НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ

Laboratory of Algorithms and Technologies for
Network Analysis

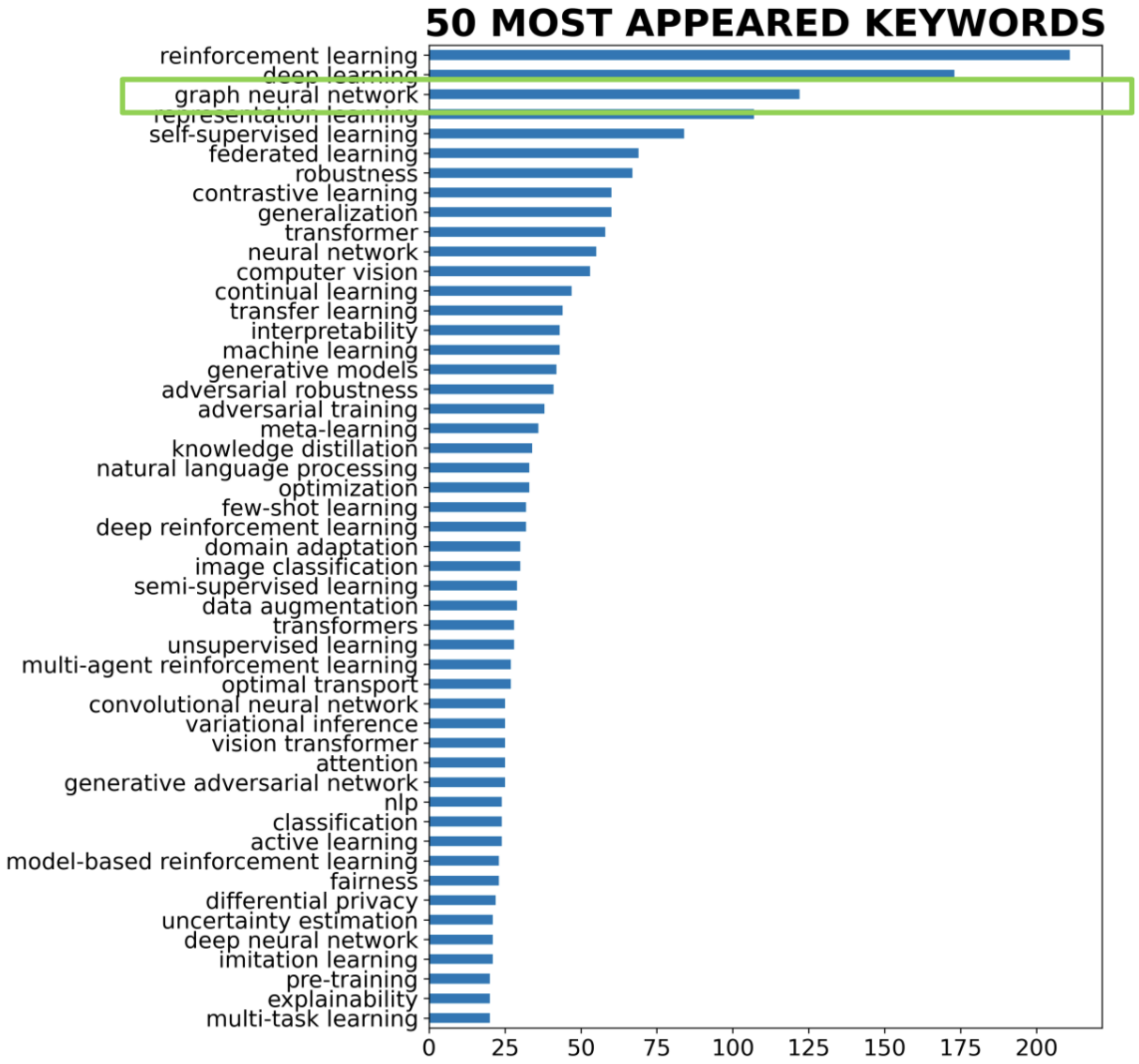
INTRODUCTION TO GRAPH NEURAL NETWORKS WITH APPLICATIONS TO COMBINATORIAL OPTIMIZATION

ORA 2024

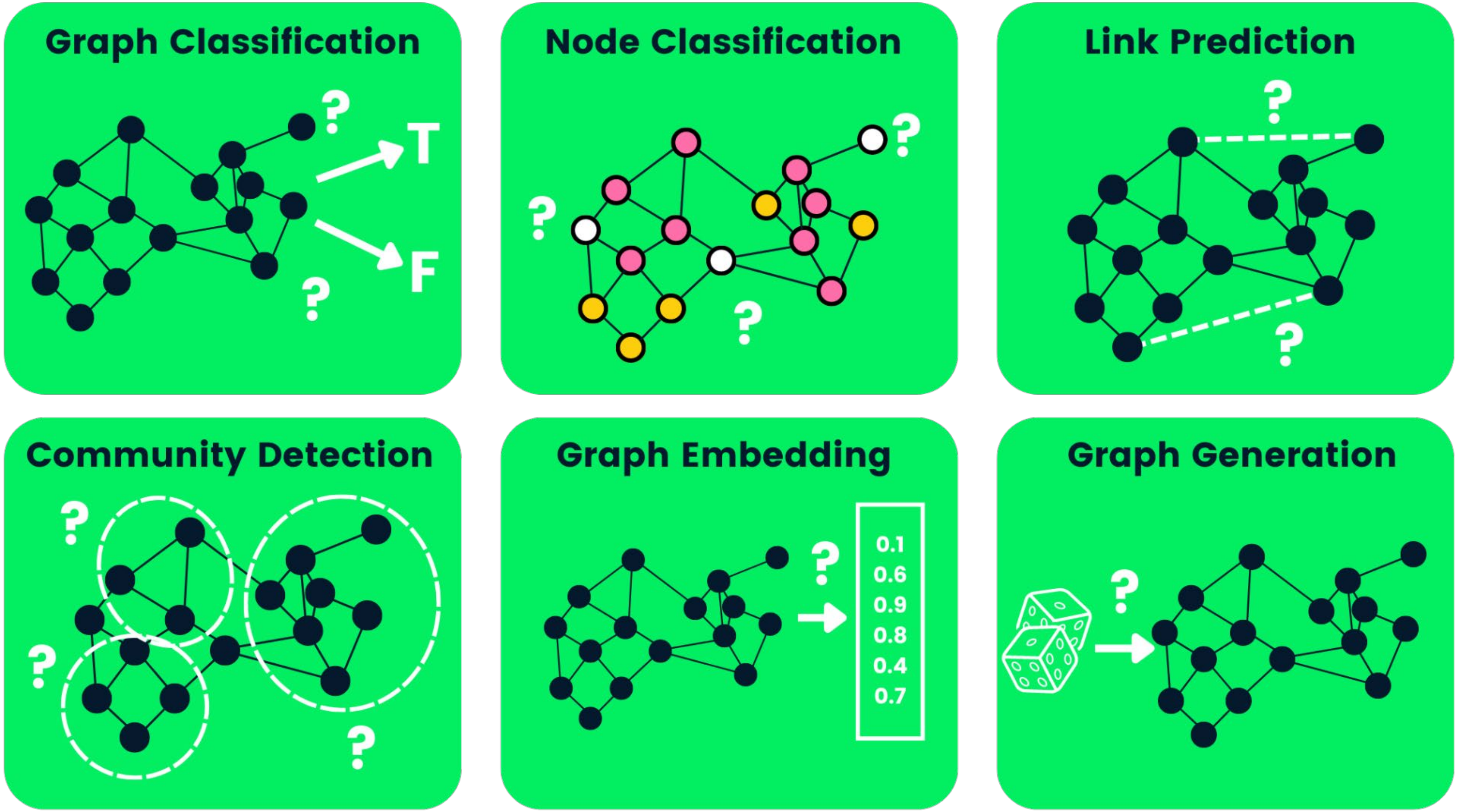
Nizhny Novgorod, 2024

HOT TOPIC?

ICLR 2022 keywords

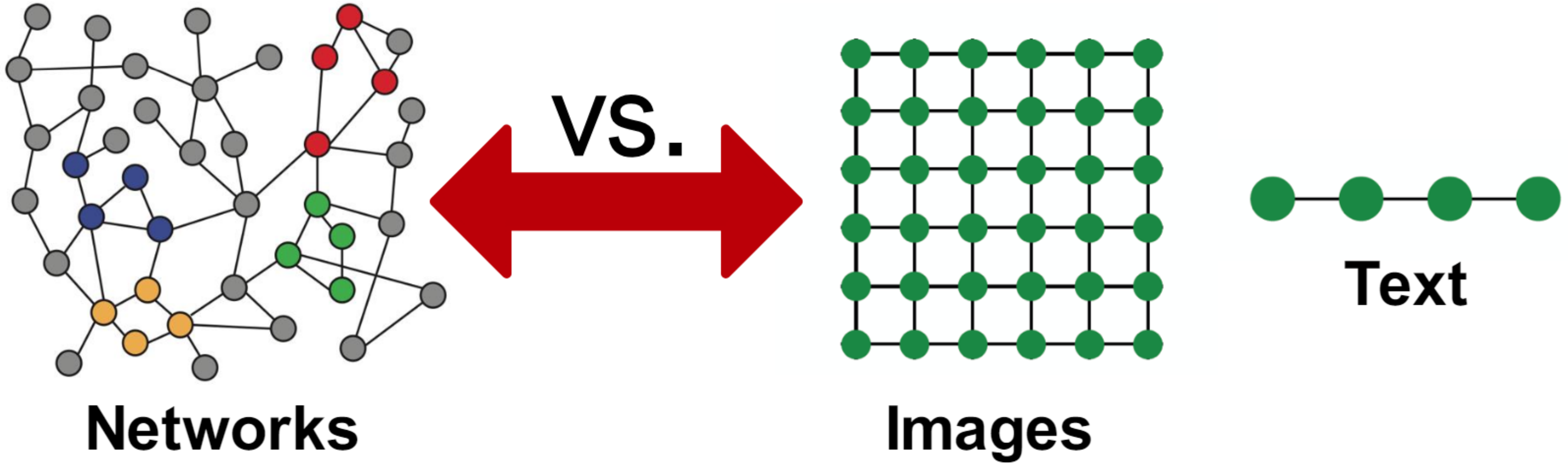


GNN TASKS

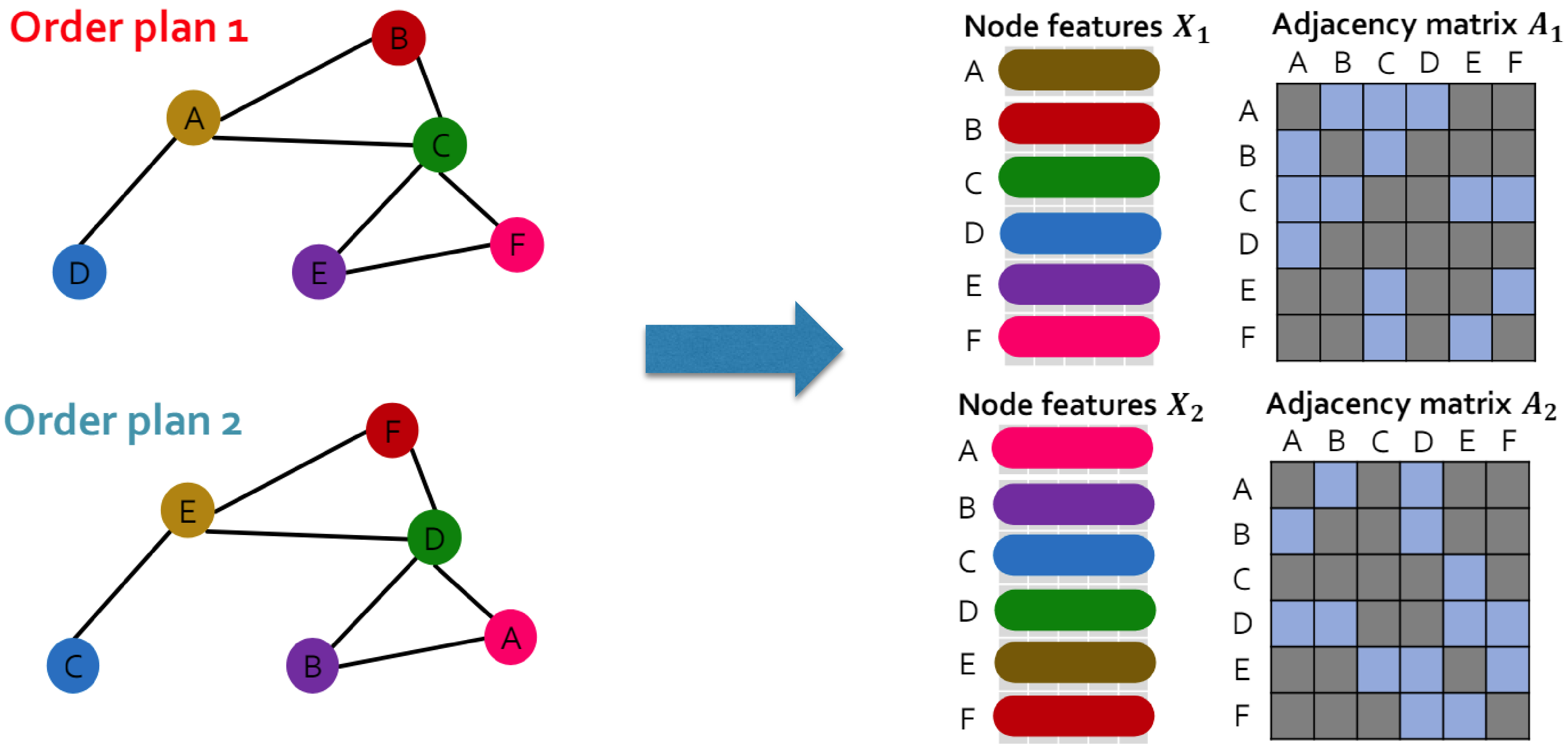

Graph Classification
T
F
Node Classification
Link Prediction
Community Detection
Graph Embedding
0.1
0.6
0.9
0.8
0.4
0.7
Graph Generation

MACHINE LEARNING WITH GRAPHS

- Arbitrary size and complex topological structure (i.e., no spatial locality like grids)
- No fixed node ordering or reference point
- Often dynamic and have multimodal features



PERMUTATION INVARIANCE



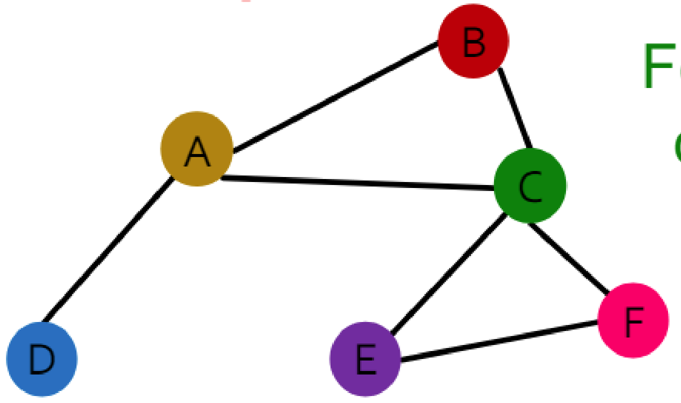
PERMUTATION INVARIANCE

Consider we learn a function f that maps a graph $G = (\boldsymbol{A}, \boldsymbol{X})$ to a vector in $\mathbb{R}^d$ then

$$f(\boldsymbol{A}_1, \boldsymbol{X}_1) = f(\boldsymbol{A}_2, \boldsymbol{X}_2)$$

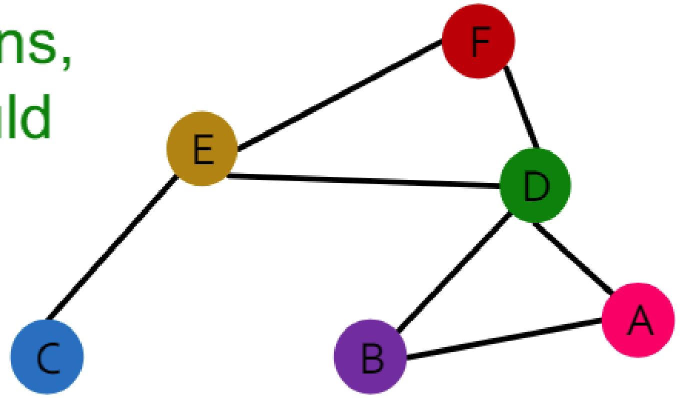
If f computes some graph representation, or, e.g. predicts a class:

Order plan 1: $\boldsymbol{A}_1, \boldsymbol{X}_1$



For two order plans, output of f should be the same!

