# Combinatorial Optimization with Graph Neural Network: Chaining to Learn the Graph Alignment Problem

Marc Lelarge
INRIA, DI/ENS, PSL Research University

Toulouse - November 2024

# Can GNN solve combinatorial optimization problems ?

## Combinatorial Optimization and Reasoning with Graph Neural Networks


**Quentin Cappart** QUENTIN.CAPPART@POLYMTL.CA
*Department of Computer Engineering and Software Engineering*
*Polytechnique Montréal*
*Montréal, Canada*

**Didier Chételat** DIDIER.CHETELAT@POLYMTL.CA
*CERC in Data Science for Real-Time Decision-Making*
*Polytechnique Montréal*
*Montréal, Canada*

**Elias B. Khalil** KHALIL@MIE.UTORONTO.CA
*Department of Mechanical & Industrial Engineering*
*University of Toronto*
*Toronto, Canada*

**Andrea Lodi** ANDREA.LODI@CORNELL.EDU
*Jacobs Technion-Cornell Institute*
*Cornell Tech and Technion - IIT*
*New York, USA*

**Christopher Morris** MORRIS@CS.RWTH-AACHEN.DE
*Department of Computer Science*
*RWTH Aachen University*
*Aachen, Germany*

**Petar Veličković** PETARV@DEEPMIND.COM
*DeepMind*
*London, UK*

# GNNs 'solving' the maximum independent set problem

Cracking nuts with a sledgehammer: when modern graph neural networks do worse than classical greedy algorithms

Maria Chiara Angelini[1,2] Federico Ricci-Tersenghi[1,2,3]

# GNNs 'solving' the maximum independent set problem

Cracking nuts with a sledgehammer: when modern graph neural networks do worse than classical greedy algorithms

Maria Chiara Angelini[1,2] Federico Ricci-Tersenghi[1,2,3]

Despite careful attempts, Böther et al. (2022) were incapable of reproducing the results, even reporting that using random weights in the GNN yields similar results as the trained weights. Thus, at this moment this approach should be considered at best inconclusive (...)

from Cappart et al. (2023)


GAP

# Graph Alignment Problem (GAP)

Given two $n \times n$ adjacency matrices $A$ and $B$, the graph alignment problem is to minimize $\|A - PBP^T\|_F$ over all permutation matrices $P$ and where $\|\cdot\|_F$ is the Frobenius norm :

$$\text{GAP} = \min_{\pi \in \mathcal{S}_n} \sum_{i,j} \left(A_{ij} - B_{\pi(i)\pi(j)}\right)^2,$$

where $\pi$ is the permutation associated to the permutation matrix $P$. We denote by $\pi^{A \to B}$ a solution to the graph alignment problem.

# Graph Alignment Problem (GAP)

Given two $n \times n$ adjacency matrices $A$ and $B$, the graph alignment problem is to minimize $\|A - PBP^T\|_F$ over all permutation matrices $P$ and where $\|\cdot\|_F$ is the Frobenius norm :

$$\text{GAP} = \min_{\pi \in \mathcal{S}_n} \sum_{i,j} \left(A_{ij} - B_{\pi(i)\pi(j)}\right)^2,$$

where $\pi$ is the permutation associated to the permutation matrix $P$. We denote by $\pi^{A \to B}$ a solution to the graph alignment problem.

For unweighted graphs, the coefficients of the matrices $A$ and $B$ are in $\{0, 1\}$, hence $\pi^{A \to B}$ also solves :

$$\max_{\pi \in \mathcal{S}_n} \sum_{i,j} A_{ij} B_{\pi(i)\pi(j)},$$

which is finding a maximum common subgraph in $G_A$ and $G_B$, known to be APX-hard.

# Measure of performances

For an algorithm producing a candidate permutation $\pi$, we measure its performance through two quantities :

- the accuracy defined by

$$\mathbf{acc}(\pi, \pi^{A \to B}) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(\pi(i) = \pi^{A \to B}(i)). \tag{1}$$

- the number of common edges defined by

$$\mathbf{nce}(\pi) = \frac{1}{2} \sum_{i,j} A_{ij} B_{\pi(i)\pi(j)} \leq \mathbf{nce}(\pi^{A \to B}). \tag{2}$$

# Graph alignment is a hard problem

$$\text{GAP} = \max_{\pi \in \mathcal{S}_n} \sum_{i,j} A_{\pi(i)\pi(j)} B_{ij}$$

- Take $G_A$ a graph on $n$ vertices and $G_B$ a path (or a cycle) of length $n$.

# Graph alignment is a hard problem

$$\text{GAP} = \max_{\pi \in \mathcal{S}_n} \sum_{i,j} A_{\pi(i)\pi(j)} B_{ij}$$

- Take $G_A$ a graph on $n$ vertices and $G_B$ a path (or a cycle) of length $n$. Then, GAP is the **Hamiltonian path/cycle problem** on $G_A$.
- Take $G_A$ a graph on $n$ vertices and $G_B$ a union of two clique of sizes $n/2$.

# Graph alignment is a hard problem

$$\text{GAP} = \max_{\pi \in \mathcal{S}_n} \sum_{i,j} A_{\pi(i)\pi(j)} B_{ij}$$

- Take $G_A$ a graph on $n$ vertices and $G_B$ a path (or a cycle) of length $n$. Then, GAP is the **Hamiltonian path/cycle problem** on $G_A$.
- Take $G_A$ a graph on $n$ vertices and $G_B$ a union of two clique of sizes $n/2$. Then, GAP is the **minimum bisection problem** on $G_A$.

# Graph alignment is a hard problem

$$\text{GAP} = \max_{\pi \in \mathcal{S}_n} \sum_{i,j} A_{\pi(i)\pi(j)} B_{ij}$$

- Take $G_A$ a graph on $n$ vertices and $G_B$ a path (or a cycle) of length $n$. Then, GAP is the **Hamiltonian path/cycle problem** on $G_A$.
- Take $G_A$ a graph on $n$ vertices and $G_B$ a union of two clique of sizes $n/2$. Then, GAP is the **minimum bisection problem** on $G_A$.
- Take $G_A = G_B$, then GAP is the **graph isomorphism problem** solvable in quasipolynomial time Babai (2016).

# Synthetic datasets

Random pairs of graphs $(G_A, G_B)$ such that the marginals are the same, i.e. the laws of $G_A$ and $G_B$ are identical but $G_A$ and $G_B$ are correlated. This correlation allows us to control the difficulty of the graph alignment problem. Then a random permutation $\pi^\star \in \mathcal{S}_n$ is applied on the nodes of $G_B$ to get $G'_B$ and the training is done on the generated triplets $(G_A, G'_B, \pi^\star)$.

# Synthetic datasets

Random pairs of graphs $(G_A, G_B)$ such that the marginals are the same, i.e. the laws of $G_A$ and $G_B$ are identical but $G_A$ and $G_B$ are correlated. This correlation allows us to control the difficulty of the graph alignment problem. Then a random permutation $\pi^\star \in \mathcal{S}_n$ is applied on the nodes of $G_B$ to get $G'_B$ and the training is done on the generated triplets $(G_A, G'_B, \pi^\star)$.

3 parameters :
the **number of nodes** $n$, the **average degree** $d$ and the **noise level** $p_{\text{noise}}$.

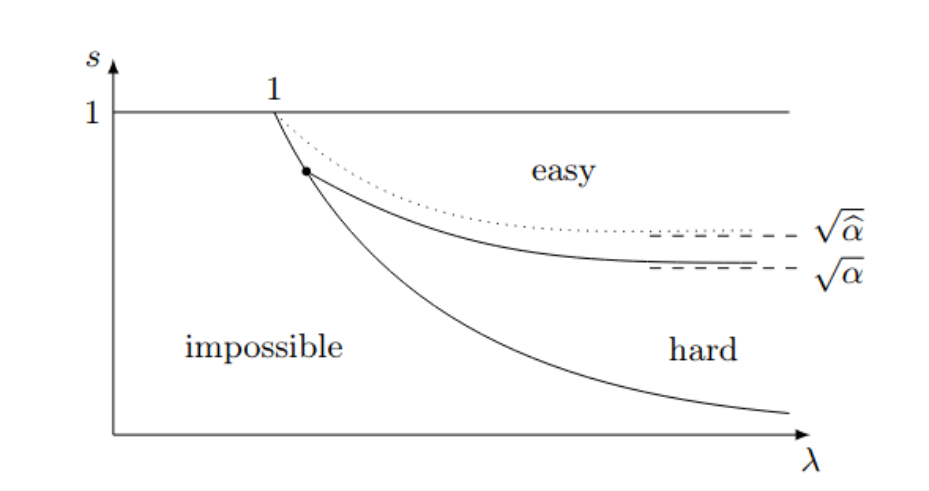
On average, $G_A$ and $G_B$ have $nd/2 = \mathbb{E}[\sum_{ij} A_{ij}/2]$ edges and the noise level $p_{\text{noise}}$ controls the number of edges that are different between $G_A$ and $G_B$ so that the average number of common edges is
$(1 - p_{\text{noise}})nd/2 = \mathbb{E}[\sum_{ij} A_{ij}B_{ij}/2]$.


