Chaining 2-FWL GNNs for Combinatorial Graph Alignment

Marc Lelarge
INRIA, École Normale Supérieure, PSL Research University
Paris, France
marc.lelarge@inria.fr

Abstract

For the combinatorial graph alignment problem (GAP) — finding the node correspondence that maximizes the number of common edges (**nce**) between two unlabeled graphs — the classical Frank–Wolfe heuristic **FAQ** remains a strong baseline once it is properly initialized, while existing GNN approaches struggle in the purely structural setting. We introduce a chaining procedure: a sequence of Folklore-type (2-FWL) GNNs in which each network is trained with cross-entropy after decoding the previous network's similarity matrix and ranking nodes by their current alignment quality. This non-differentiable ranking step injects discrete combinatorial feedback at every stage; at inference, we iterate the final network and keep the candidate with highest observed **nce**. On sparse Erdős–Rényi graphs at noise level 0.25, chained FGNNs used to initialize **FAQ** reach 85% accuracy versus 13% for **FAQ** initialized from the convex relaxation, and essentially 0% for prior GNN methods. On correlated regular graphs, where MPNNs with constant features produce identical node embeddings (1-WL fails to refine) and **FAQ**'s convex initialization is degenerate, chaining is the only method among those we evaluate, including Gromov–Wasserstein solvers, that recovers a non-trivial alignment. On three real-world benchmarks (yeast PPI, coauthorship, and road networks), recent comparisons underestimate **FAQ** by initializing it from a uniform doubly stochastic matrix; initialized from the convex relaxation it already surpasses prior reported numbers, and dataset-specific chained FGNNs further improve on this strengthened baseline. Code and pretrained chains available at: `https://github.com/mlelarge/chaining-gnn-graph-alignment`.

1 Introduction

Graph neural networks are attractive for graph combinatorial problems because they can learn structural representations from simulated instances, but in graph alignment they have not yet matched strong classical relaxations in the purely structural setting. We study whether learned representations can improve such solvers on one concrete problem: the combinatorial graph alignment problem (GAP), which seeks the node correspondence between two unlabeled graphs that maximizes the number of common edges. GAP encompasses graph isomorphism as a special case and, in its general form, reduces to the NP-hard quadratic assignment problem (QAP). The classical solver we build on is the Fast Approximate QAP algorithm (**FAQ**) of Vogelstein et al. [2015], an existing Frank–Wolfe heuristic for a continuous relaxation of QAP whose output depends on its initialization (Section 2.2). **FAQ** is not a contribution of this paper: our contributions are a corrected evaluation of it and a learned way to initialize it. The graph matching problem can be cast as a GAP, and machine learning methods have been widely applied to it in pattern recognition [Conte et al., 2004], computer vision [Sun et al., 2020], and social network analysis [Narayanan and Shmatikov, 2008] (Section A.10), motivated by the fact that in noisy real-world data the ground-truth matching may deviate from the mathematically

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

Table 1: Normalized alignment quality $\frac{\text{ALG}}{\text{REF}}$ (in %) for sparse, dense and regular random graphs. **Proj** and **FAQ** are used to produce a permutation from the convex relaxation solution D_{cx} or from the similarity matrix computed by our chained FGNN (ChFGNN).

TYPE OF GRAPHS		SPARSE	DENSE	REGULAR
BASELINES (NON-NEURAL)	**PROJ**(D_{cx})	17.3	24.4	2.9
	FAQ(D_{cx})	67.1	53	27
BASELINES (NEURAL)	FGNN **PROJ**	17.8	23.6	6.7
	FGNN **FAQ**	71.1	47	54
CHAINING	CHFGNN **PROJ**	95.8	44	67.1
CHAINING	CHFGNN **FAQ**	**98.8**	**77.4**	**81.8**

optimal solution. We focus strictly on the combinatorial optimization setting, where only the optimal solution is relevant, and accordingly use the term *graph alignment* rather than *graph matching*.