Order plan 2: $\boldsymbol{A}_2, \boldsymbol{X}_2$



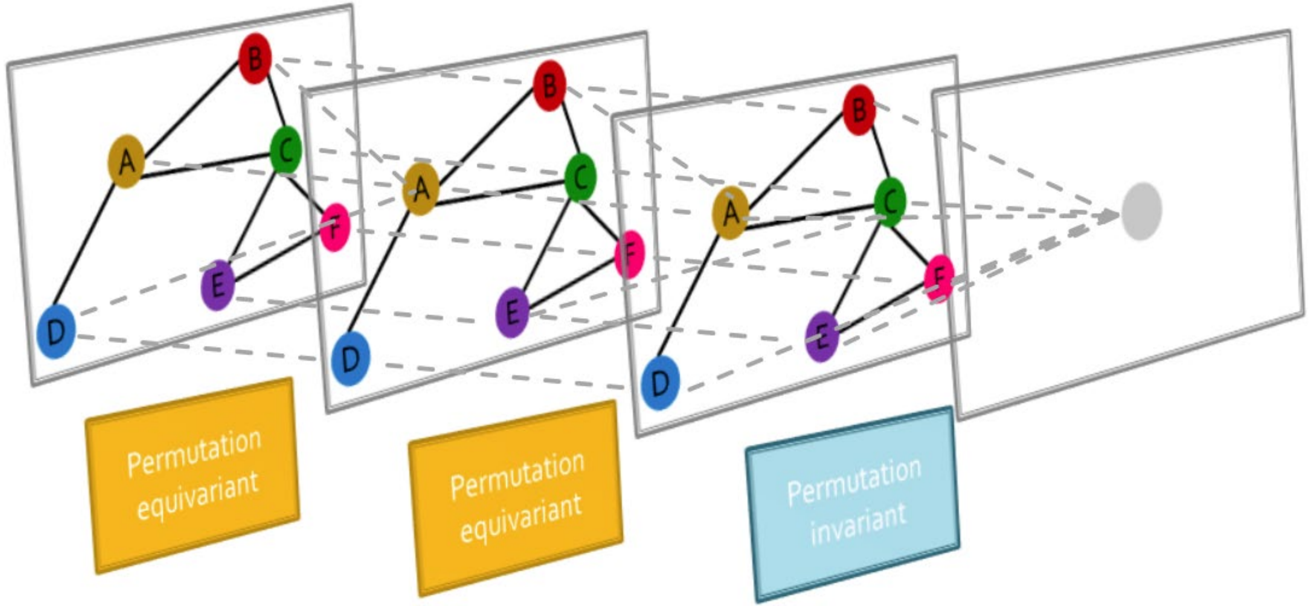
PERMUTATION INVARIANCE/EQUIVARIANCE

Permutation-invariant

$$f(\boldsymbol{A}, \boldsymbol{X}) = f(P\boldsymbol{A}P^T, P\boldsymbol{X})$$

Permutation-equivariant

$$Pf(\boldsymbol{A}, \boldsymbol{X}) = f(P\boldsymbol{A}P^T, P\boldsymbol{X})$$



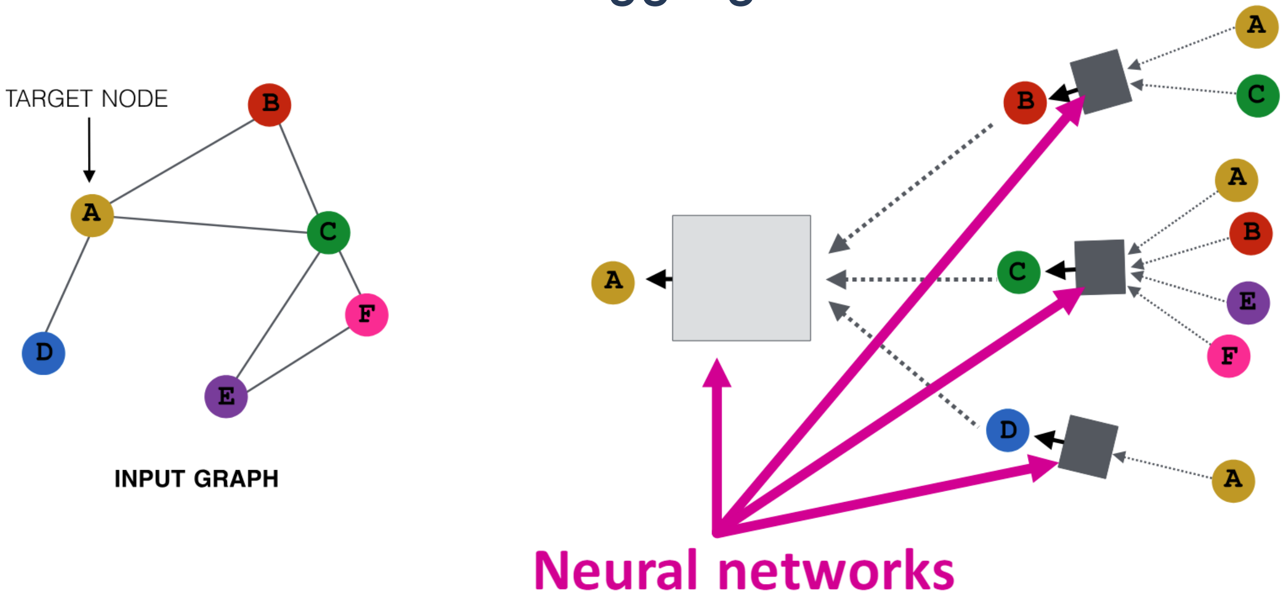
Graph neural networks consist of multiple permutation equivariant / invariant functions.

GENERAL IDEA

Idea: Node's neighborhood defines a computation graph

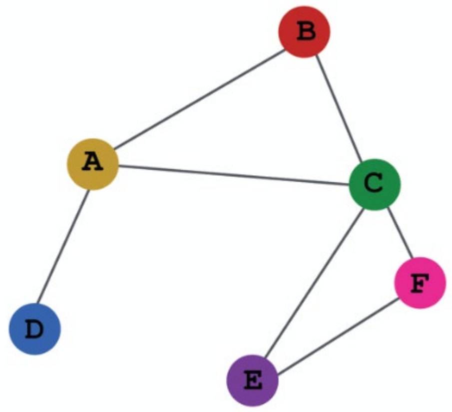
Intuition: Nodes aggregate information from their neighbors using neural networks
Each node defines a computation graph

- Each edge in this graph is a transformation/aggregation function

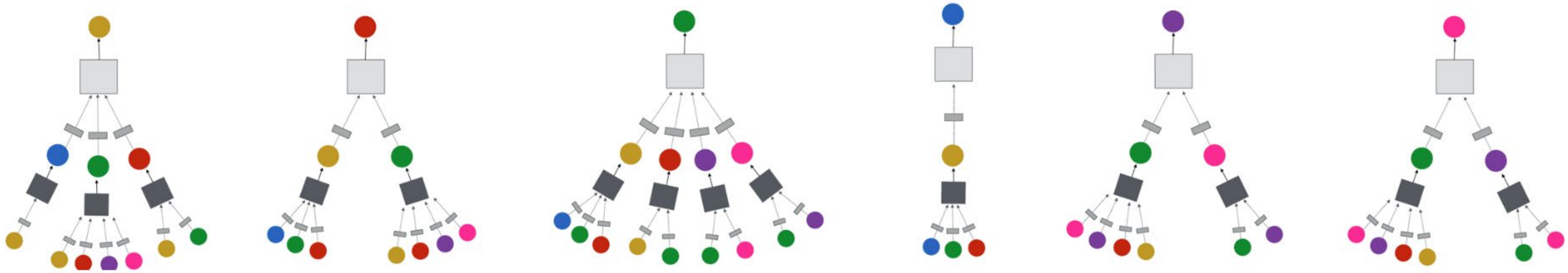


GENERAL IDEA

Intuition: Nodes aggregate information from their neighbors using neural networks



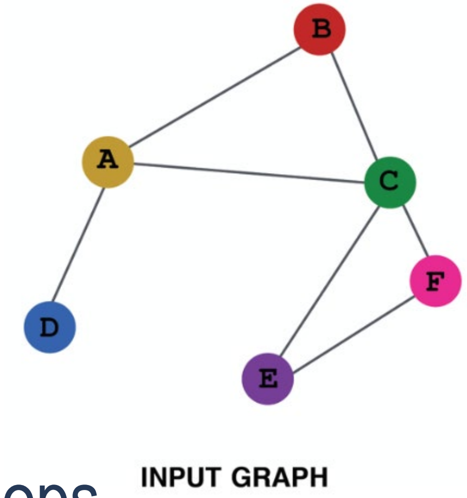
INPUT GRAPH

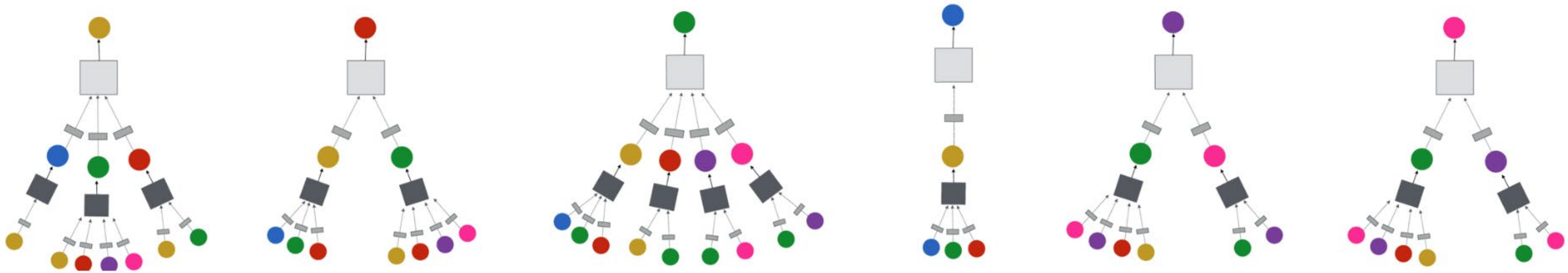


GENERAL IDEA

Model can be of **arbitrary depth**:

- Nodes (edges) have embeddings at each layer
- Layer-0 embedding of node v is its input feature, x_v
- Layer-k embedding gets information from nodes that are k hops away

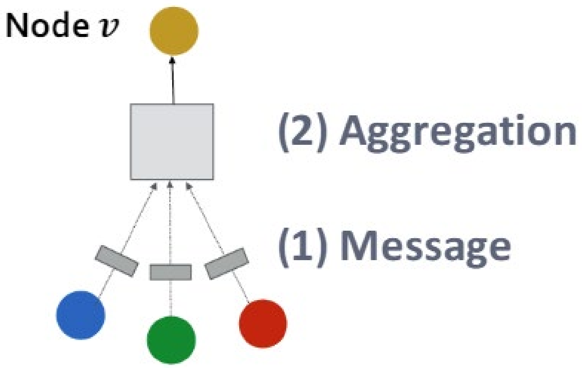


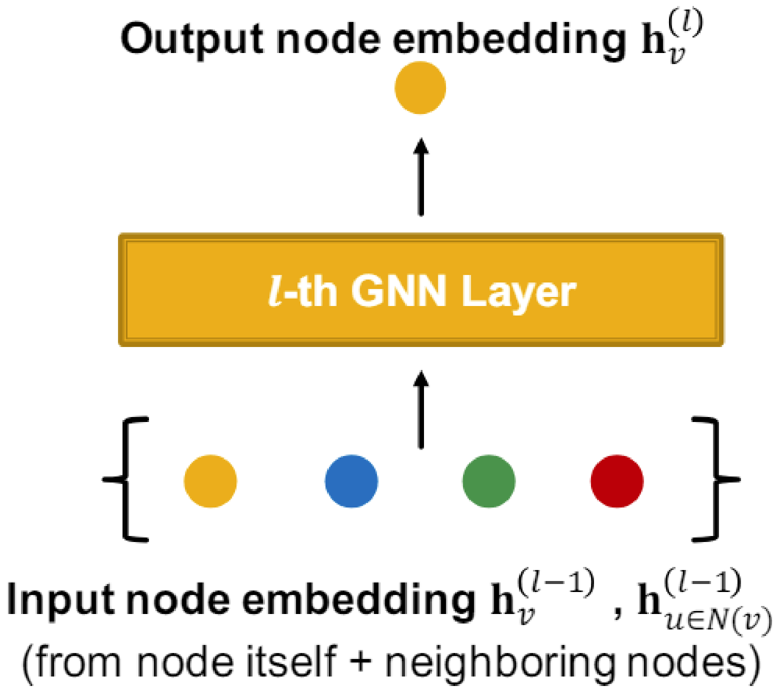


A SINGLE LAYER

Idea of a GNN Layer:

- Compress a set of vectors into a single vector
- Two-step process:
  - (1) **Message**
  - (2) **Aggregation**