Tell me the truth...I'm...I'm ready to hear it.
You don't need machine learning for that.

# Recovering the planted permutation (without learning)

**Faster algorithms for the alignment of sparse correlated Erdős-Rényi random graphs**

Andrea Muratori[1] and Guilhem Semerjian[2]




Otter's threshold : $\sqrt{\alpha} \approx 0.581$.

Ganassali et al. (2021b), Ganassali et al. (2021a), Piccioli et al. (2022), Ding et al. (2021), Mao et al. (2023), Muratori and Semerjian (2024)

# Continuous relaxations of GAP (1)

Using basic properties of permutation matrices, we get :

$$\begin{aligned} \|A - PBP^T\|_F^2 &= \|(AP - PB)P^T\|_F^2 \\ &= \|AP - PB\|_F^2 \\ &= \|A\|_F^2 + \|B\|_F^2 - 2\langle AP, PB\rangle. \end{aligned}$$

where $\langle C, D\rangle = \text{trace}(C^T D)$ is the Frobenius inner product.

# Continuous relaxations of GAP (1)

Using basic properties of permutation matrices, we get :

$$\begin{aligned}\|A - PBP^T\|_F^2 &= \|(AP - PB)P^T\|_F^2 \\ &= \|AP - PB\|_F^2 \\ &= \|A\|_F^2 + \|B\|_F^2 - 2\langle AP, PB\rangle.\end{aligned}$$

where $\langle C, D\rangle = \text{trace}(C^T D)$ is the Frobenius inner product.

Replacing the discrete set of permutations matrices $\mathcal{S}_n$ by the set of doubly stochastic matrices $\mathcal{D}_n$ :

- **convex relaxation** :

$$\arg\min_{D \in \mathcal{D}_n} \|AD - DB\|_F^2 = D_{\text{CX}}$$

- **indefinite relaxation** (still NP-hard) :

$$\max_{D \in \mathcal{D}_n} \langle AD, DB\rangle.$$

# Continuous relaxations of GAP (2)

- **convex relaxation** :

$$\arg \min_{D \in \mathcal{D}_n} \|AD - DB\|_F^2 = D_{\mathrm{CX}}$$

- **FAQ indefinite relaxation** :

$$\max_{D \in \mathcal{D}_n} \langle AD, DB \rangle .$$

# Continuous relaxations of GAP (2)

- **convex relaxation** :

$$\arg\min_{D\in\mathcal{D}_n} \|AD - DB\|_F^2 = D_{\mathrm{CX}}$$

- **FAQ indefinite relaxation** :

$$\max_{D\in\mathcal{D}_n} \langle AD, DB\rangle.$$

In both continuous relaxations, we use Frank-Wolfe algorithm and obtain a doubly stochastic matrix in $\mathcal{D}_n$ that needs to be projected to the nearest permutation matrix by solving a linear assignment problem (in $O(n^3)$ time) : for $D \in \mathcal{D}_n$, $\max_{P\in\mathcal{S}_n} \langle P, D\rangle$. We denote by $\mathbf{Proj}(D) \in \mathcal{S}_n$ the resulting projection of $D$ on $\mathcal{S}_n$.

# Continuous relaxations of GAP (2)

- **convex relaxation** :

$$\arg\min_{D\in\mathcal{D}_n} \|AD - DB\|_F^2 = D_{\text{CX}}$$

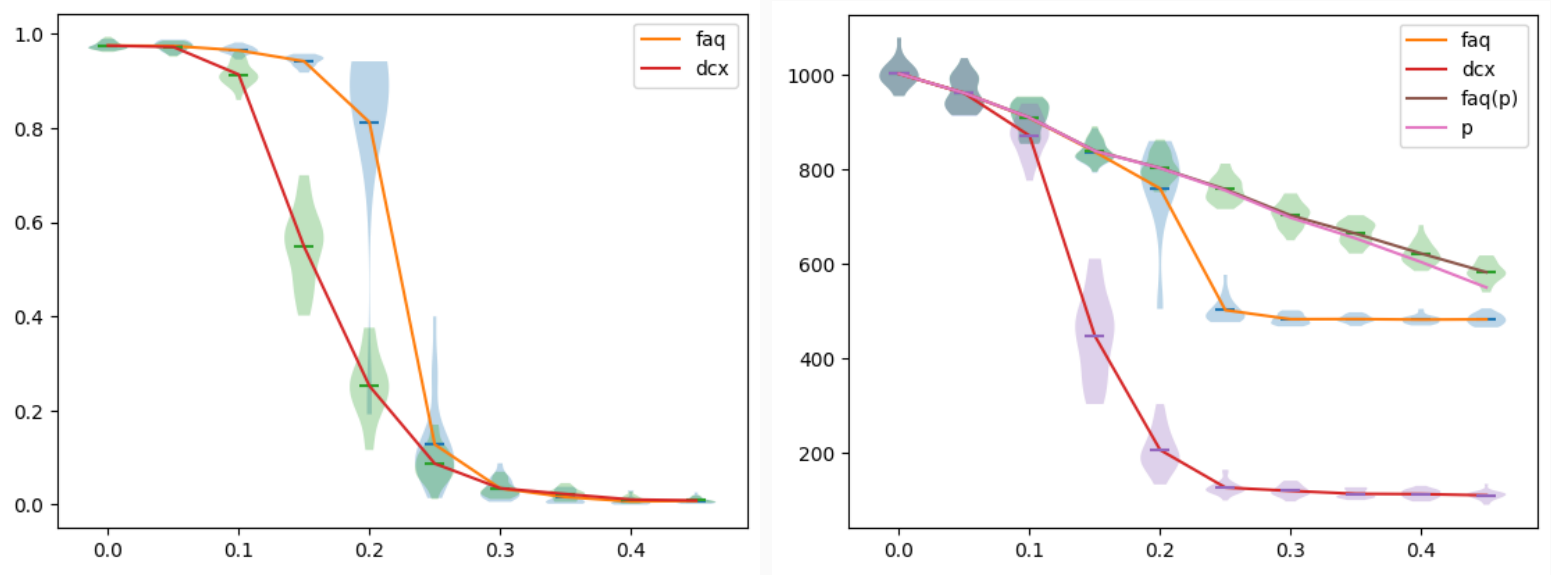
- **FAQ indefinite relaxation** :

$$\max_{D\in\mathcal{D}_n} \langle AD, DB\rangle.$$

In both continuous relaxations, we use Frank-Wolfe algorithm and obtain a doubly stochastic matrix in $\mathcal{D}_n$ that needs to be projected to the nearest permutation matrix by solving a linear assignment problem (in $O(n^3)$ time) : for $D \in \mathcal{D}_n$, $\max_{P\in\mathcal{S}_n} \langle P, D\rangle$. We denote by $\textbf{Proj}(D) \in \mathcal{S}_n$ the resulting projection of $D$ on $\mathcal{S}_n$.

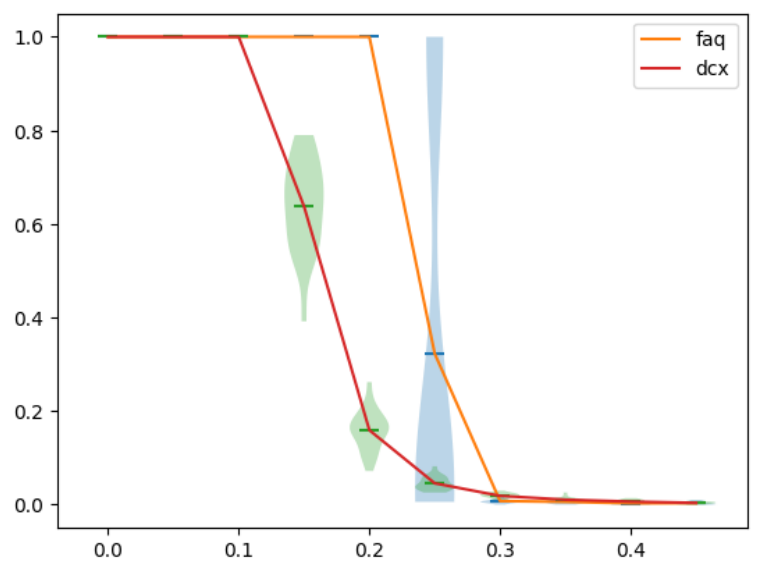
$\textbf{FAQ}(D) \in \mathcal{S}_n$ is the solution obtained with initial condition $D$ and after projection on $\mathcal{S}_n$. There are cases where $\textbf{Proj}(D_{\text{CX}})$ is indeed very far from an optimal solution and $\textbf{FAQ}(D_{\text{CX}})$ gives a better approximation.