Iterative refinement through chaining. We introduce *chaining*: a sequential training procedure in which each GNN consumes a discrete ranking induced by the alignment decoded from the previous GNN. In our experiments, this ranking feedback produces similarity matrices that are substantially better initializations for **FAQ** on hard sparse and regular instances. Table 1 illustrates our key results across different graph types. To compare how close the different methods get to a common high-quality reference, we measure their *normalized alignment quality*[1] as $\frac{\text{ALG}}{\text{REF}}$ in percent, where ALG is the number of aligned edges obtained by the method and REF is the corresponding reference score. Our chained GNNs, particularly when coupled with **FAQ** post-processing (ChFGNN **FAQ**), consistently achieve the best performance.

We train and evaluate primarily on synthetic correlated random graphs — the standard benchmarking methodology for combinatorial optimization [Skorin-Kapov, 1990, Taillard, 1991] — because they let us control problem difficulty via a single noise parameter and run fine-grained comparisons; we also evaluate on three real-world graph pairs (biology, social, road networks) to test transfer.

We formulate the graph alignment problem as a supervised learning task in Section 2. While prior learning-based approaches have not matched **FAQ**(D_{cx}) in the purely structural setting (Section 4), we introduce a method that improves upon **FAQ**(D_{cx}) on hard sparse and regular instances.

Contributions. (i) A corrected evaluation of **FAQ**: initialized from the convex relaxation D_{cx} as advocated by Lyzinski et al. [2015], instead of the uniform matrix used in recent comparisons, it is a much stronger baseline than reported (Section A.2). (ii) Chaining, a training and inference procedure for learning **FAQ** initializations, with an FGNN backbone expressive enough to avoid the immediate 1-WL obstruction faced by constant-feature MPNNs on regular graphs. (iii) Comparisons on identical test pairs with the Gromov–Wasserstein solvers BAPG and FGWAlign, in addition to **FAQ** and FUGAL (Appendix A.4). The empirical gains are largest on hard sparse Erdős–Rényi graphs and on correlated regular graphs; dense ER and real-world results are more modest. We see iterative refinement with discrete combinatorial feedback as a promising direction for combinatorial problems with a natural ranking interpretation; we do not claim it transfers without modification to arbitrary CO problems.

Mathematical notations. Let $G = (V, E)$ be a simple graph with $V = \{1, \dots, n\}$ and adjacency matrix $A \in \{0,1\}^{n\times n}$, where $A_{ij} = 1$ if $(i, j) \in E$ and 0 otherwise. Let $\mathcal{S}_n$ denote the set of permutations of V, with each $\pi \in \mathcal{S}_n$ associated to a permutation matrix $P \in \{0,1\}^{n\times n}$ defined by $P_{ij} = 1$ iff $\pi(i) = j$. The set of doubly stochastic matrices is denoted $\mathcal{D}_n$. For $A, B \in \mathbb{R}^{n\times n}$, the Frobenius inner product and norm are $\langle A, B\rangle = \operatorname{trace}(A^\top B)$ and $\|A\|_F = \sqrt{\langle A, A\rangle}$, respectively.

2 From combinatorial optimization to a supervised learning task

This section introduces the graph alignment problem (GAP) from a combinatorial optimization perspective, presents the **FAQ** algorithm, and describes how we formulate GAP as a learning problem using synthetic datasets with controllable difficulty.

[1] In Table 1, REF is the same denominator for all methods in a column: sparse/dense Erdős–Rényi at $p_{\text{noise}} = 0.25$ and regular graphs at $p_{\text{noise}} = 0.1$. It is a near-optimal reference score, not a certified optimum for every instance.

2.1 Graph alignment in combinatorial optimization

Problem formulation. Given two $n \times n$ adjacency matrices A and B representing graphs G_A and G_B, the graph alignment problem seeks to find the permutation that best aligns their structures. Formally, we minimize the Frobenius norm:

$$\text{GAP}(A, B) = \min_{P \in \mathcal{S}_n} \|AP - PB\|_F^2. \tag{1}$$

Expanding the right-hand term, we see that minimizing (1) is equivalent to maximizing the number of matched edges:

$$\max_{P \in \mathcal{S}_n} \langle AP, PB \rangle = \max_{\pi \in \mathcal{S}_n} \sum_{i,j} A_{ij} B_{\pi(i)\pi(j)}. \tag{2}$$

This formulation connects GAP to the broader class of Quadratic Assignment Problems (QAP) [Burkard et al., 1998].