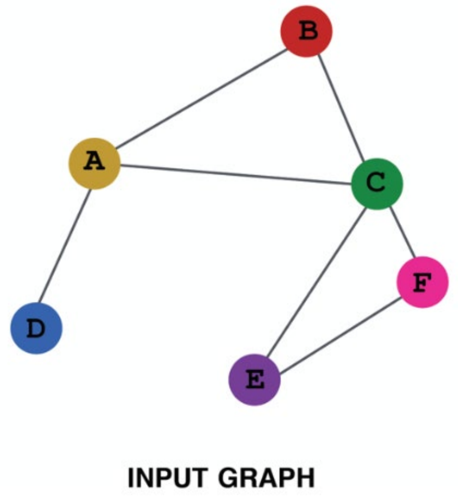


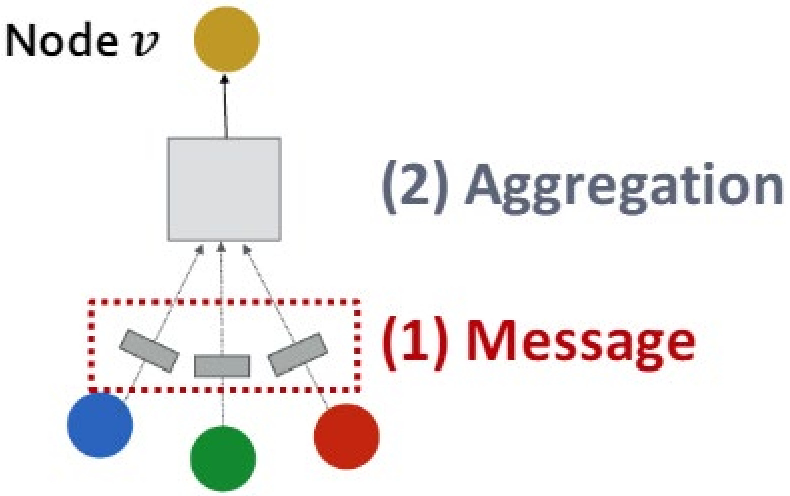


MESSAGE COMPUTATION

Message function: $$\mathbf{m}_u^{(l)} = \mathrm{MSG}^{(l)}\left(\mathbf{h}_u^{(l-1)}\right)$$

- **Intuition**: Each node will create a message, which will be sent to other nodes later
- **Example**: A Linear layer $\mathbf{m}_u^{(l)} = \mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}$



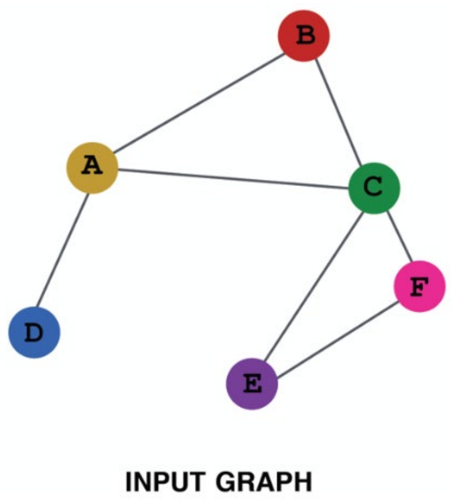


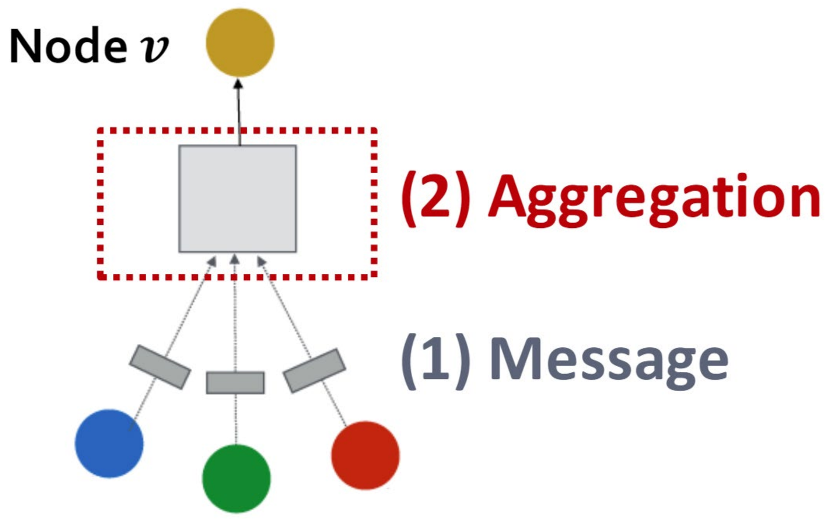
MESSAGE AGGREGATION

- **Intuition**: Each node will aggregate the messages from node v's neighbors

$$\mathbf{h}_v^{(l)} = \mathrm{AGG}^{(l)}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}\right)$$

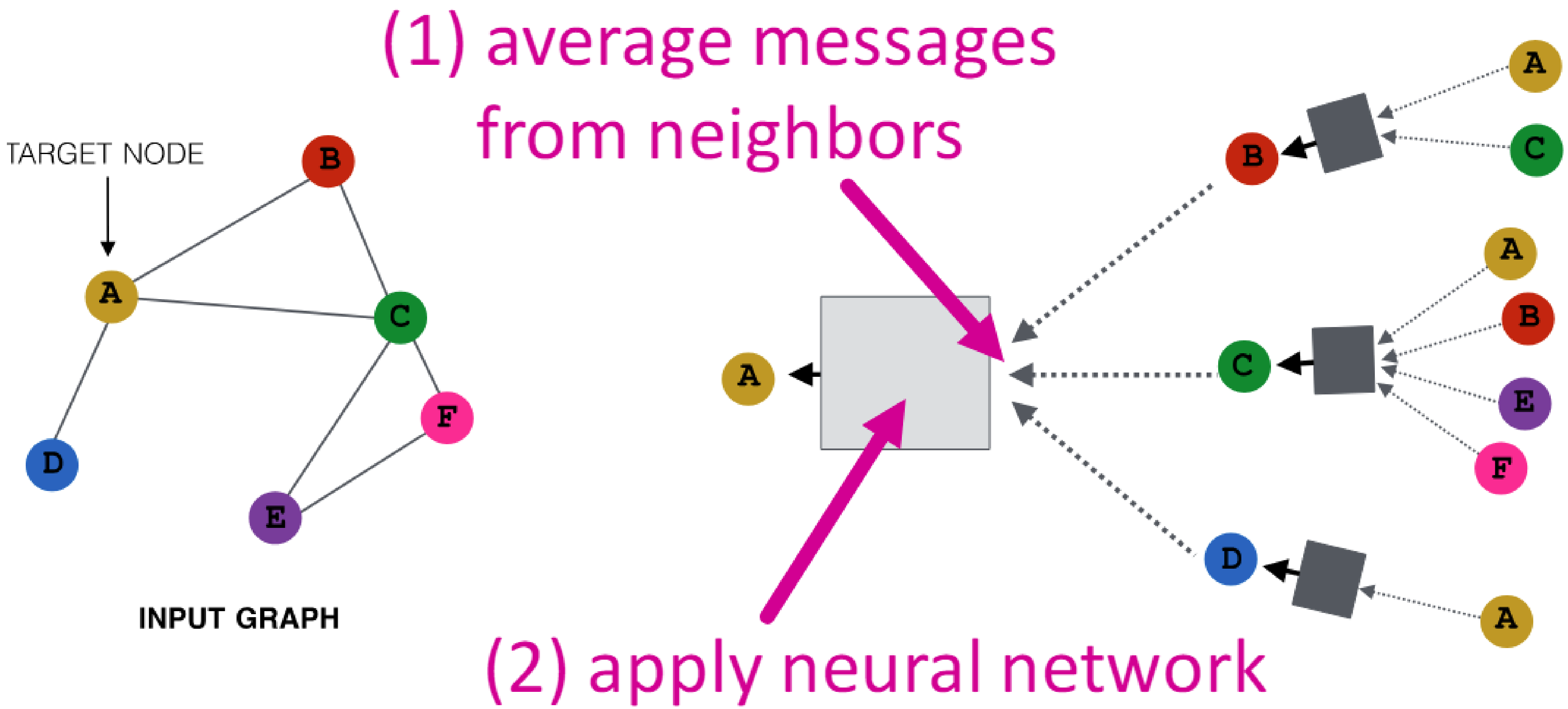
- **Example**: Sum(·), Mean(·) or Max(·) aggregator





BASIC LAYER (GRAPH CONVOLUTIONAL NETWORKS)

Basic approach: Average neighbor messages and apply a neural network



BASIC LAYER (GCN)

Basic approach: Average neighbor messages and apply a neural network

Initial 0-th layer embeddings are equal to node features

$$\boxed{h_v^0 = x_v}$$

embedding of v at layer k

$$h_v^{(k+1)} = \boxed{\sigma}\left(W_k \boxed{\sum_{u \in N(v)} \frac{h_u^{(k)}}{|N(v)|}} + B_k \boxed{h_v^{(k)}}\right), \forall k \in \{0, \ldots, \boxed{K} - 1\}$$

$$\boxed{z_v} = h_v^{(K)}$$

Embedding after K layers of neighborhood aggregation

Non-linearity (e.g., ReLU)

Average of neighbor's previous layer embeddings

Total number of layers

Notice summation is a permutation invariant pooling/aggregation.

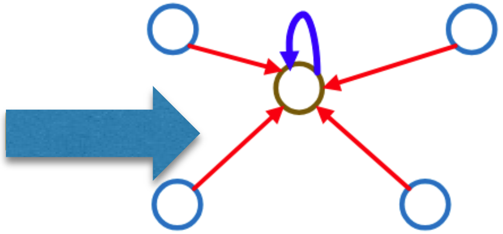
BASIC LAYER (GCN)

Basic approach: Average neighbor messages and apply a neural network

In matrix form:

$$H^{(k+1)} = \sigma(\tilde{A} H^{(k)} W_k^{\mathrm{T}} + H^{(k)} B_k^{\mathrm{T}})$$

where $\quad \tilde{A} = D^{-1} A$

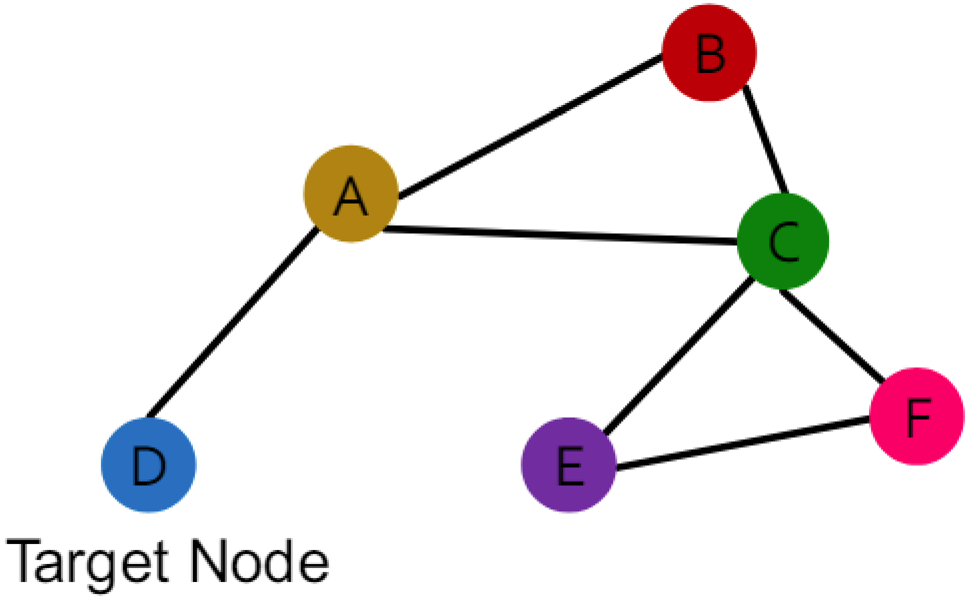


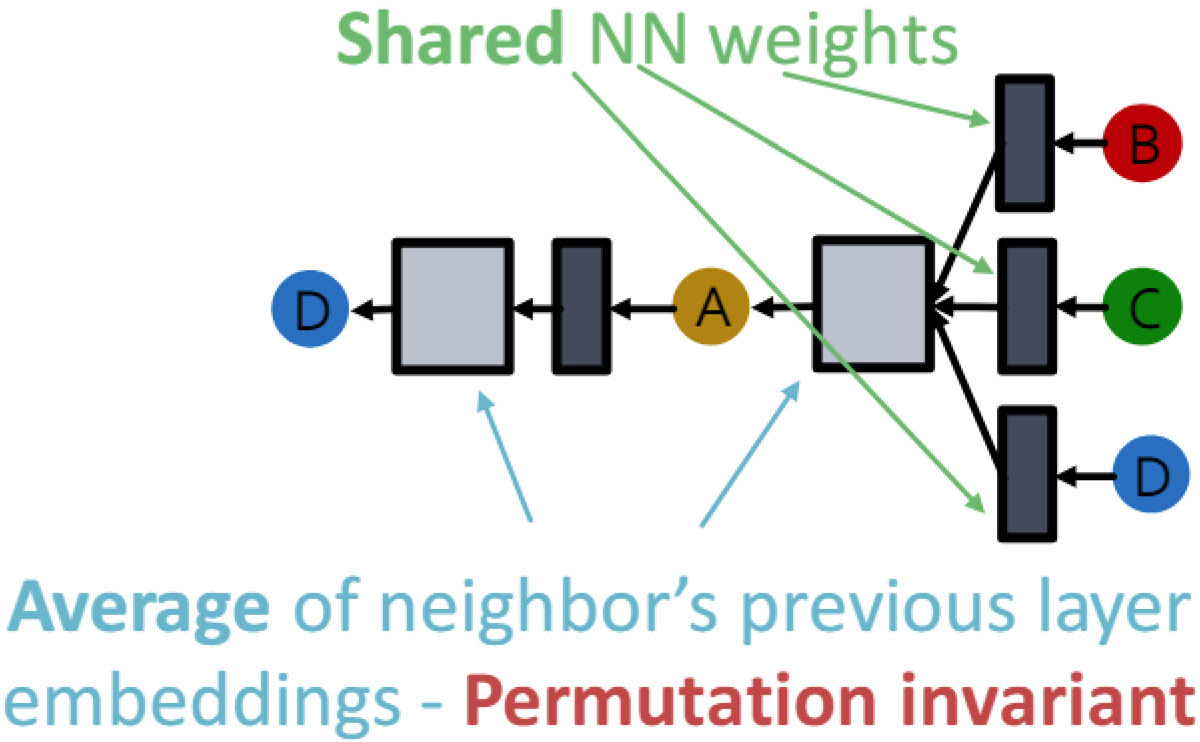
$$H^{(k)} = [h_1^{(k)} \dots h_{|V|}^{(k)}]^T$$

- Red: neighborhood aggregation
- Blue: self transformation

BASIC LAYER (GCN)