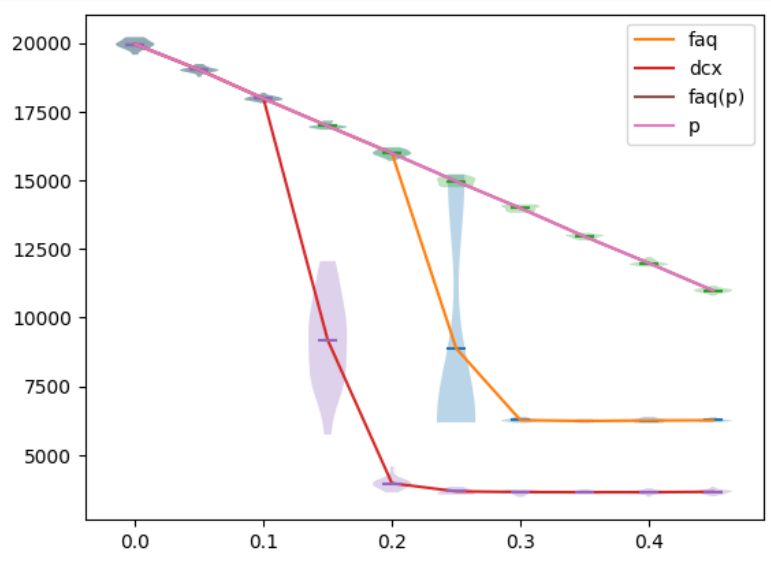
# Erdős-Rényi ($n = 500, d = 4$)




Accuracy (left) and number of common edges (right) as a function of noise.

# Erdős-Rényi ($n = 500, d = 80$)




Accuracy (left) and number of common edges (right) as a function of noise.


YOU KNOW, IF YOU GO IN THERE, THERE'S NO COMING BACK.
I... I HAVE TO DO IT. I HAVE TO KNOW WHAT HAPPENED TO MY FRIENDS.
Machine Learning
THE FOOL...
SHEN COMIX

# FAQ performs better than GNNs

Our SeedGNN achieves the following matching accuracy (%) on sparse ER graphs ($n = 500, p=0.01, s=0.8$):

| **Fraction of Seeds** | **0%** | **2%** | **4%** | **6%** | **8%** | **10%** | **12%** | **14%** | **16%** | **18%** | **20%** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| SeedGNN | 0.3 | 15.1 | 47.4 | 82.8 | 96.0 | 96.6 | 97.0 | 97.6 | 97.6 | 97.6 | 97.6 |
| 1-hop ($T=6$) | 0.2 | 1.5 | 2.89 | 4.91 | 6.0 | 9.2 | 12.5 | 15.3 | 19.7 | 23.6 | 32.3 |
| 2-hop ($T=3$) | 0.2 | 2.4 | 18.0 | 57.9 | 81.1 | 92.1 | 95.8 | 96.4 | 96.4 | 96.7 | 97.0 |
| 3-hop ($T=2$) | 0.3 | 2.4 | 7.1 | 29.7 | 64.9 | 90.8 | 96.0 | 97.1 | 97.2 | 97.4 | 97.5 |
| PGM | 0.2 | 2.3 | 6.1 | 16.3 | 31.6 | 54.5 | 73.3 | 79.2 | 86.3 | 88.9 | 92.7 |
| SGM | 0.3 | 3.6 | 8.9 | 13.8 | 22.3 | 36.3 | 54.5 | 67.3 | 84.4 | 89.6 | 91.6 |
| MGCN | 0.1 | 2.0 | 4.0 | 6.7 | 8.4 | 11.1 | 12.4 | 14.0 | 16.3 | 18.9 | 20.5 |

Our SeedGNN achieves the following matching accuracy (%) on dense ER graphs ($n = 500, p=0.2, s=0.8$):

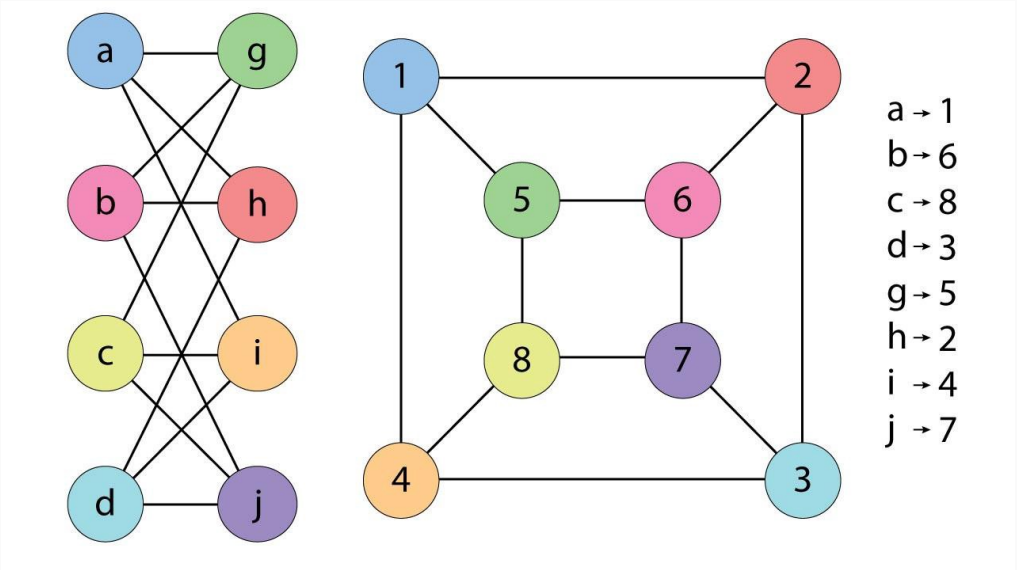
| **Fraction of Seeds** | **0.0** | **0.5%** | **1%** | **1.5%** | **2%** | **2.5%** | **3%** | **3.5%** | **4%** | **4.5%** | **5%** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| SeedGNN | 0.1 | 0.7 | 91.4 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |
| 1-hop ($T=6$) | 0.1 | 0.7 | 2.3 | 7.4 | 95.0 | 100 | 100 | 100 | 100 | 100 | 100 |
| 2-hop ($T=3$) | 0.0 | 0.7 | 2.2 | 5.6 | 46.6 | 100 | 100 | 100 | 100 | 100 | 100 |
| 3-hop ($T=2$) | 0.1 | 0.2 | 0.38 | 0.6 | 0.4 | 0.54 | 0.9 | 1.3 | 1.2 | 1.4 | 2.1 |
| PGM | 0.1 | 0.6 | 1.8 | 4.3 | 19.3 | 51.2 | 96.6 | 100 | 100 | 100 | 100 |
| SGM | 0.2 | 1.5 | 85.8 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |
| MGCN | 0.1 | 0.7 | 1.5 | 1.9 | 3.7 | 5.2 | 6.9 | 8.0 | 10.9 | 12.3 | 13.7 |

Yu et al. (2023)

# Learning with graph symmetries

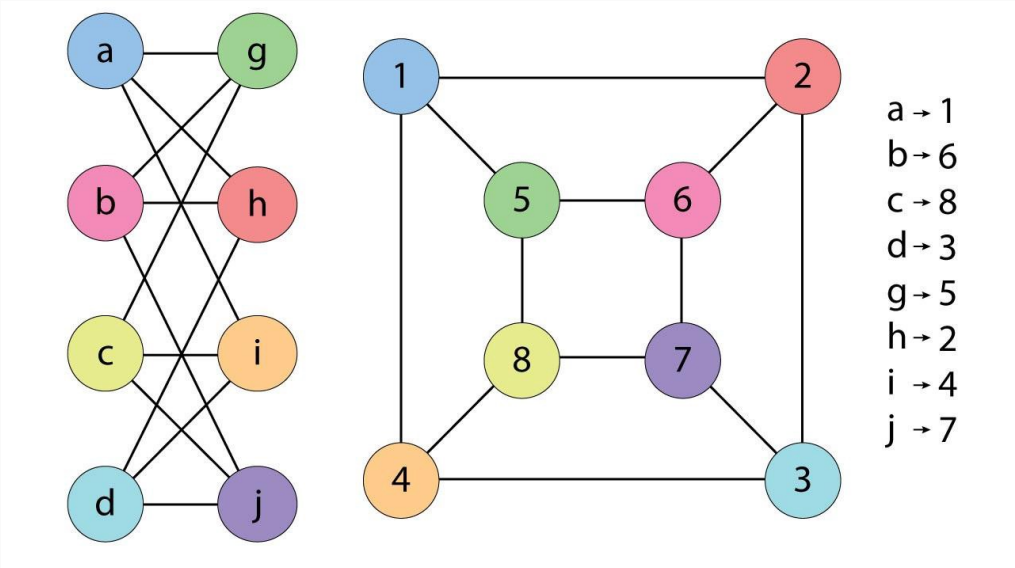
# Graph isomorphism

$G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if there is a bijection $V_1 \longrightarrow V_2$ which preserves edges.



