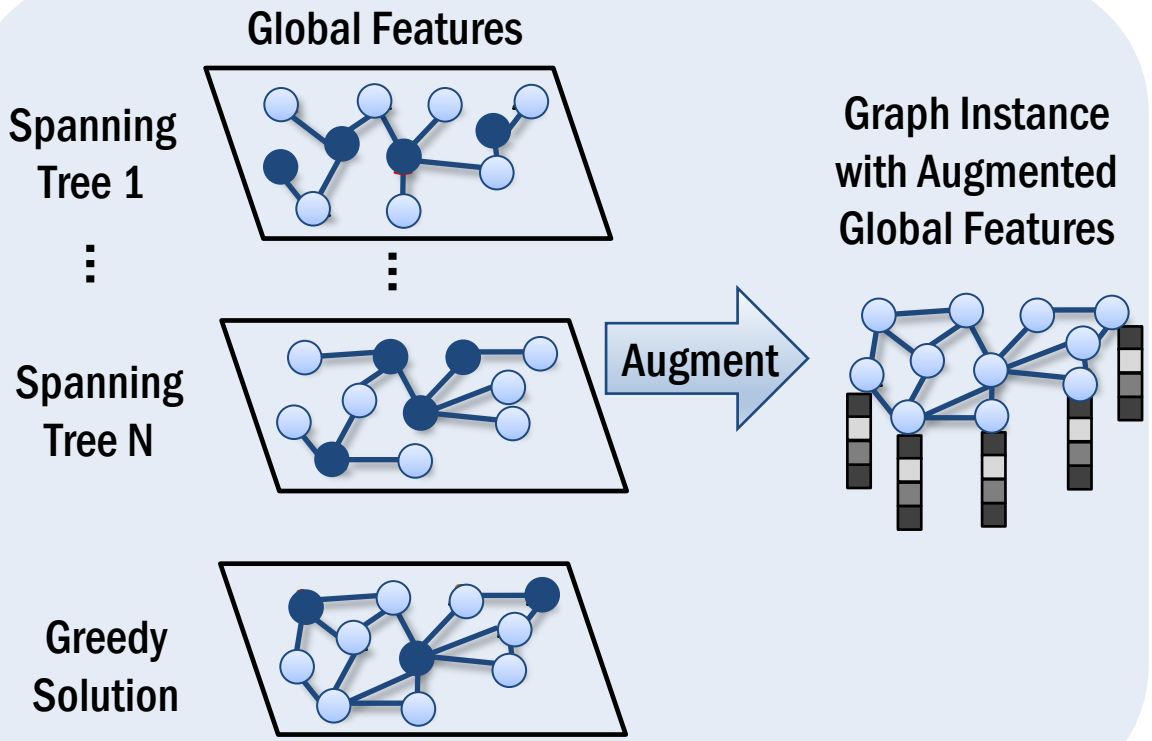
- Explanation setting:
  - Concatenate TreeDP + Greedy + other global features
  - Use our explainer to select most important feature(s)

Table 3. Most important (used in more that 99% of the graphs) features identified by our explainer.

	Features	Most important
MVC	Node/Edge Greedy, TreeDP, Weak 2-color, RandFeat	Node Greedy, TreeDP
MDS	Greedy, TreeDP, Weak 2-color, RandFeat	Greedy
Max-Cut	Greedy, min/akpw/max-TreeDP, Weak 2-color, RandFeat	Greedy, Max-TreeDP, Weak 2-color

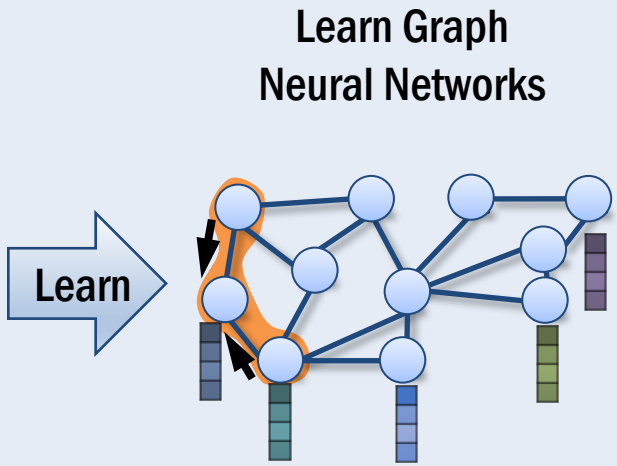
Differentiable Algorithm Discovery (DAD)

1. How to design the search space?



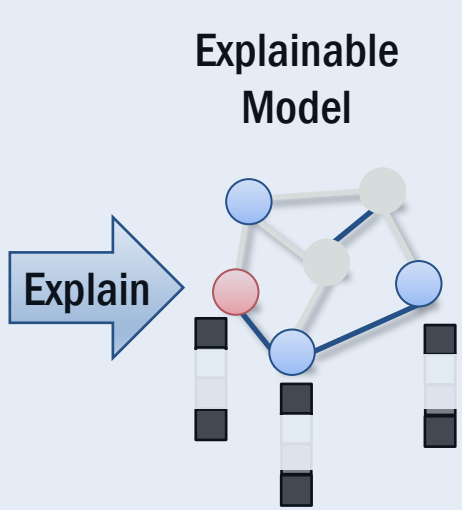
2. How to learn these algorithms?

Supervised, Unsupervised



Supervised
Unsupervised

3. Can we interpret what's discovered?



The End