Given a node, the GCN that computes its embedding is **permutation invariant**





GRAPHSAGE

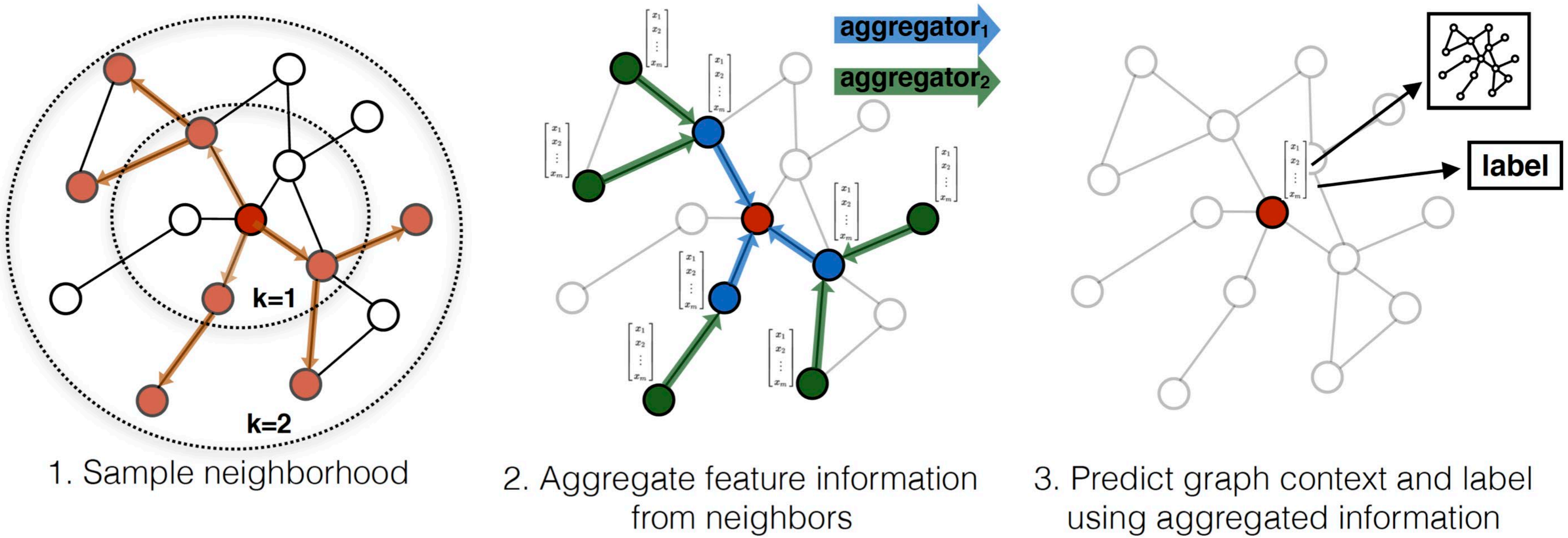
$$\mathbf{h}_v^{(l)} = \sigma\left(\mathbf{W}^{(l)} \cdot \mathrm{CONCAT}\left(\mathbf{h}_v^{(l-1)}, \mathrm{AGG}\left(\left\{\mathbf{h}_u^{(l-1)}, \forall u \in N(v)\right\}\right)\right)\right)$$

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

- Message is computed within the AGG $(\cdot)$
- Two-stage aggregation
  - Stage 1: Aggregate from node neighbors
    $$\mathbf{h}_{N(v)}^{(l)} \leftarrow \mathrm{AGG}\left(\left\{\mathbf{h}_u^{(l-1)}, \forall u \in N(v)\right\}\right)$$
  - Stage 2: Further aggregate over the node itself
    $$\mathbf{h}_v^{(l)} \leftarrow \sigma\left(\mathbf{W}^{(l)} \cdot \mathrm{CONCAT}(\mathbf{h}_v^{(l-1)}, \mathbf{h}_{N(v)}^{(l)})\right)$$

Inductive Representation Learning on Large Graphs. W.L. Hamilton, R. Ying, and J. Leskovec *arXiv:1706.02216 [cs.SI]*, 2017.

GRAPHSAGE



Inductive Representation Learning on Large Graphs. W.L. Hamilton, R. Ying, and J. Leskovec *arXiv:1706.02216 [cs.SI]*, 2017.

GRAPH ATTENTION NETWORKS

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

- Not all node's neighbors are equally important
- The **attention** $\boldsymbol{\alpha_{vu}}$ focuses on the important parts of the input data and fades out the rest
  - **Idea**: the NN should devote more computing power on that small but important part of the data.
  - Which part of the data is more important depends on the context and is learned through training.

P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph Attention ´ Networks. ICLR, 2018.

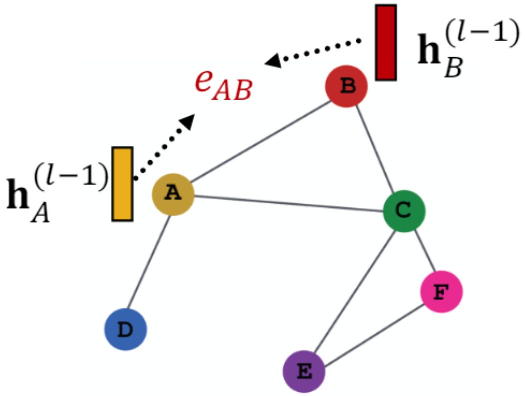
ATTENTION MECHANISM (1)

Let $\boldsymbol{\alpha_{vu}}$ be computed as a byproduct of an **attention mechanism** $\boldsymbol{a}$:

- Let $\boldsymbol{a}$ compute attention coefficients $\boldsymbol{e_{vu}}$ across pairs of nodes $\boldsymbol{v}$, $\boldsymbol{u}$ based on their messages:

$$e_{vu} = a(\mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}, \mathbf{W}^{(l)}\boldsymbol{h}_v^{(l-1)})$$

  - **$\boldsymbol{e_{vu}}$ indicates the importance of $\boldsymbol{u}'$s message to node $\boldsymbol{v}$**



$$e_{AB} = a(\mathbf{W}^{(l)}\mathbf{h}_A^{(l-1)}, \mathbf{W}^{(l)}\mathbf{h}_B^{(l-1)})$$

P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph Attention ´ Networks. ICLR, 2018.

ATTENTION MECHANISM (2)

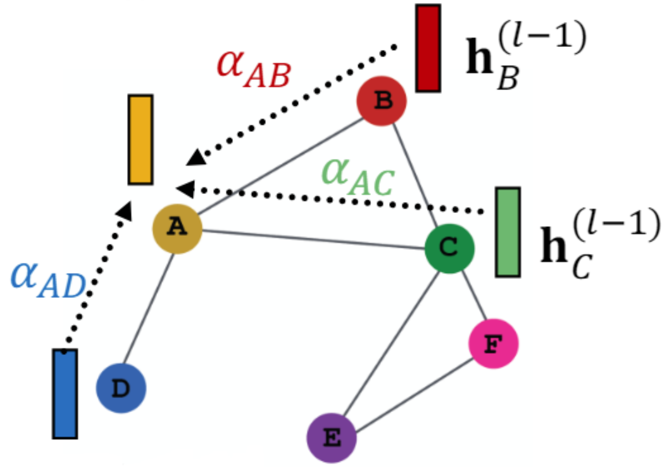
- Normalize $\boldsymbol{e_{vu}}$ into the final attention weight $\boldsymbol{\alpha_{vu}}$

$$\alpha_{vu} = \frac{\exp(e_{vu})}{\sum_{k \in N(v)} \exp(e_{vk})}$$

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

Weighted sum using α_{AB}, α_{AC}, α_{AD}:

$$\mathbf{h}_A^{(l)} = \sigma(\alpha_{AB}\mathbf{W}^{(l)}\mathbf{h}_B^{(l-1)} + \alpha_{AC}\mathbf{W}^{(l)}\mathbf{h}_C^{(l-1)} + \alpha_{AD}\mathbf{W}^{(l)}\mathbf{h}_D^{(l-1)})$$



P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph Attention ´ Networks. ICLR, 2018.

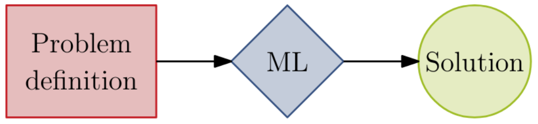
LEARNING AND ALGORITHMS

Integrating ML with combinatorial optimization:

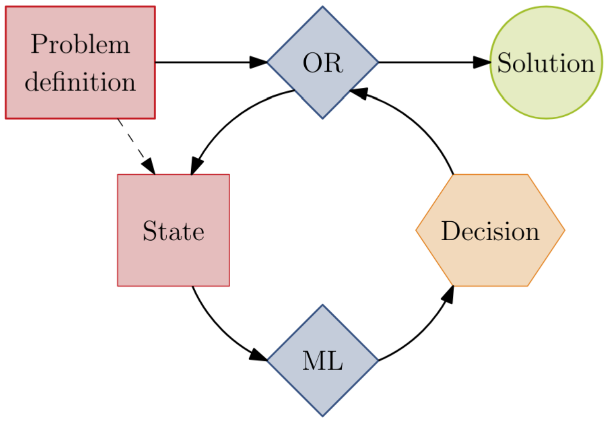
- **Algo-level** approaches focus on learning entire algorithms, end-to-end, from inputs to outputs
- **Step-level** approaches focus on learning atomic steps of algorithms (e.g. current branch selection in BnB algorithms), through strong intermediate supervision
- **Unit-level** approaches focus on strongly learning primitive units of computation, then specifying hard-coded or nonparametric means of combining such units

LEARNING AND ALGORITHMS

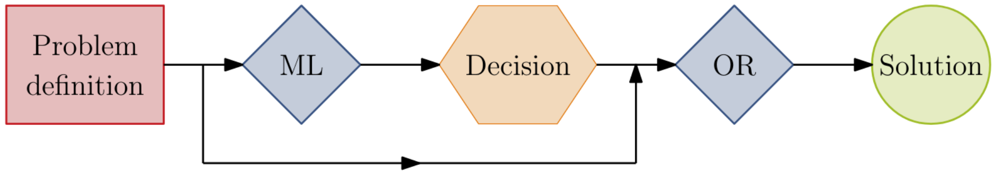
1.



2.



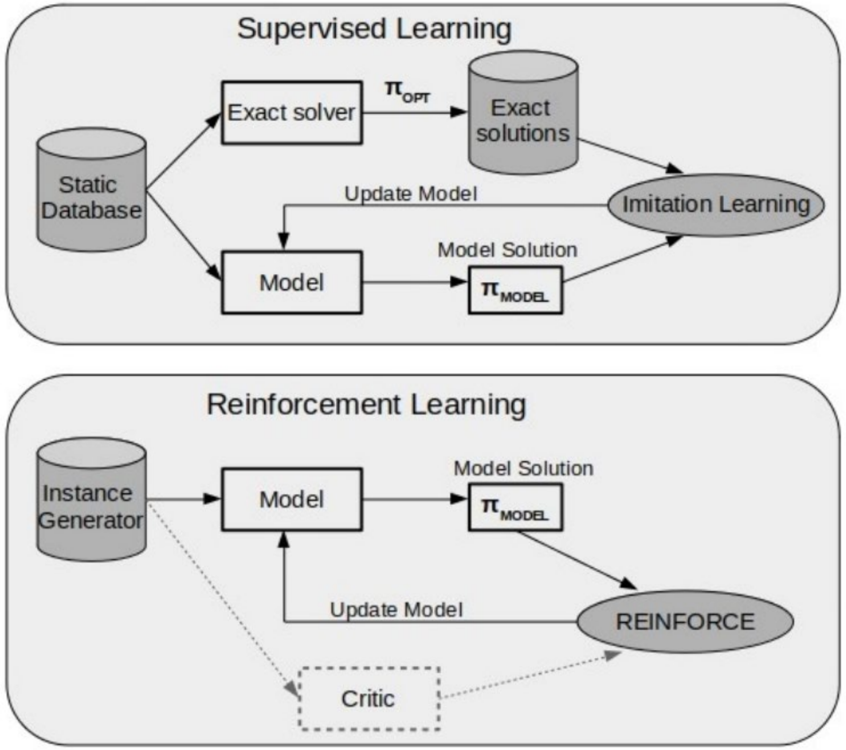
3.



WHY LEARN AN ALGORITHM?

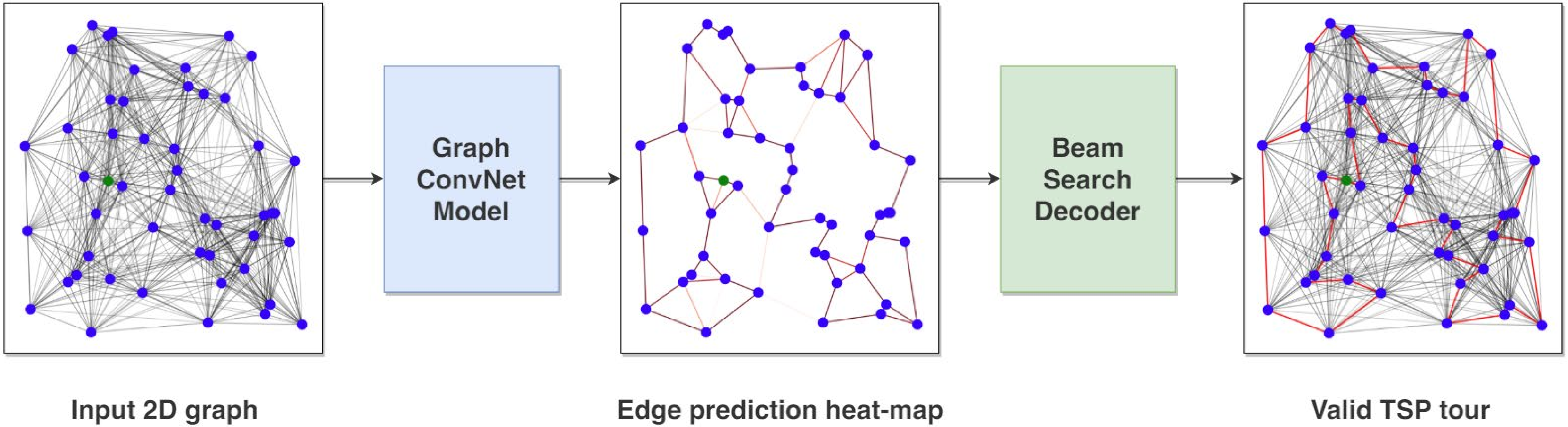
- Adapt to specific input distributions (without analytic form)
- Bypass expensive exact algorithm
- Discover new heuristics/algorithms, augment expert knowledge

EXISTING ALGO APPROACHES



Garmendia, A.I., Ceberio, J., & Mendiburu, A. (2022). Neural Combinatorial Optimization: a New Player in the Field

AN EFFICIENT GRAPH CONVNET FOR TSP



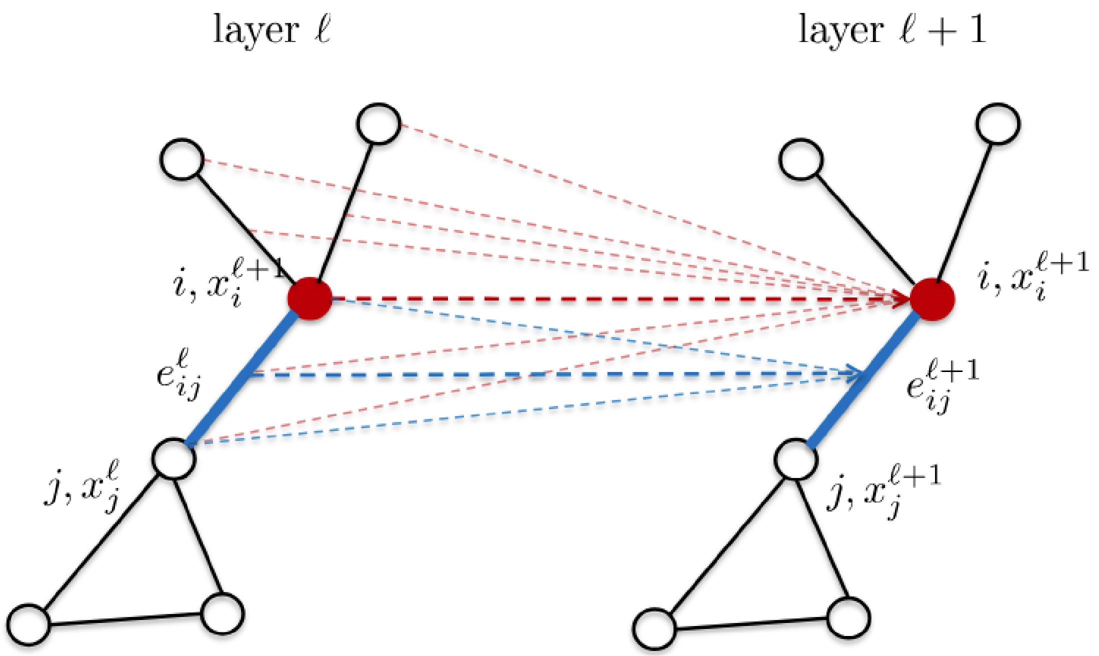
Chaitanya K. Joshi, Thomas Laurent, Xavier Bresson (2019). An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem

INPUT LAYER

As input node feature, we are given the two-dimensional coordinates $x_i \in [0,1]^2$ which are embedded as *h*-dimensional features $\alpha_i = A_1 x_i + b_1$

The edge input feature is: $\beta_{ij} = A_2 d_{ij} + b_2 \parallel A_3 \delta_{ij}^{\text{k-NN}}$ $\quad A_2 \in \mathbb{R}^{\frac{h}{2} \times 1}, A_3 \in \mathbb{R}^{\frac{h}{2} \times 3}$

NODE/EDGE EMBEDDING



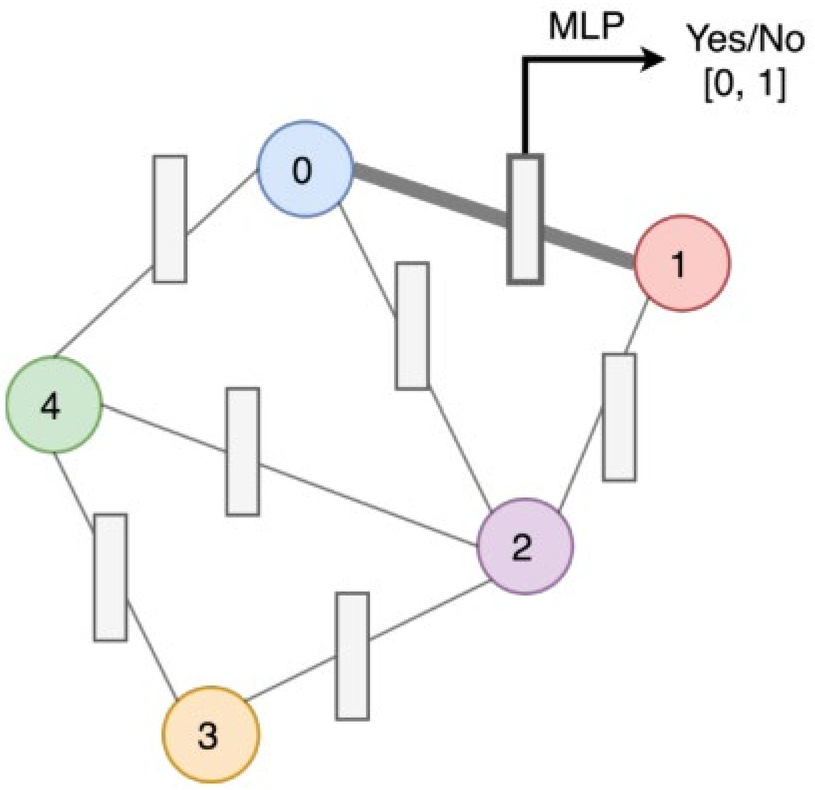
$$x_i^{\ell+1} = x_i^\ell + \mathrm{ReLU}\Big(\mathrm{BN}\Big(W_1^\ell x_i^\ell + \sum_{j\sim i} \eta_{ij}^\ell \odot W_2^\ell x_j^\ell\Big)\Big)$$

$$\eta_{ij}^\ell = \frac{\sigma(e_{ij}^\ell)}{\sum_{j'\sim i} \sigma(e_{ij'}^\ell) + \varepsilon}$$

$$e_{ij}^{\ell+1} = e_{ij}^\ell + \mathrm{ReLU}\Big(\mathrm{BN}\Big(V_1^\ell e_{ij}^\ell + V_2^\ell x_i^\ell + V_3^\ell x_j^\ell\Big)\Big)$$

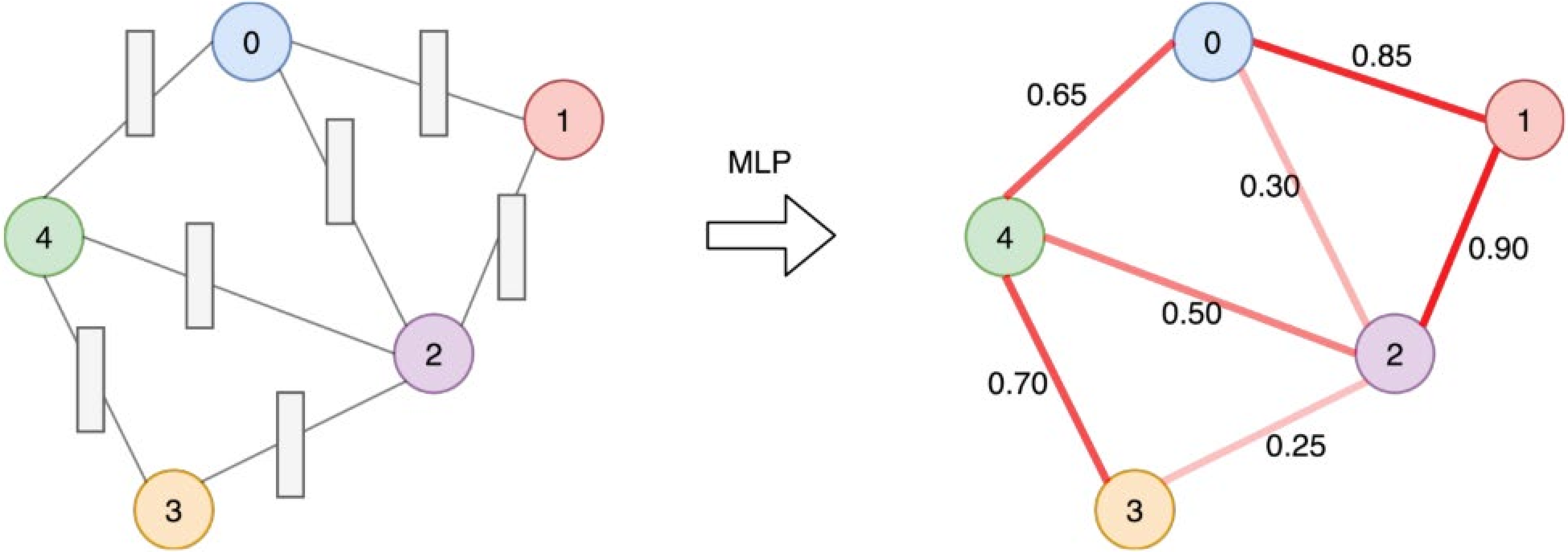
Residual Gated Graph ConvNets

PREDICTION



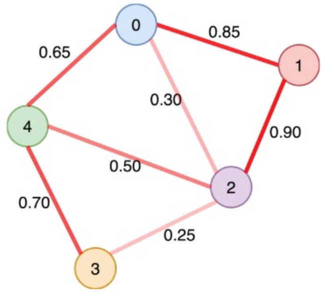
does an edge belong to the optimal tour?

PREDICTION



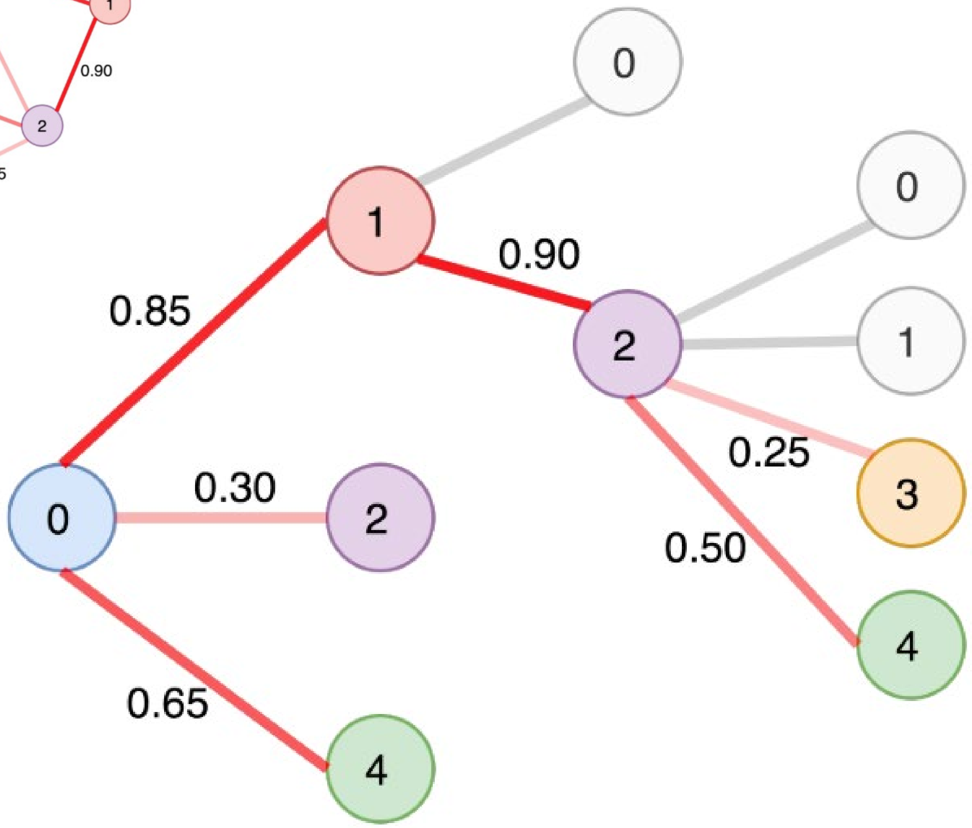
probability distribution over edges

SEARCH FOR FEASIBLE SOLUTIONS

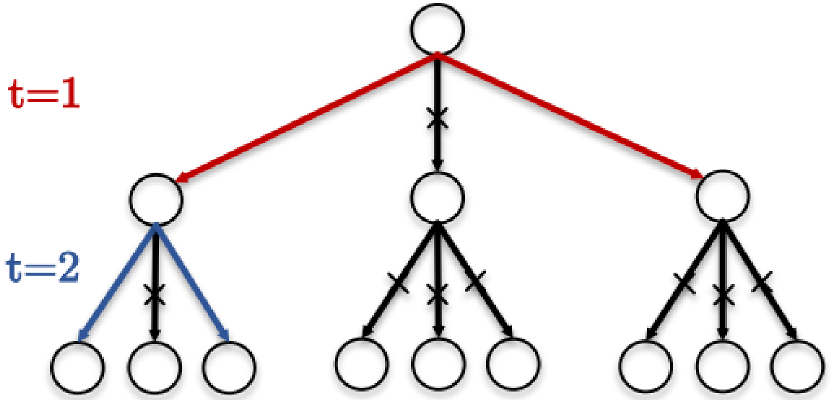


We can use any search algorithm for graphs + enforce problem constraints:

- Greedy search
- Beam search
- Monte Carlo tree search



BEAM SEARCH



Beam Search explores the search space by expanding to a limited set of children nodes selected by a criteria. Beam size is B=2

TRAINING THE POLICY

Learning by **Imitation** (Supervised)

- Minimize the loss between optimal solutions (Concorde) and model's prediction
- Binary classification problem on edges

RESULTS

Method	Type	TSP20 Tour Len.	TSP20 Opt. Gap.	TSP20 Time	TSP50 Tour Len.	TSP50 Opt. Gap.	TSP50 Time	TSP100 Tour Len.	TSP100 Opt. Gap.	TSP100 Time
Concorde	Solver	3.84	0.00%	(1m)	5.70	0.00%	(2m)	7.76	0.00%	(3m)
LKH3	Solver	3.84	0.00%	(18s)	5.70	0.00%	(5m)	7.76	0.00%	(21m)
Gurobi	Solver	3.84	0.00%	(7s)	5.70	0.00%	(2m)	7.76	0.00%	(17m)
Nearest Insertion	H, G	4.33	12.91%	(1s)	6.78	19.03%	(2s)	9.46	21.82%	(6s)
Random Insertion	H, G	4.00	4.36%	(0s)	6.13	7.65%	(1s)	8.52	9.69%	(3s)
Farthest Insertion	H, G	3.93	2.36%	(1s)	6.01	5.53%	(2s)	8.35	7.59%	(7s)
Nearest Neighbor	H, G	4.50	17.23%	(0s)	7.00	22.94%	(0s)	9.68	24.73%	(0s)
PtrNet [Vinyals et al., 2015]	SL, G	3.88	1.15%		7.66	34.48%			-	
PtrNet [Bello et al., 2016]	RL, G	3.89	1.42%		5.95	4.46%		8.30	6.90%	
S2V [Dai et al., 2017]	RL, G	3.89	1.42%		5.99	5.16%		8.31	7.03%	
GAT [Deudon et al., 2018]	RL, G	3.86	0.66%	(2m)	5.92	3.98%	(5m)	8.42	8.41%	(8m)
GAT [Deudon et al., 2018]	RL, G, 2OPT	3.85	0.42%	(4m)	5.85	2.77%	(26m)	8.17	5.21%	(3h)
GAT [Kool et al., 2019]	RL, G	3.85	0.34%	(0s)	5.80	1.76%	(2s)	8.12	4.53%	(6s)
GCN (Ours)	**SL, G**	**3.86**	**0.60%**	**(6s)**	**5.87**	**3.10%**	**(55s)**	**8.41**	**8.38%**	**(6m)**
OR Tools	H, S	3.85	0.37%		5.80	1.83%		7.99	2.90%	
Chr.f. + 2OPT	H, 2OPT	3.85	0.37%		5.79	1.65%			-	
GNN [Nowak et al., 2017]	SL, BS	3.93	2.46%			-			-	
PtrNet [Bello et al., 2016]	RL, S		-		5.75	0.95%		8.00	3.03%	
GAT [Deudon et al., 2018]	RL, S	3.84	0.11%	(5m)	5.77	1.28%	(17m)	8.75	12.70%	(56m)
GAT [Deudon et al., 2018]	RL, S, 2OPT	3.84	0.09%	(6m)	5.75	1.00%	(32m)	8.12	4.64%	(5h)
GAT [Kool et al., 2019]	RL, S	3.84	0.08%	(5m)	5.73	0.52%	(24m)	7.94	2.26%	(1h)
GCN (Ours)	**SL, BS**	**3.84**	**0.10%**	**(20s)**	**5.71**	**0.26%**	**(2m)**	**7.92**	**2.11%**	**(10m)**
GCN (Ours)	**SL, BS***	**3.84**	**0.01%**	**(12m)**	**5.70**	**0.01%**	**(18m)**	**7.87**	**1.39%**	**(40m)**