# Graph isomorphism

$G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if there is a bijection $V_1 \longrightarrow V_2$ which preserves edges.




**Idea :** design a machine learning algorithm whose result does not depend on the representation of the input.

# Invariant and equivariant functions

For a permutation $\sigma \in \mathcal{S}_n$, we define ($\mathbb{F} = \mathbb{R}^p$ feature space) :

- for $X \in \mathbb{F}^n$, $(\sigma \star X)_{\sigma(i)} = X_i$
- for $G \in \mathbb{F}^{n \times n}$, $(\sigma \star G)_{\sigma(i_1),\sigma(i_2)} = G_{i_1,i_2}$

# Invariant and equivariant functions

For a permutation $\sigma \in \mathcal{S}_n$, we define ($\mathbb{F} = \mathbb{R}^p$ feature space) :

- for $X \in \mathbb{F}^n$, $(\sigma \star X)_{\sigma(i)} = X_i$
- for $G \in \mathbb{F}^{n \times n}$, $(\sigma \star G)_{\sigma(i_1),\sigma(i_2)} = G_{i_1,i_2}$

$G_1, G_2$ are isomorphic iff $G_1 = \sigma \star G_2$.

# Invariant and equivariant functions

For a permutation $\sigma \in \mathcal{S}_n$, we define ($\mathbb{F} = \mathbb{R}^p$ feature space) :

- for $X \in \mathbb{F}^n$, $(\sigma \star X)_{\sigma(i)} = X_i$
- for $G \in \mathbb{F}^{n \times n}$, $(\sigma \star G)_{\sigma(i_1),\sigma(i_2)} = G_{i_1,i_2}$

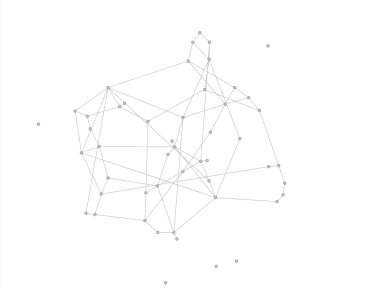
$G_1, G_2$ are isomorphic iff $G_1 = \sigma \star G_2$.

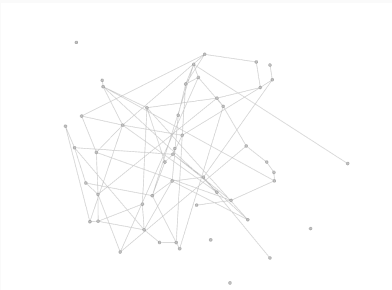
**Definition**
($k = 1$ or $k = 2$)

A function $f : \mathbb{F}^{n^k} \to \mathbb{F}$ is said to be invariant if $f(\sigma \star G) = f(G)$.

A function $f : \mathbb{F}^{n^k} \to \mathbb{F}^n$ is said to be equivariant if $f(\sigma \star G) = \sigma \star f(G)$.

# Learning the graph alignment problem with Siamese GNNs





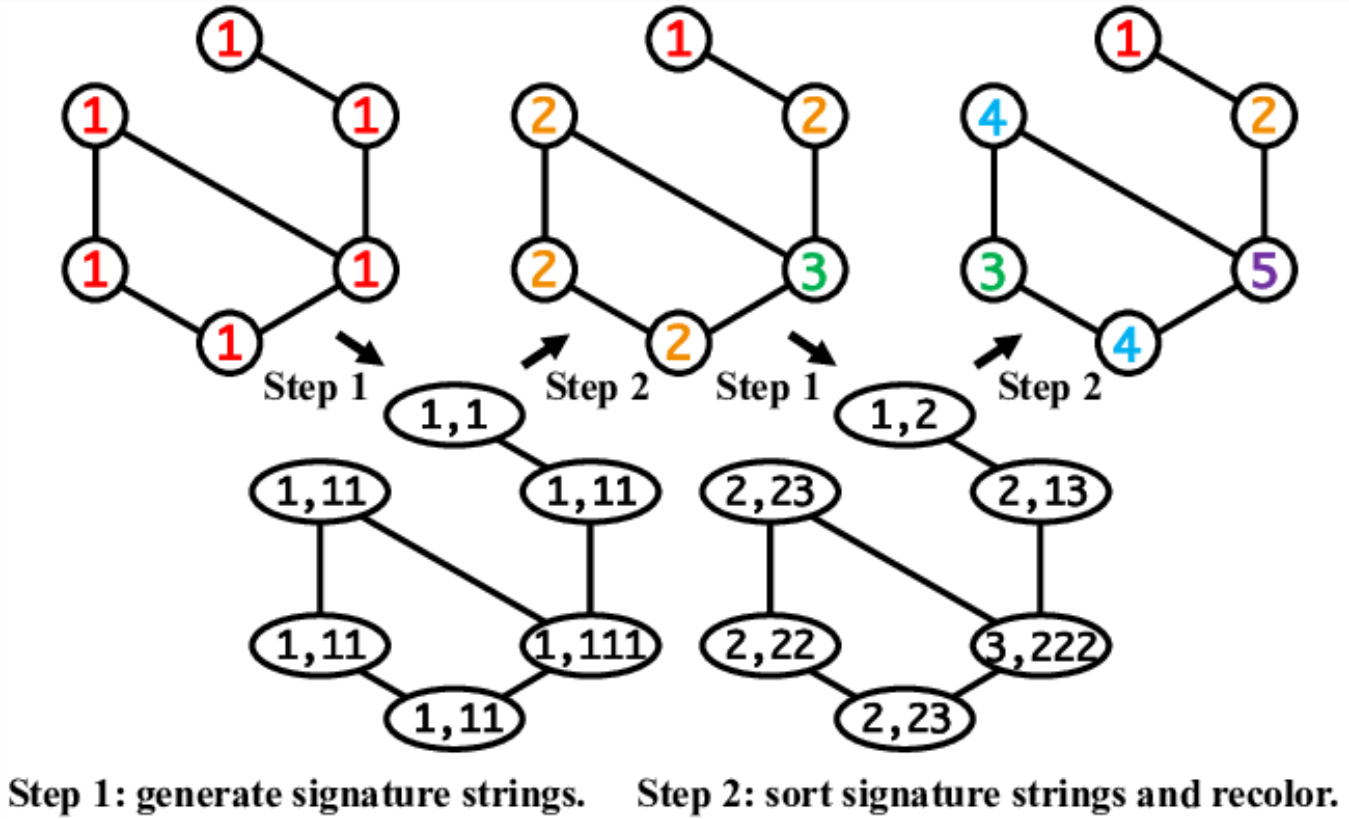
$$G_1 \in \{0,1\}^{n^2} \xrightarrow{\text{GNN}} E_1 \in \mathbb{R}^{n \times b}$$

$$G_2 \in \{0,1\}^{n^2} \xrightarrow{\text{GNN}} E_2 \in \mathbb{R}^{n \times b}$$

$$E_1 E_2^T \in \mathbb{R}^{n^2}$$

- The same GNN is used for both graphs.
- From the node similarity matrix $E_1 E_2^T$, we extract a mapping from nodes of $G_1$ to nodes of $G_2$ (using **Proj** to get a permutation).