GENERALIZATION

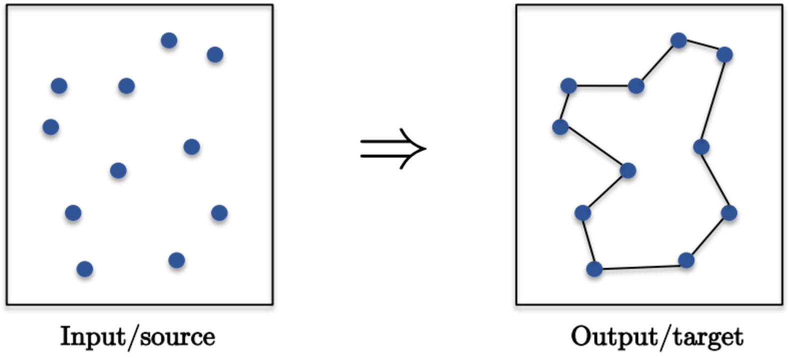
Method/Model	TSP20			TSP50			TSP100		
	Tour Len.	Opt. Gap.	Time	Tour Len.	Opt. Gap.	Time	Tour Len.	Opt. Gap.	Time
Concorde	3.84	0.00%	(1m)	5.70	0.00%	(2m)	7.76	0.00%	(3m)
TSP20 Model [Kool et al., 2019]	3.84	0.08%	(5m)	5.79	1.78%	(24m)	9.50	22.61%	(1h)
TSP50 Model [Kool et al., 2019]	3.84	0.35%	(5m)	5.73	0.52%	(24m)	7.98	2.95%	(1h)
TSP100 Model [Kool et al., 2019]	3.97	3.78%	(5m)	5.82	2.33%	(24m)	7.94	2.26%	(1h)
TSP20 Model (Ours)	3.84	0.10%	(20s)	7.66	34.46%	(2m)	13.18	69.95%	(10m)
TSP50 Model (Ours)	5.31	38.46%	(20s)	5.71	0.26%	(2m)	12.83	65.39%	(10m)
TSP100 Model (Ours)	4.94	28.68%	(20s)	7.43	30.49%	(2m)	7.92	2.11%	(10m)

SUPERVISED LEARNING

- + More stable training
- + Can perform quite well
- – High computational cost of acquiring labels for hard instances
- – Limited generalization as NN learns to imitate solver solutions

TRANSFORMER FOR TRAVELING SALESMAN PROBLEM (TSP)

- **TSP** as “translation” problem (similar to NLP)
  Source is a set of 2D points
  Target is a tour (sequence of indices) with minimal length
- Trained with reinforcement learning, i.e. without labeled data



Bresson, X., & Laurent, T. (2021). The Transformer Network for the Traveling Salesman Problem. *https://arxiv.org/pdf/2103.03012.pdf*
Kool, Wouter et al. *(2018).* “Attention, Learn to Solve Routing Problems!” *International Conference on Learning Representations*

TRAINING

- Given instance s the model defines probability distribution $p_\theta(\pi|s)$, from which we can sample to obtain a solution (tour) $\pi|s$

- Loss function – expectation of the cost $L(\pi)$ (tour length for TSP):

$$\mathcal{L}(\theta|s) = \mathbb{E}_{p_\theta(\pi|s)}[L(\pi)]$$

- $\mathcal{L}$ is optimized with gradient descent extension (Adam), using REINFORCE (Williams, 1992) gradient estimator with baseline $b(s)$:

$$\nabla\mathcal{L}(\theta|s) = \mathbb{E}_{p_\theta(\pi|s)}[\big(L(\pi) - b(s)\big)\nabla \log p_\theta(\pi|s)]$$

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning, 8(3-4):229–256, 1992

TRAINING

Algorithm 1 REINFORCE with Rollout Baseline

Input: number of epochs E, steps per epoch T, batch size B, significance α

Init $\boldsymbol{\theta}$, $\boldsymbol{\theta}^{\text{BL}} \leftarrow \boldsymbol{\theta}$

for epoch $= 1, \ldots, E$ **do**

  for step $= 1, \ldots, T$ **do**

    $s_i \leftarrow \text{RandomInstance}()\ \forall i \in \{1, \ldots, B\}$

    $\boldsymbol{\pi}_i \leftarrow \text{SampleRollout}(s_i, p_{\boldsymbol{\theta}})\ \forall i \in \{1, \ldots, B\}$

    $\boldsymbol{\pi}_i^{\text{BL}} \leftarrow \text{GreedyRollout}(s_i, p_{\boldsymbol{\theta}^{\text{BL}}})\ \forall i \in \{1, \ldots, B\}$

    $\nabla \mathcal{L} \leftarrow \sum_{i=1}^{B} \left(L(\boldsymbol{\pi}_i) - L(\boldsymbol{\pi}_i^{\text{BL}})\right) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{\pi}_i)$

    $\boldsymbol{\theta} \leftarrow \text{Adam}(\boldsymbol{\theta}, \nabla \mathcal{L})$

  end for

  if $\text{OneSidedPairedTTest}(p_{\boldsymbol{\theta}}, p_{\boldsymbol{\theta}^{\text{BL}}}) < \alpha$ **then**

    $\boldsymbol{\theta}^{\text{BL}} \leftarrow \boldsymbol{\theta}$

  end if

end for

NUMERICAL RESULTS

	Method	TSP50 Obj	TSP50 Gap	TSP50 T Time	TSP50 I Time	TSP100 Obj	TSP100 Gap	TSP100 T Time	TSP100 I Time
MIP	Concorde'06	**5.689**	0.00%	2m*	0.05s	**7.765**	0.00%	3m*	0.22s
MIP	Gurobi'08	-	0.00%*	2m*	-	7.765*	0.00%*	17m*	-
Heuristic	Nearest insertion	7.00*	22.94%*	0s*	-	9.68*	24.73%*	0s*	-
Heuristic	Farthest insertion	6.01*	5.53%*	2s*	-	8.35*	7.59%*	7s*	-
Heuristic	OR tools'15	5.80*	1.83%*	-	-	7.99*	2.90%*	-	-
Heuristic	LKH-3'17	-	0.00%*	5m*	-	**7.765***	0.00%*	21m*	-
Neural Network Greedy Sampling	Vinyals et-al'15	7.66*	34.48%*	-	-	-	-	-	-
Neural Network Greedy Sampling	Bello et-al'16	5.95*	4.46%*	-	-	8.30*	6.90%*	-	-
Neural Network Greedy Sampling	Dai et-al'17	5.99*	5.16%*	-	-	8.31*	7.03%*	-	-
Neural Network Greedy Sampling	Deudon et-al'18	5.81*	2.07%*	-	-	8.85*	13.97%*	-	-
Neural Network Greedy Sampling	Kool et-al'18	5.80*	1.76%*	2s*	-	8.12*	4.53%*	6s*	-
Neural Network Greedy Sampling	Kool et-al'18 (our version)	-	-	-	-	8.092	4.21%	-	-
Neural Network Greedy Sampling	Joshi et-al'19	5.87	3.10%	55s	-	8.41	8.38%	6m	-
Neural Network Greedy Sampling	Our model	**5.707**	0.31%	13.7s	0.07s	**7.875**	1.42%	4.6s	0.12s
Neural Network Advanced Sampling	Kool et-al'18 (B=1280)	5.73*	0.52%*	24m*	-	7.94*	2.26%*	1h*	-
Neural Network Advanced Sampling	Kool et-al'18 (B=5000)	5.72*	0.47%*	2h*	-	7.93*	2.18%*	5.5h*	-
Neural Network Advanced Sampling	Joshi et-al'19 (B=1280)	5.70	0.01%	18m	-	7.87	1.39%	40m	-
Neural Network Advanced Sampling	Xing et-al'20 (B=1200)	-	0.20%*	-	3.5s*	-	1.04%*	-	27.6s*
Neural Network Advanced Sampling	Wu et-al'20 (B=1000)	5.74*	0.83%*	16m*	-	8.01*	3.24%*	25m*	-
Neural Network Advanced Sampling	Wu et-al'20 (B=3000)	5.71*	0.34%*	45m*	-	7.91*	1.85%*	1.5h*	-
Neural Network Advanced Sampling	Wu et-al'20 (B=5000)	5.70*	0.20%*	1.5h*	-	7.87*	1.42%*	2h*	-
Neural Network Advanced Sampling	Our model (B=100)	5.692	0.04%	2.3m	**0.09s**	7.818	0.68%	4m	**0.16s**
Neural Network Advanced Sampling	Our model (B=1000)	5.690	0.01%	17.8m	0.15s	7.800	0.46%	35m	0.27s
Neural Network Advanced Sampling	Our model (B=2500)	**5.689**	4e-3%	44.8m	0.33s	**7.795**	0.39%	1.5h	0.62s

GENERALIZATION (TSPLIB)

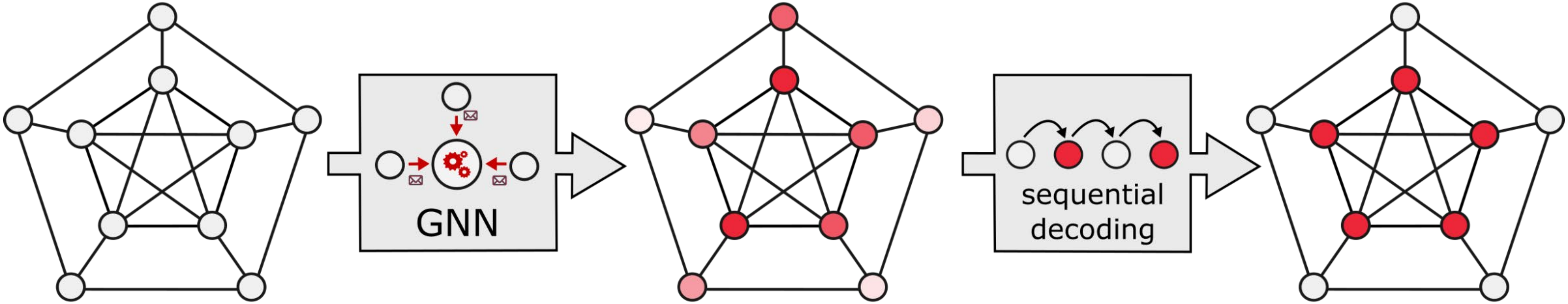
Problem	Critical parameter	Concorde	TSP Transformer		Ours	
			Tour length	Gap	Tour length	Gap
kroC100	0.75	20,749	21,788	5.01%	**21,523**	**3.73%**
berlin52	0.74	7,542	7,637	1.26%	**7,610**	**0.90%**
kroA100	0.77	21,282	21,747	2.18%	**21,620**	**1.59%**
ch150	0.78	6,528	7,390	13.20%	**7,050**	**8.00%**
ch130	0.78	6,110	6,569	7.51%	**6,552**	**7.23%**
rd100	0.81	7,910	**8,078**	**2.12%**	8,221	3.93%
st70	0.86	675	710	5.19%	**676**	**0.15%**
eil101	0.98	629	681	8.27%	**673**	**7.00%**
eil76	1.03	538	565	5.02%	**564**	**4.83%**
eil51	1.05	426	438	2.82%	**429**	**0.70%**

Jung, Minseop & Lee, Jaeseung & Kim, Jibum. (2023). A Lightweight CNN-Transformer Model for Learning Traveling Salesman Problems.

REINFORCEMENT LEARNING

+ Flexible and straightforward for simple CO problems
+ Can be used to model complicated combinatorial decision-making problems
– Often faces training stability/convergence issues
– Resource intensive

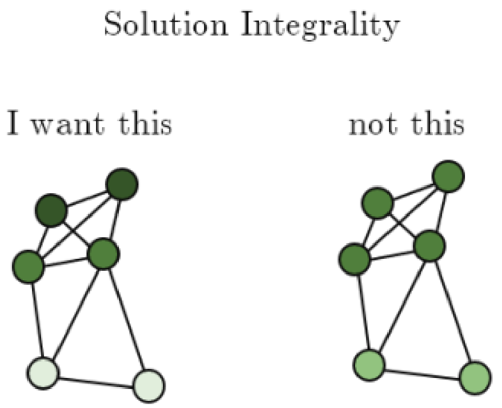
ERDŐS GOES NEURAL

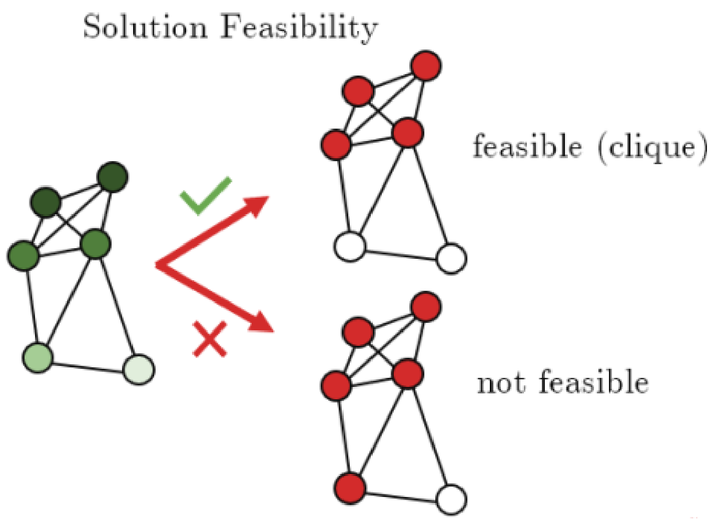


Karalias, Nikolaos & Loukas, Andreas. (2020). Erdos Goes Neural: an Unsupervised Learning Framework for Combinatorial Optimization on Graphs.

MAIN PROBLEM

Hard to obtain **integral** and **valid** solutions (w.r.t. constraints)

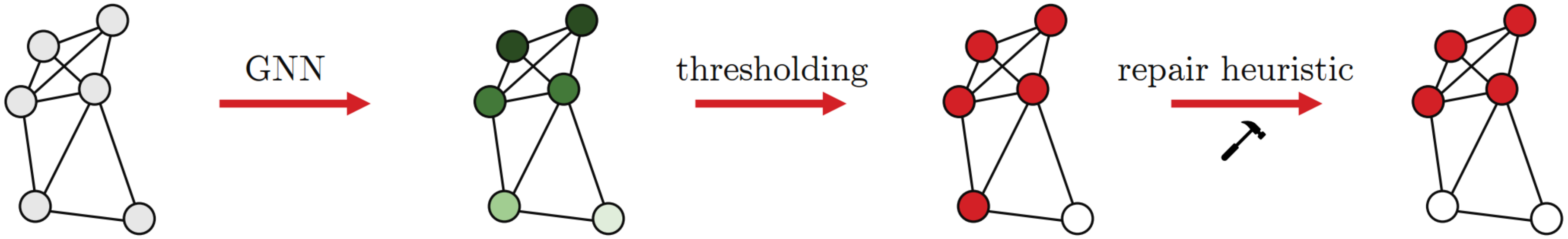




PREVIOUS SOLUTIONS

- **Regularization** $l = l_{objective} + \textcolor{green}{\text{Regularizer}}$
  configuring the regularizer can be an art.