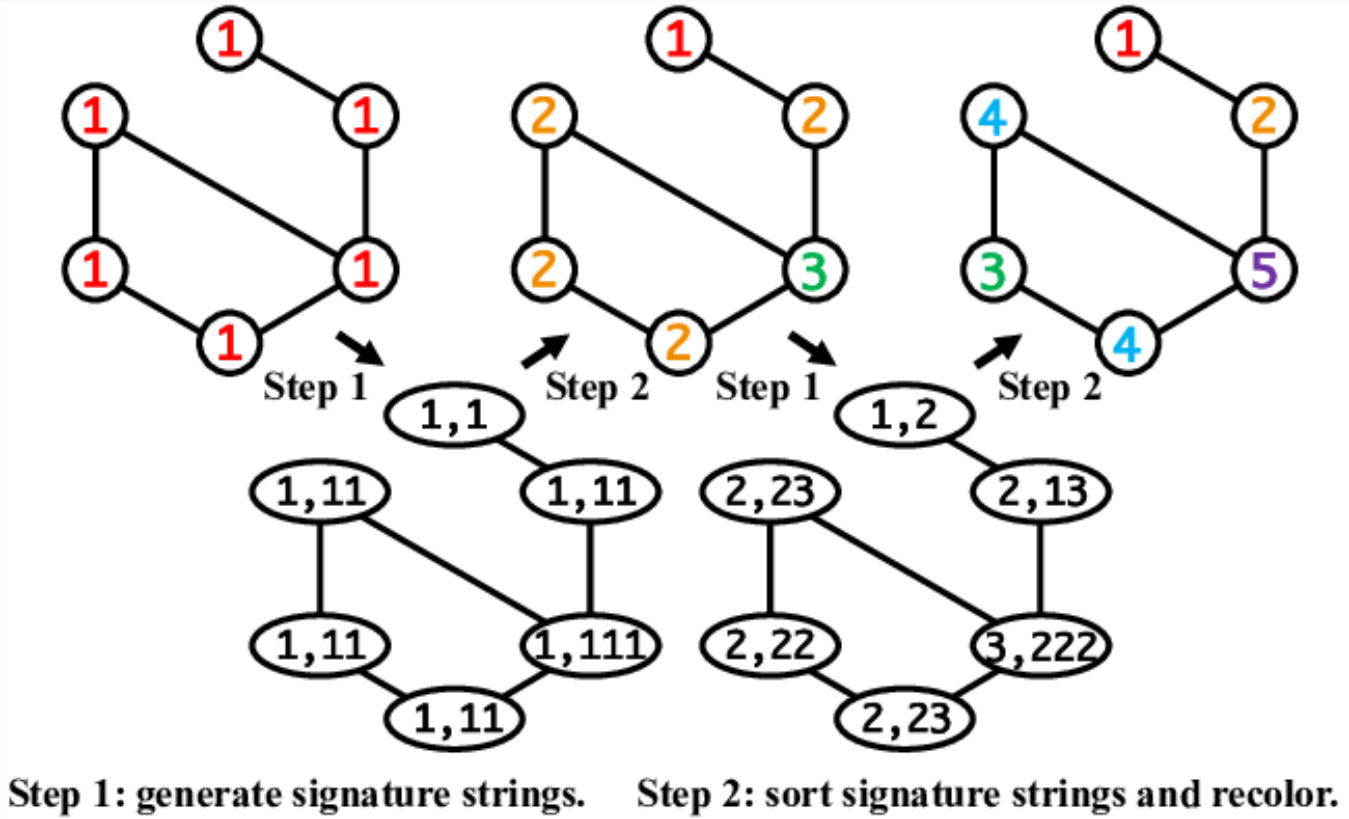
# MPNN and Weisfeiler-Lehman




**Step 1: generate signature strings.** **Step 2: sort signature strings and recolor.**

Xu et al. (2019) : MPNN are as powerful as Weisfeiler-Lehman.

# MPNN and Weisfeiler-Lehman




Xu et al. (2019) : MPNN are as powerful as Weisfeiler-Lehman.

$\min_{D \in \mathcal{D}_n} \|AD - DB\|_F^2 = 0$ iff $G_A$ and $G_B$ cannot be distinguished by 1-WL.

# GNN architecture

For FGNNs Maron et al. (2019), messages are associated with pairs of vertices as opposed to MPNN where messages are associated with vertices.

A residual version of this layer

$$h_{i\to j}^{t+1} = h_{i\to j}^{t} + f_1\left(h_{i\to j}^{t}, \sum_{\ell} h_{i\to \ell}^{t} \odot f_0(h_{\ell\to j}^{t})\right),$$

where $f_0 : \mathbb{R}^d \to \mathbb{R}^d$ and $f_1 : \mathbb{R}^{2d} \to \mathbb{R}^d$ are multilayer perceptrons (MLP) combined with a final graph normalization layer and where $\odot$ is the component-wise multiplication.

# GNN architecture

For FGNNs Maron et al. (2019), messages are associated with pairs of vertices as opposed to MPNN where messages are associated with vertices.

A residual version of this layer

$$h^{t+1}_{i\to j} = h^t_{i\to j} + f_1\left(h^t_{i\to j}, \sum_\ell h^t_{i\to \ell} \odot f_0(h^t_{\ell\to j})\right),$$

where $f_0 : \mathbb{R}^d \to \mathbb{R}^d$ and $f_1 : \mathbb{R}^{2d} \to \mathbb{R}^d$ are multilayer perceptrons (MLP) combined with a final graph normalization layer and where $\odot$ is the component-wise multiplication.

No proof but this architecture is more powerful than MPNN (TBC!).

# Chaining FGNNs

The **second step** takes as input two graphs $G_A$ and $G_B$ as well as a similarity matrix $S^{A\to B}$ and produces two rankings $r^A$ and $r^B$, one for each graph.

Compute the projected permutation $\pi = \textbf{Proj}(S^{A\to B})$ by solving the linear assignment problem : $\max_{\pi\in\mathcal{S}_n} \sum_i S^{A\to B}_{i\pi(i)}$.

Intuition : the entry $S^{A\to B}_{ij}$ is a measure of the similarity between nodes $i \in G_A$ and $j \in G_B$. Hence $\pi$ is a mapping from nodes in $G_A$ to nodes in $G_B$ which approximately solves the graph matching problem.

The goal of chaining is to improve incrementally this approximation.

# Chaining FGNNs

The **second step** takes as input two graphs $G_A$ and $G_B$ as well as a similarity matrix $S^{A\to B}$ and produces two rankings $r^A$ and $r^B$, one for each graph.

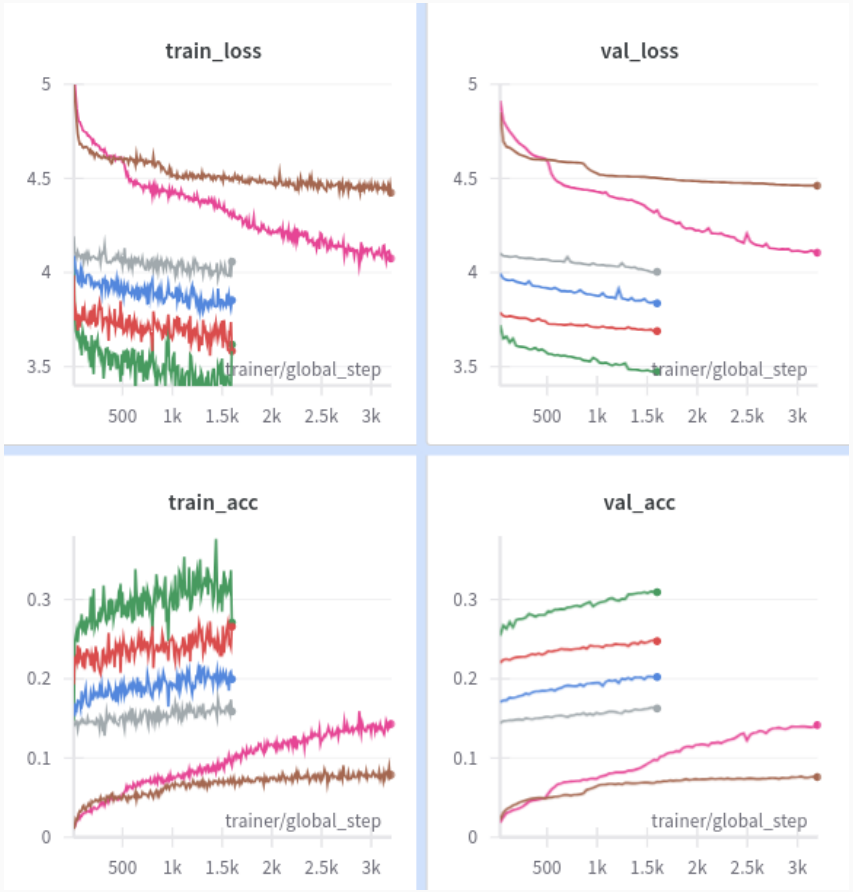
Compute the projected permutation $\pi = \mathbf{Proj}(S^{A\to B})$ by solving the linear assignment problem : $\max_{\pi\in\mathcal{S}_n} \sum_i S^{A\to B}_{i\pi(i)}$.

Intuition : the entry $S^{A\to B}_{ij}$ is a measure of the similarity between nodes $i \in G_A$ and $j \in G_B$. Hence $\pi$ is a mapping from nodes in $G_A$ to nodes in $G_B$ which approximately solves the graph matching problem.

The goal of chaining is to improve incrementally this approximation.

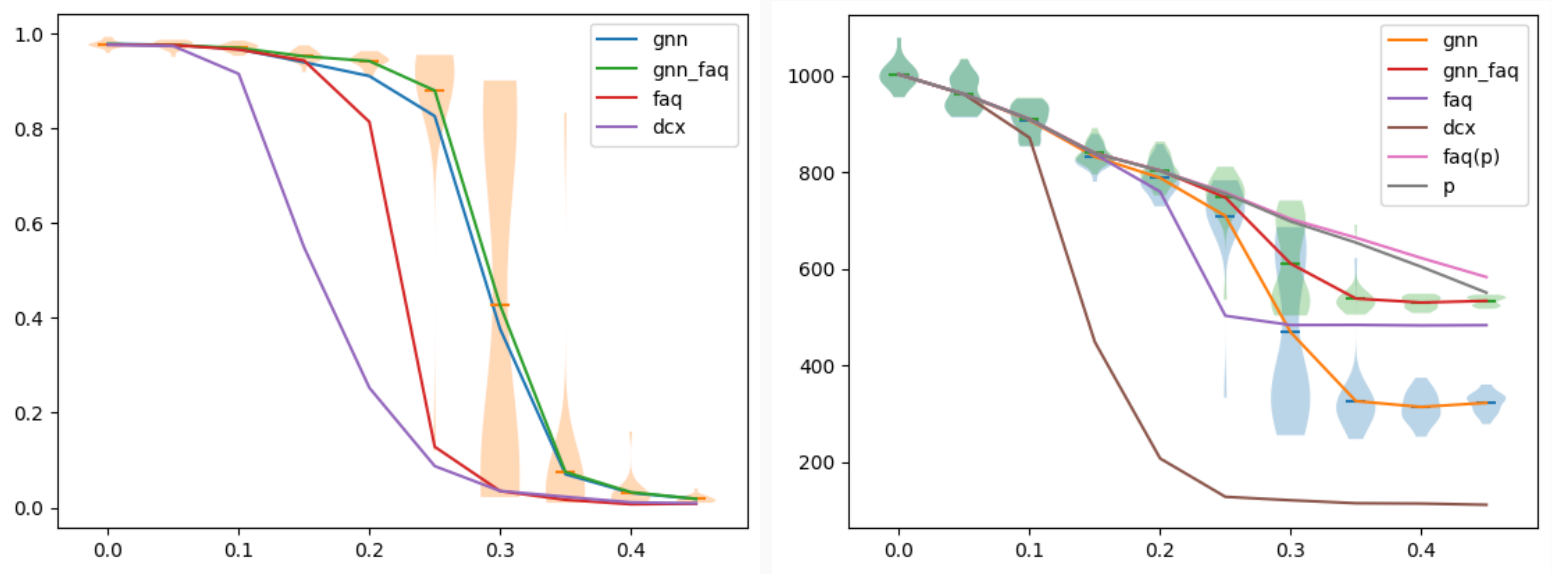
For this, we need to transfer the information contained in $\pi$ into node features for both graphs : compute a score for each node $i$ in the graph $A$ by $s(i) = \sum_j A_{ij} B_{\pi(i)\pi(j)}$. We can then sort the nodes in $A$ in decreasing order of their scores $s(i)$ and obtain a ranking $r^A \in \mathcal{S}_n$ for the nodes in $A$. In order to get the ranking $r^B$, we use the permutation $\pi : G_A \to G_B$ as follows : $r^B_i = \pi\left(r^A_i\right)$.

# Training procedure



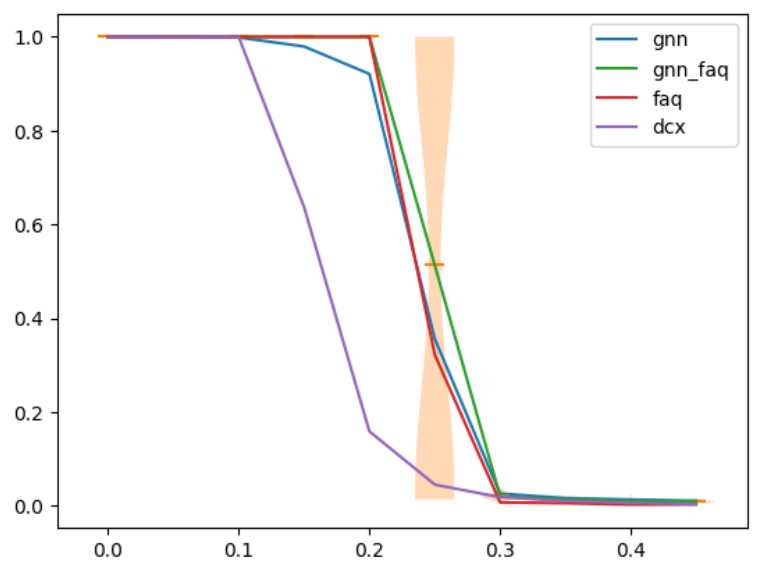

Training chained GNNs. Each color corresponds to a different training and GNN.

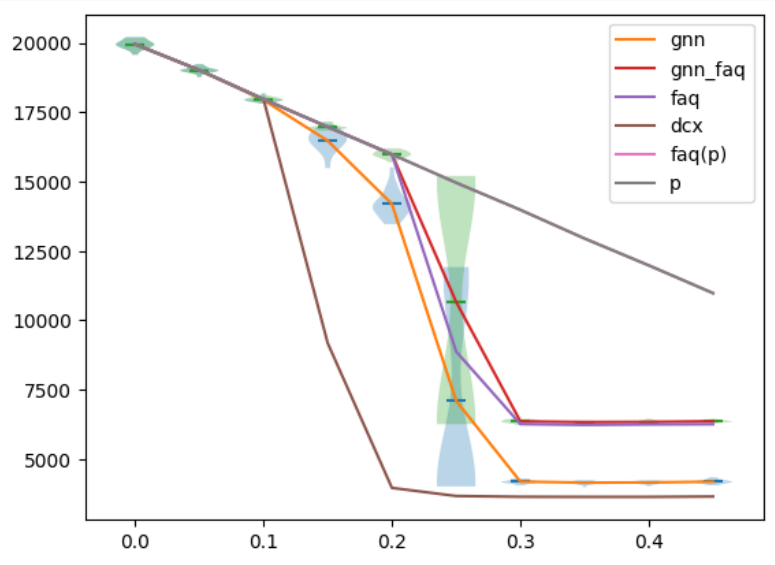
# Erdős-Rényi ($n = 500, d = 4$)




Accuracy (left) and number of common edges (right) as a function of noise.