- **Repair/improve solutions**



NN is trained independently of repair heuristic

ERDŐS GOES NEURAL APPROACH

1. Train a neural network to minimize a special **differentiable** loss
2. At test-time, use a sequential decoding scheme to obtain discrete solutions

ERDŐS GOES NEURAL

- Combinatorial optimization problems on graphs

$$\min_{S \subseteq V} f(S, G) \;\text{ subject to } S \in \Omega$$

- Trained with unsupervised learning, i.e. without labeled data
- Use probabilistic method pioneered by Paul Erdős

PROBABILISTIC METHOD

To prove the existence of an object with a desired property, proceed as follows:

1. Define a probability distribution over the space of all possible objects
2. Show that $P(object\ has\ property) > 0$

From this we can get simple **randomized algorithm**:

- sample *S* multiple times and select the one whose objective is the smallest

CONDITIONAL EXPECTATION METHOD

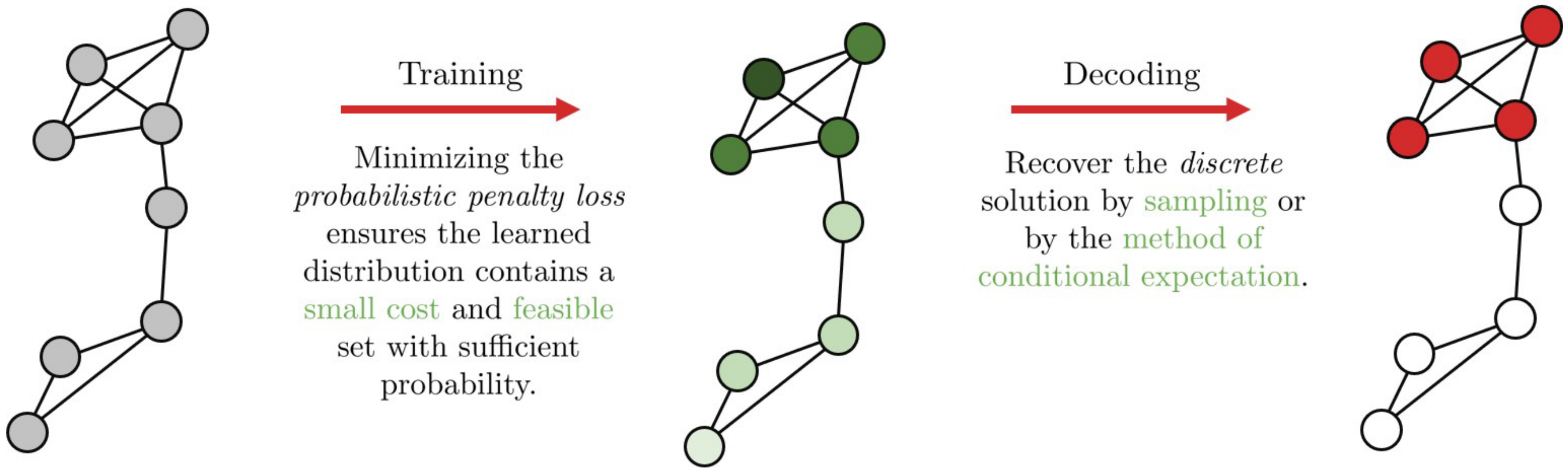
The **randomized** algorithm can be easily **derandomized**

- No need for sampling
- Find deterministic sequential algorithm that achieves same performance

Key idea: visit nodes sequentially and add node if the conditional expected objective value decreases

THE “ERDŐS GOES NEURAL” APPROACH

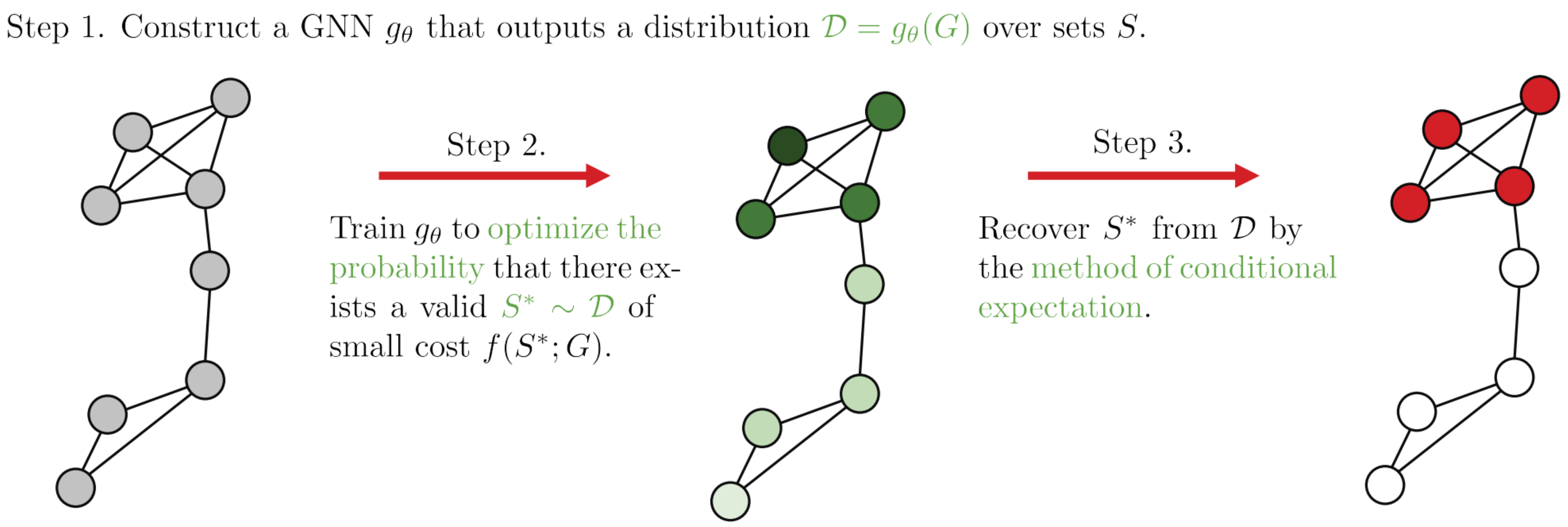
Construct a GNN that outputs a probability p_i on each node v_i.



ERDŐS GOES NEURAL APPROACH (DETAILS)

To solve the CO problem: $\min_{S \subseteq V} f(S, G)$ subject to $S \in \Omega$

- Apply probabilistic method
- But use a GNN to learn the probability distribution



THE PROBABILISTIC PENALTY LOSS

- Loss function is defined as

$$l(D, G) = \mathbb{E}_D[f(S, G)] + P_D(S \notin \Omega)\beta$$

where β is sufficiently large scalar and D is distribution over sets (solutions) in G

- The following theorem guarantees the existence of a solution:

Theorem 1. Fix any $\beta > \max_S f(S, G)$, let f be non-negative, and suppose $l(D, G) < (1 - t)\beta$. With probability at least t, set $S^* \sim D$ satisfies:

$$f(S^*, G) < \frac{l(D, G)}{(1 - t)} \text{and } S^* \in \Omega$$

THE PROBABILISTIC PENALTY LOSS (MAX CLIQUE)

Optimization problem formulation

Ω_{clique} : family of cliques

$w(S) = \sum_{v_i, v_j \in S} w_{ij}$

$$\min_{S \subseteq V} -w(S) \text{ subject to } S \in \Omega_{clique}$$

Translate the objective (for non-negativity): $f(S, G) = \gamma - w(S)$, where $w(S) < \gamma$ for $\forall S$

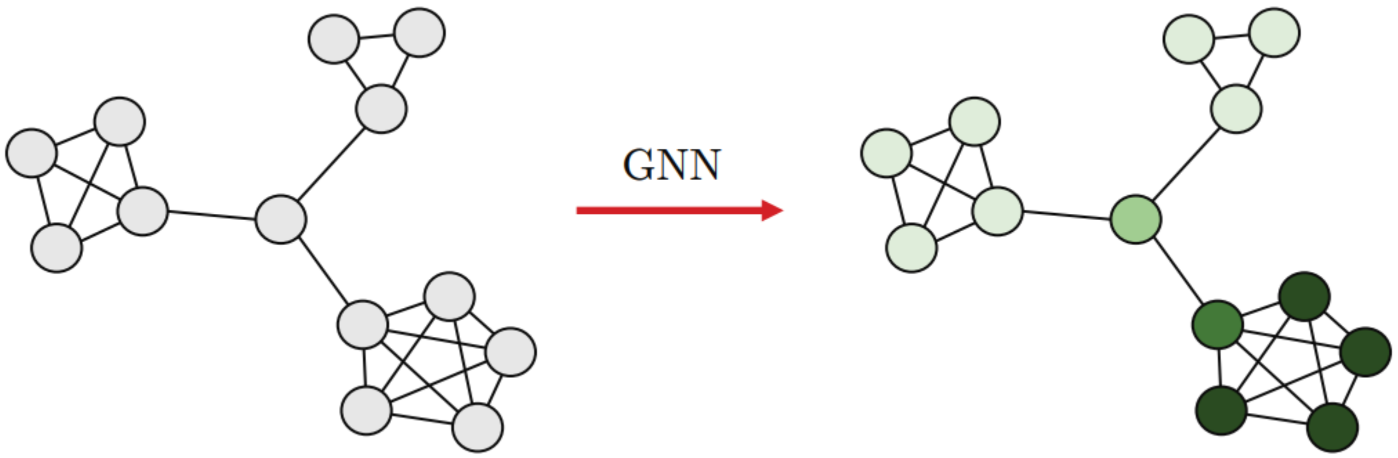
Probabilistic penalty loss function is defined as

$$l_{clique}(D, G) = \gamma - (\beta + 1) \sum_{(v_i, v_j) \in E} w_{ij} p_i p_j + \frac{\beta}{2} \sum_{v_i \neq v_j} p_i p_j$$

With probability at least t, set $S^* \sim D$ satisfies:

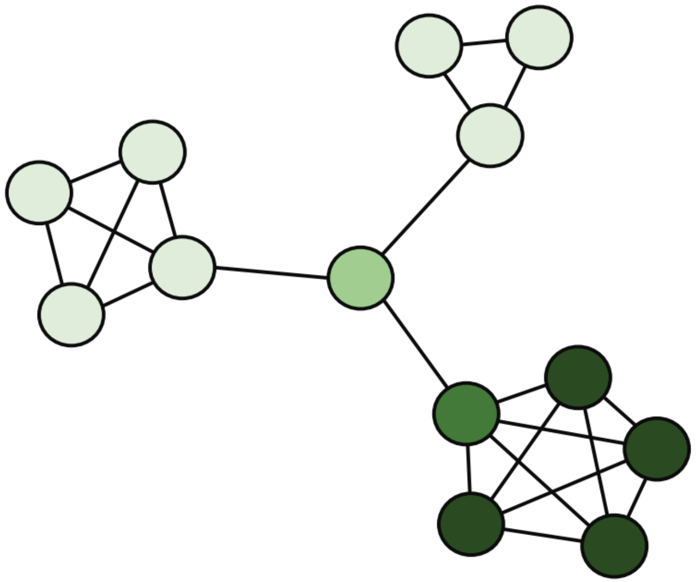
$$\gamma - \frac{l_{\text{clique}}(D, G)}{1 - t} < w(S^*) \text{ and } S^* \in \Omega$$

DECODING (CONDITIONAL EXPECTATION)



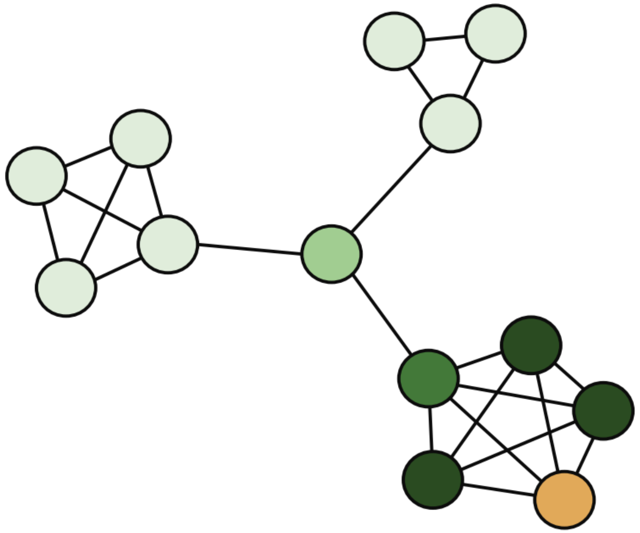
DECODING (CONDITIONAL EXPECTATION)

- At test time, we use the method of conditional expectation to sequentially decode a solution:



DECODING (CONDITIONAL EXPECTATION)

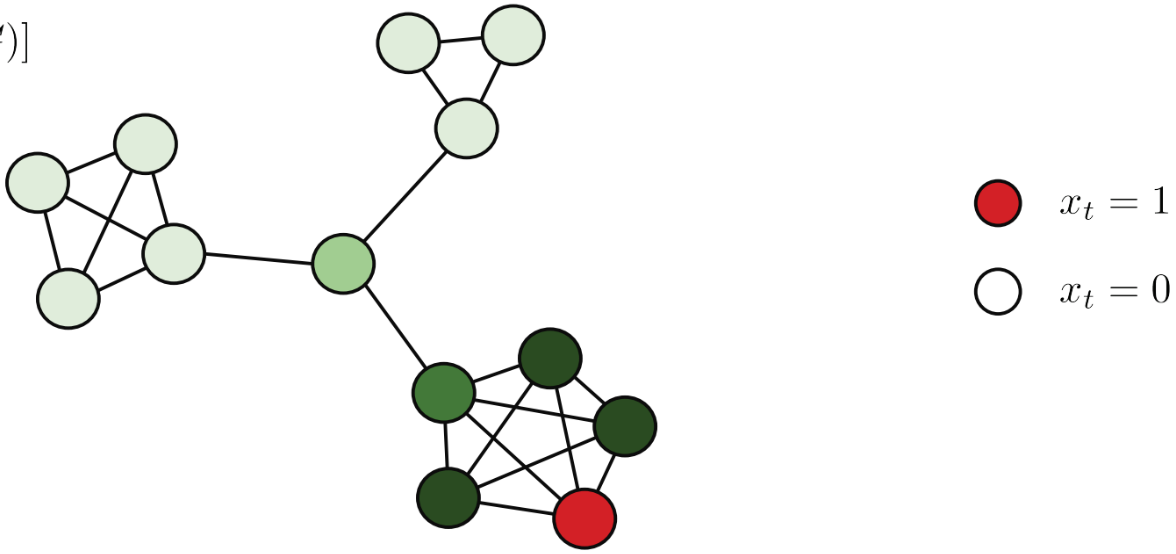
- At test time, we use the method of conditional expectation to sequentially decode a solution:



DECODING (CONDITIONAL EXPECTATION)