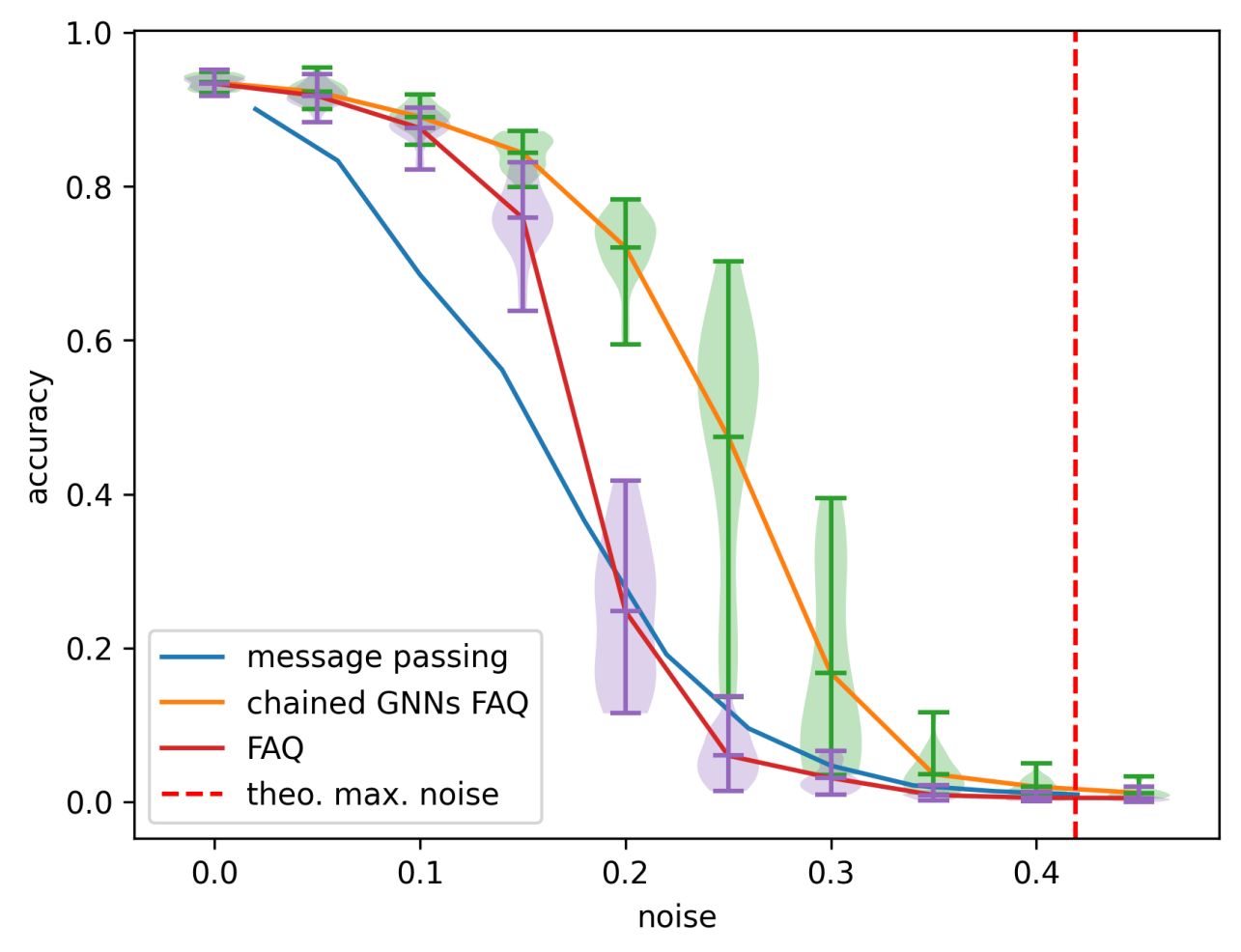
# Erdős-Rényi ($n = 500, d = 80$)




Accuracy (left) and number of common edges (right) as a function of noise.

# Erdős-Rényi ($n = 1000, d = 3$, training 0.25)



# Training : optimal noise level

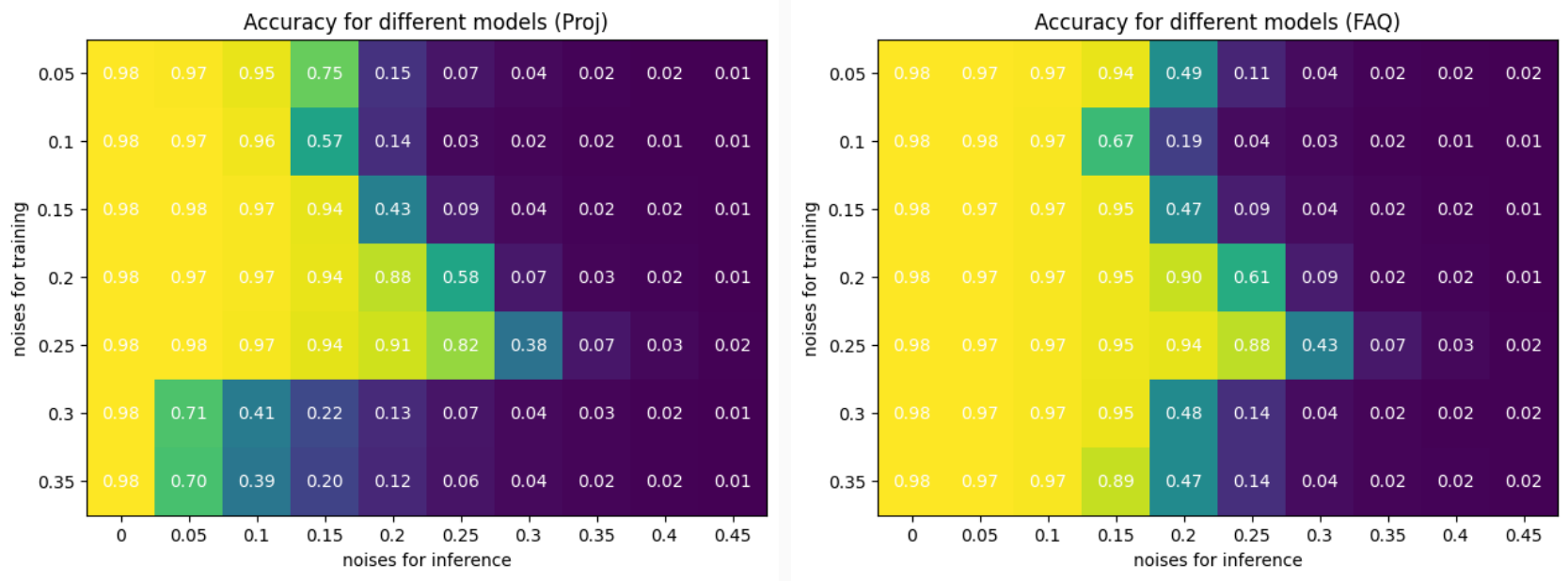



**Figure 2 –** Each line corresponds to a chained FGNN trained at a given level of noise and evaluated across all different level of noises. Performances are **acc** for sparse Erdős-Rényi (ER 4) with last operation **Proj** (left) or **FAQ** (right).

# Varying the number of steps in chaining

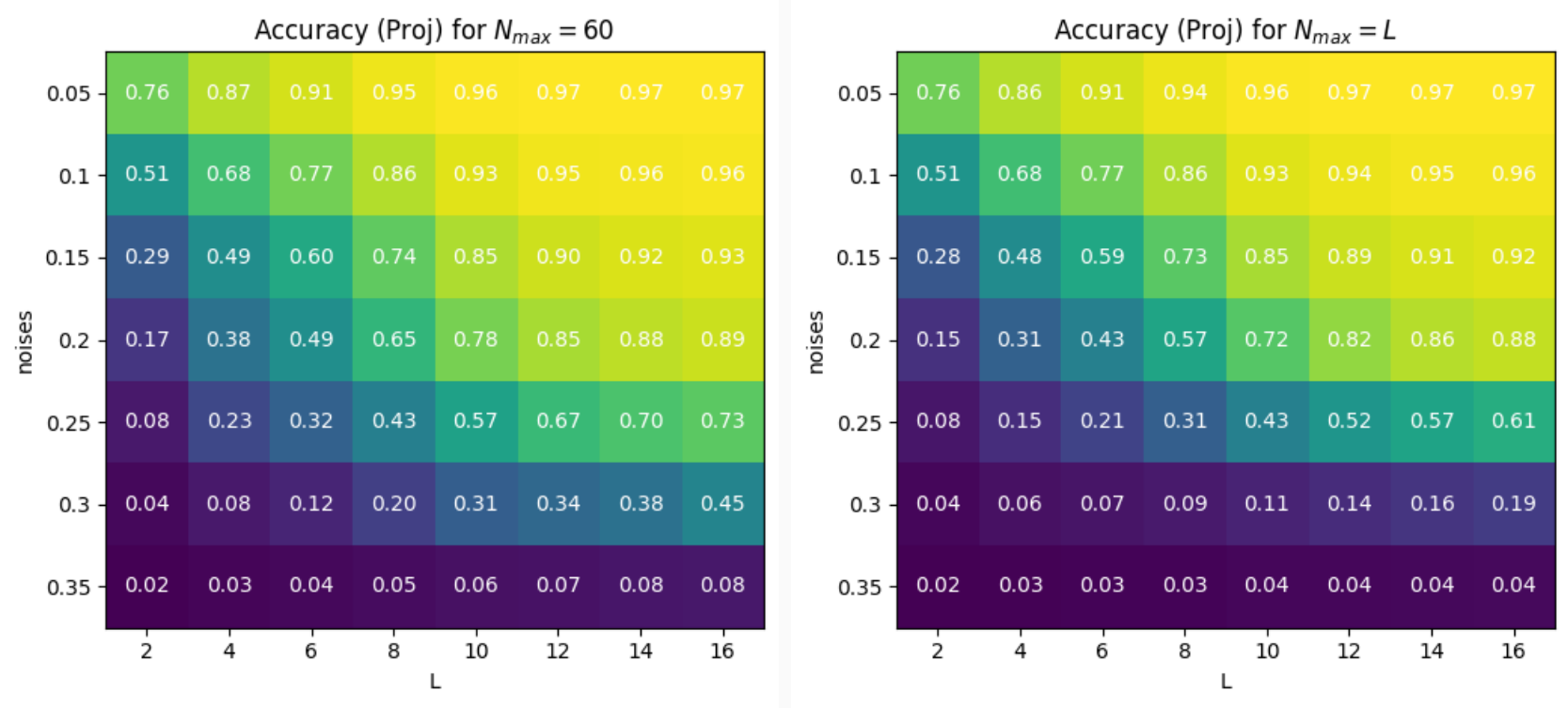
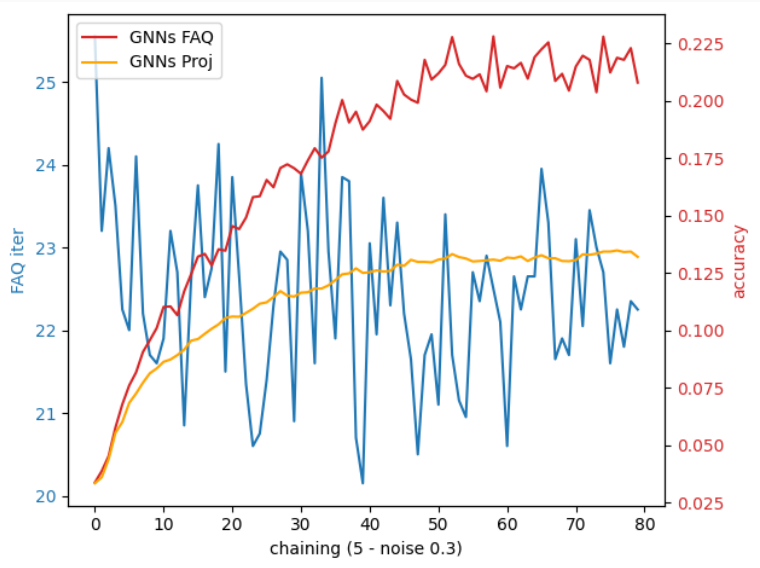
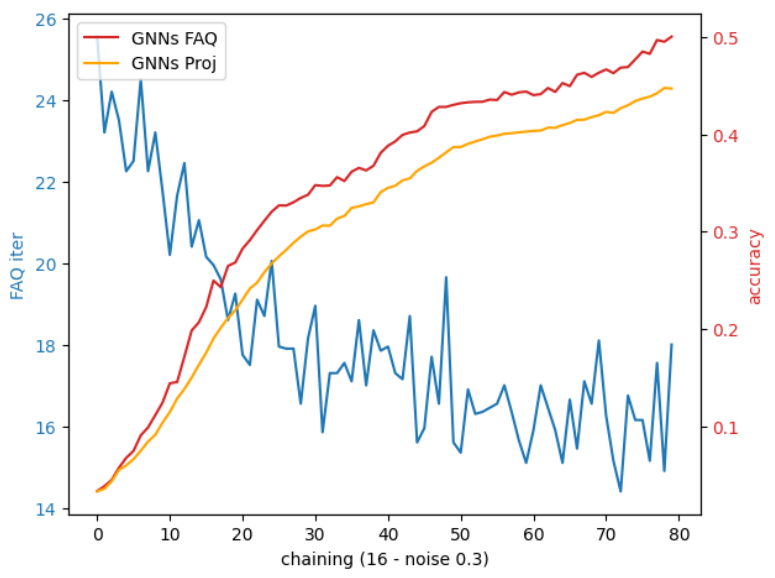


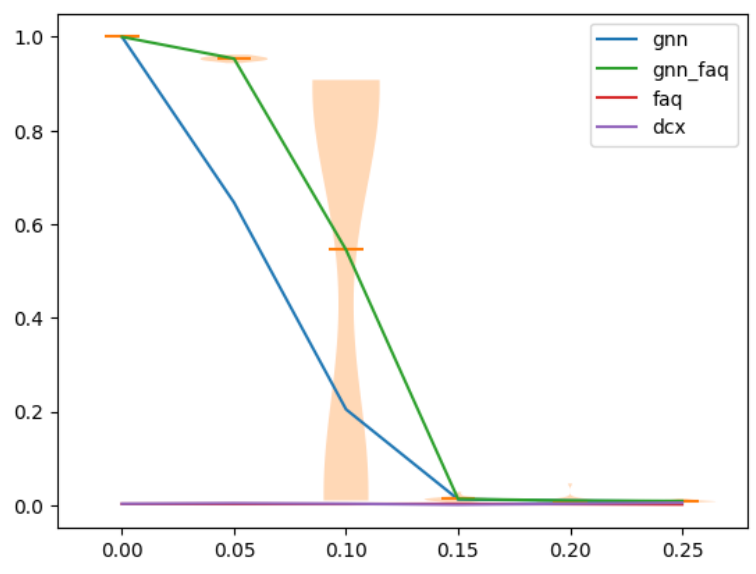

**Figure 3 –** Accuracy for a chained FGNN on sparse Erdős-Rényi graphs (trained at optimal noise level). Each column corresponds to a different $L$, with **looping** and $N_{\max} = 60$ on the left and no looping, $N_{\max} = L$ on the right. Each line corresponds to a different noise level for inference.

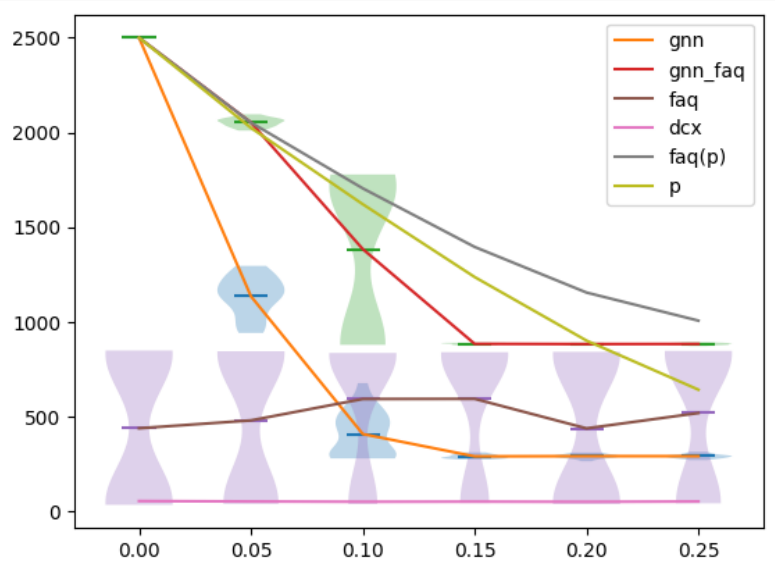
# Trading CPU for GPU



# Regular ($n = 500, d = 10$)




Accuracy (left) and number of common edges (right) as a function of noise.

# Conclusion

- General problem : how to learn a permutation ?
- What is the role of chaining ?
  Can it be interpreted as implementing an interpretable iterative algorithm ?
  Similar to diffusions, can it be interpreted as a denoising ?
- GNNs chaining is working for correlated random graphs !
- Results with GNNs corroborate theoretical predictions.
- New hard instances (regular graphs) are solved with GNNs (and I do not know of any alternative solutions).

# Thank You!

# Références


L. Babai. Graph isomorphism in quasipolynomial time. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 684–697, 2016.

M. Böther, O. Kißig, M. Taraz, S. Cohen, K. Seidel, and T. Friedrich. What's wrong with deep learning in tree search for combinatorial optimization. *ICLR*, 2022.

Q. Cappart, D. Chételat, E. B. Khalil, A. Lodi, C. Morris, and P. Veličković. Combinatorial optimization and reasoning with graph neural networks. *Journal of Machine Learning Research*, 24(130) :1–61, 2023.

J. Ding, Z. Ma, Y. Wu, and J. Xu. Efficient random graph matching via degree profiles. *Probability Theory and Related Fields*, 179 :29–115, 2021.

L. Ganassali, L. Massoulié, and M. Lelarge. Correlation detection in trees for planted graph alignment. *arXiv preprint arXiv :2107.07623*, 2021a.

L. Ganassali, L. Massoulié, and M. Lelarge. Impossibility of partial recovery in the graph alignment problem. In *Conference on Learning Theory*, pages 2080–2102. PMLR, 2021b.

C. Mao, Y. Wu, J. Xu, and S. H. Yu. Random graph matching at otter's threshold via counting chandeliers. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 1345–1356, 2023.

**Temporary page!**

LaTeX was unable to guess the total number of pages correctly. As there wa
some unprocessed data that should have been added to the final page th
extra page has been added to receive it.

If you rerun the document (without altering it) this surplus page will go
away, because LaTeX now knows how many pages to expect for this docume