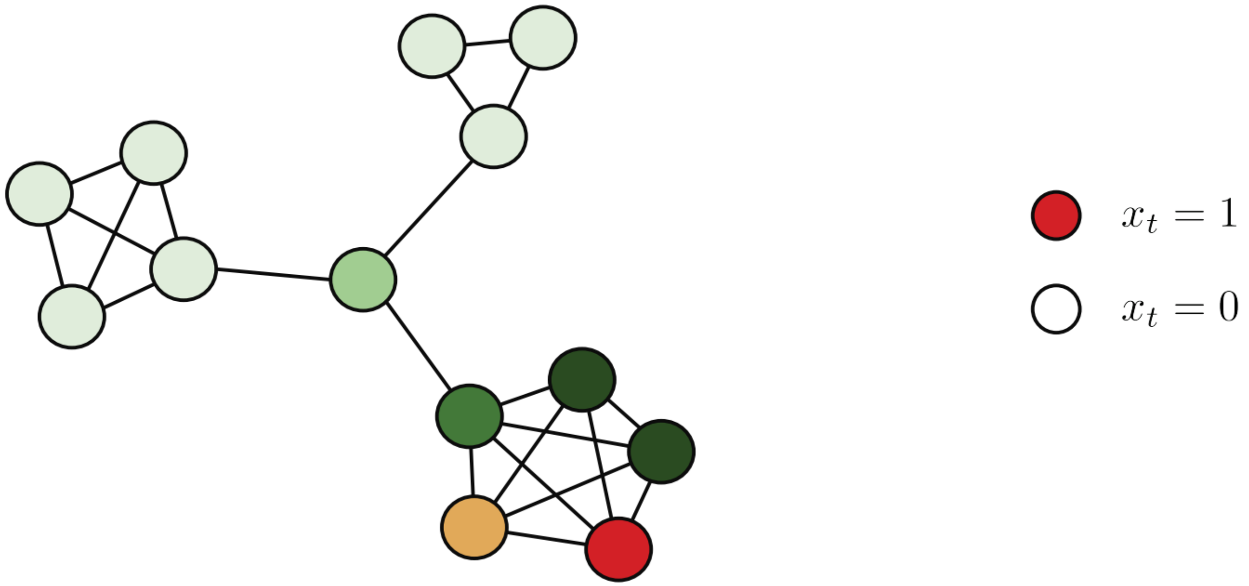
- At test time, we use the method of conditional expectation to sequentially decode a solution:

$$\mathbb{E}[f_p(S;G)\,|\,x_1 = 1] < \mathbb{E}[f_p(S;G)]$$



DECODING (CONDITIONAL EXPECTATION)

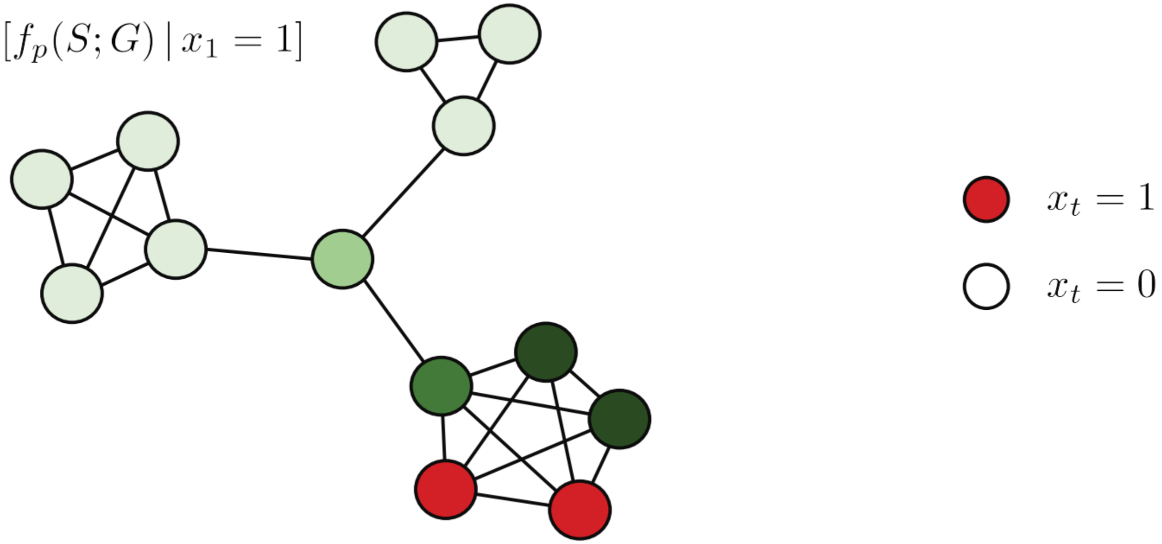
- At test time, we use the method of conditional expectation to sequentially decode a solution:



DECODING (CONDITIONAL EXPECTATION)

- At test time, we use the method of conditional expectation to sequentially decode a solution:

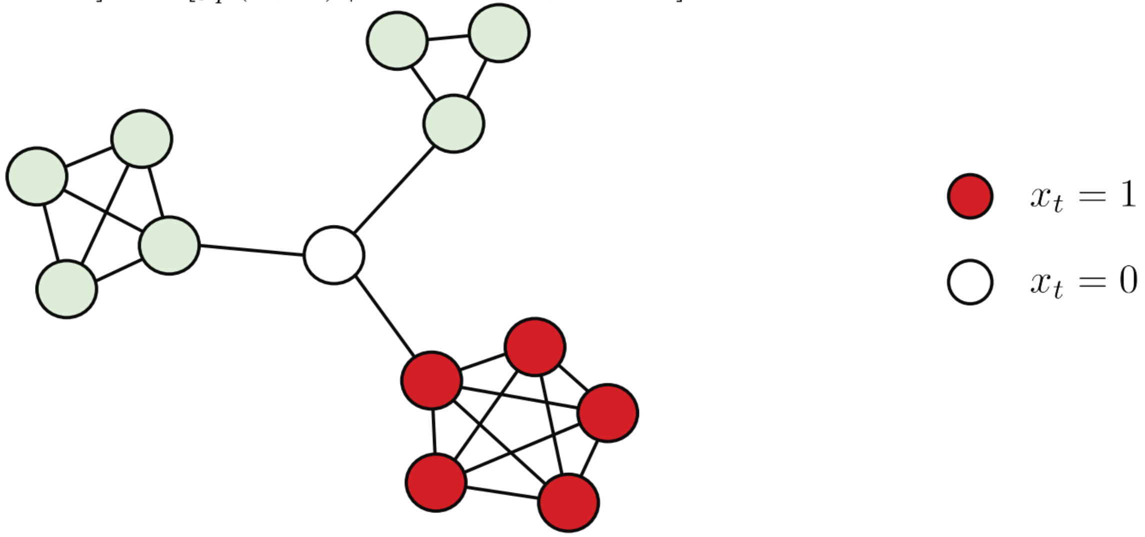
$$\mathbb{E}[f_p(S;G)\,|\,x_1 = 1, x_2 = 1] < \mathbb{E}[f_p(S;G)\,|\,x_1 = 1]$$



DECODING (CONDITIONAL EXPECTATION)

- At test time, we use the method of conditional expectation to sequentially decode a solution:

$$\mathbb{E}[f_p(S;G)\,|\,x_1=1,\ldots,x_5=1,x_6=0] \geq \mathbb{E}[f_p(S;G)\,|\,x_1=1,\ldots,x_5=1]$$

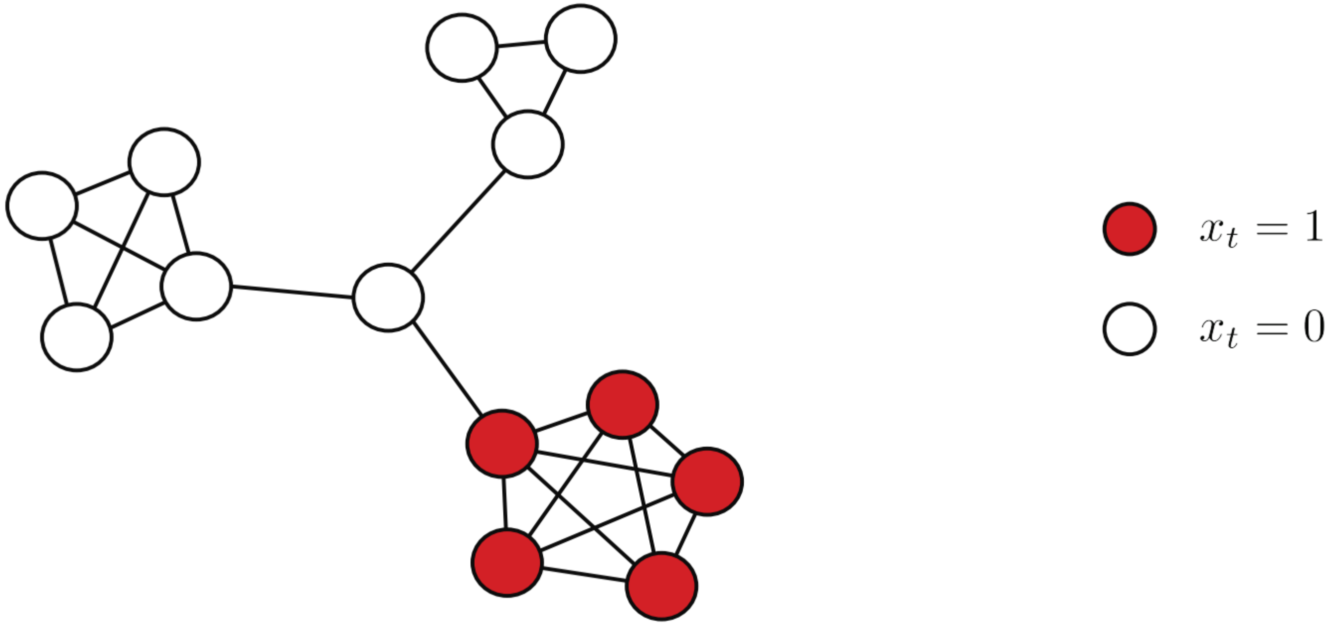


DECODING (CONDITIONAL EXPECTATION)

- At test time, we use the method of conditional expectation to sequentially decode a solution:

Avoid always computing the conditional expectation

- any subset of a clique is also a clique
- directly disregard invalid solutions



RESULTS

	IMDB	COLLAB	TWITTER
Erdős' GNN (fast)	1.000 (0.08 s/g)	0.982 ± 0.063 (0.10 s/g)	**0.924 ± 0.133 (0.17 s/g)**
Erdős' GNN (accurate)	1.000 (0.10 s/g)	0.990 ± 0.042 (0.15 s/g)	0.942 ± 0.111 (0.42 s/g)
RUN-CSP (fast)	0.823 ± 0.191 (0.11 s/g)	0.912 ± 0.188 (0.14 s/g)	0.909 ± 0.145 (0.21 s/g)
RUN-CSP (accurate)	0.957 ± 0.089 (0.12 s/g)	0.987 ± 0.074 (0.19 s/g)	0.987 ± 0.063 (0.39 s/g)
Bomze GNN	0.996 ± 0.016 (0.02 s/g)	~~0.984 ± 0.053 (0.03 s/g)~~	~~0.785 ± 0.163 (0.07 s/g)~~
MS GNN	0.995 ± 0.068 (0.03 s/g)	~~0.938 ± 0.171 (0.03 s/g)~~	~~0.805 ± 0.108 (0.07 s/g)~~
NX MIS approx.	0.950 ± 0.071 (0.01 s/g)	0.946 ± 0.078 (1.22 s/g)	0.849 ± 0.097 (0.44 s/g)
Greedy MIS Heur.	0.878 ± 0.174 (1e-3 s/g)	0.771 ± 0.291 (0.04 s/g)	0.500 ± 0.258 (0.05 s/g)
Toenshoff-Greedy	0.987 ± 0.050 (1e-3 s/g)	0.969 ± 0.087 (0.06 s/g)	**0.917 ± 0.126 (0.08 s/g)**
CBC (1s)	0.985 ± 0.121 (0.03 s/g)	0.658 ± 0.474 (0.49 s/g)	0.107 ± 0.309 (1.48 s/g)
CBC (5s)	1.000 (0.03 s/g)	0.841 ± 0.365 (1.11 s/g)	0.198 ± 0.399 (4.77 s/g)
Gurobi 9.0 (0.1s)	**1.000 (1e-3 s/g)**	0.982 ± 0.101 (0.05 s/g)	0.803 ± 0.258 (0.21 s/g)
Gurobi 9.0 (0.5s)	1.000 (1e-3 s/g)	0.997 ± 0.035 (0.06 s/g)	0.996 ± 0.019 (0.34 s/g)
Gurobi 9.0 (1s)	1.000 (1e-3 s/g)	0.999 ± 0.015 (0.06 s/g)	**1.000 (0.34 s/g)**
Gurobi 9.0 (5s)	1.000 (1e-3 s/g)	**1.000 (0.06 s/g)**	1.000 (0.35 s/g)

	IMDB	COLLAB	TWITTER	RB (all datasets)
Erdős' GNN (fast)	**0%**	**0%**	**0%**	**0%**
Erdős' GNN (accurate)	**0%**	**0%**	**0%**	**0%**
Bomze GNN	**0%**	11.8%	78.1%	–
MS GNN	1%	15.1%	84.7%	–

Constraint violation

Erdős' GNN	U ∼ [0,1]
0.905 ± 0.139	0.760 ± 0.172

Importance of learning

RESULTS

	Training set	Test set	Large Instances
Erdős' GNN (fast)	0.899 ± 0.064 (0.27 s/g)	0.788 ± 0.065 (0.23 s/g)	0.708 ± 0.027 (1.58 s/g)
Erdős' GNN (accurate)	0.915 ± 0.060 (0.53 s/g)	0.799 ± 0.067 (0.46 s/g)	0.735 ± 0.021 (6.68 s/g)
RUN-CSP (fast)	0.833 ± 0.079 (0.27 s/g)	0.738 ± 0.067 (0.23 s/g)	0.771 ± 0.032 (1.84 s/g)
RUN-CSP (accurate)	0.892 ± 0.064 (0.51 s/g)	0.789 ± 0.053 (0.47 s/g)	0.804 ± 0.024 (5.46 s/g)
Toenshoff-Greedy	**0.924 ± 0.060 (0.02 s/g)**	0.816 ± 0.064 (0.02 s/g)	**0.829 ± 0.027 (0.35 s/g)**
Gurobi 9.0 (0.1s)	0.889 ± 0.121 (0.18 s/g)	**0.795 ± 0.118 (0.16 s/g)**	0.697 ± 0.033 (1.17 s/g)
Gurobi 9.0 (0.5s)	**0.962 ± 0.076 (0.34 s/g)**	**0.855 ± 0.083 (0.31 s/g)**	0.697 ± 0.033 (1.54 s/g)
Gurobi 9.0 (1.0s)	**0.980 ± 0.054 (0.45 s/g)**	**0.872 ± 0.070 (0.40 s/g)**	0.705 ± 0.039 (2.05 s/g)
Gurobi 9.0 (5.0s)	**0.998 ± 0.010 (0.76 s/g)**	**0.884 ± 0.062 (0.68 s/g)**	0.790 ± 0.285 (6.01 s/g)
Gurobi 9.0 (20.0s)	**0.999 ± 0.003 (1.04 s/g)**	**0.885 ± 0.063 (0.96 s/g)**	0.807 ± 0.134 (21.24 s/g)

CONCLUSIONS

- Promising Results of GNNs:
  - **Innovative Approaches**: GNNs introduce innovative ways to leverage graph structures, providing new perspectives on problem-solving in various domains.
  - **Research Momentum**: Rapid advancements in GNN research indicate a strong potential for significant improvements and breakthroughs.
  - **Scalability**: GNNs can handle large-scale graph data, making them suitable for real-world applications.

CONCLUSIONS

- Current Limitations:
  - **Performance Gap**: While GNNs show great promise, they currently lag behind specialized state-of-the-art solvers in terms of precision and efficiency for certain combinatorial optimization problems.
  - **Optimization Challenges**: Fine-tuning GNNs for specific tasks can be complex and resource-intensive.
  - **Generalization** : The ability of GNNs to generalize across different types and sizes of graphs is a critical challenge that impacts their applicability to a wide range of problems.

THANK YOU FOR YOUR ATTENTION!

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