Computational complexity. The GAP is NP-hard: when G_B is a path or a cycle it is the Hamiltonian path or cycle problem, when G_B is two disjoint cliques of size $n/2$ it is minimum bisection, and in general (2) is a maximum common subgraph problem, which is APX-hard [Crescenzi et al., 1995].

Performance metrics. We denote an optimal solution as $\pi^{A \to B}$. We evaluate alignment quality using two complementary metrics (that should be maximized), Accuracy and Number of common edges:

$$\textbf{acc}(\pi, \pi^{A \to B}) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(\pi(i) = \pi^{A \to B}(i)), \qquad \textbf{nce}(\pi) = \frac{1}{2} \sum_{i,j} A_{ij} B_{\pi(i)\pi(j)}. \tag{3}$$

Accuracy measures the fraction of correctly matched nodes, while the number of common edges quantifies structural similarity. In Table 1, the ratio $\frac{\text{ALG}}{\text{REF}}$ is computed as $\frac{\textbf{nce}(\pi^{\text{ALG}})}{\text{REF}}$, with the same reference score used for all methods in a graph family. Note that even if this ratio is one, the accuracy may still be low if the GAP has no unique solution (as illustrated on real datasets in Section 5.3).

2.2 Continuous relaxations and the FAQ algorithm

Since the discrete optimization in (1) is intractable, continuous relaxations have been proposed where the discrete permutation set $\mathcal{S}_n$ is replaced by the continuous set of doubly stochastic matrices $\mathcal{D}_n$ in (1) or (2):

- **Convex relaxation**:

$$\arg \min_{D \in \mathcal{D}_n} \|AD - DB\|_F^2 = D_{\text{cx}} \tag{4}$$

  This yields a convex optimization problem with guaranteed global optimum.

- **Indefinite relaxation**:

$$\max_{D \in \mathcal{D}_n} \langle AD, DB \rangle \tag{5}$$

  This non-convex formulation often provides better solutions but is NP-hard in general due to its indefinite Hessian [Pardalos and Vavasis, 1991].

Solution extraction. Both relaxations produce doubly stochastic matrices D that must be projected to permutation matrices. This projection solves the linear assignment problem $\max_{P \in \mathcal{S}_n} \langle P, D \rangle$, efficiently solved by the Hungarian algorithm in $O(n^3)$ time [Kuhn, 1955]. We denote this projection as $\textbf{Proj}(D) \in \mathcal{S}_n$.

FAQ algorithm. The Fast Approximate Quadratic (**FAQ**) algorithm proposed by Vogelstein et al. [2015] approximately solves the indefinite relaxation (5) using Frank-Wolfe optimization and then projects this solution in $\mathcal{S}_n$. Unlike the convex relaxation, **FAQ**'s performance depends critically on initialization. We denote the **FAQ** solution with initial condition D as $\textbf{FAQ}(D) \in \mathcal{S}_n$. As demonstrated in Lyzinski et al. [2015], **FAQ** often significantly outperforms simple projection: $\textbf{FAQ}(D_{\text{cx}})$ typically yields much better solutions than $\textbf{Proj}(D_{\text{cx}})$, especially for challenging instances. **This improvement motivates our approach of providing FAQ with better initializations through learned similarity matrices.**

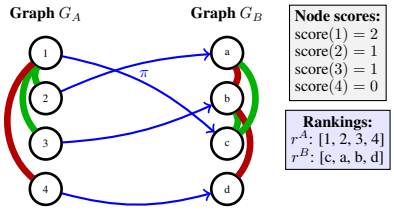


Figure 1: Illustration of Step 2. The permutation π maps 1→c, 2→a, 3→b, 4→d. Green edges show matches: edge 1-2 with a-c, and edge 1-3 with b-c. Node 1 has the highest score (2 matched edges), nodes 2 and 3 each have 1 matched edge, and node 4 has no matched edges.

2.3 Synthetic datasets: controlled difficulty through noise

Connection to graph isomorphism. When G_A and G_B are isomorphic (GAP$(A, B) = 0$), the alignment problem reduces to graph isomorphism, whose complexity remains open (neither known to be in P nor NP-complete; quasipolynomial by Babai [2016]). We study a natural generalization, noisy graph isomorphism, where a noise level controls the difficulty: at zero noise the graphs are isomorphic, and they become increasingly different as the noise grows.

Correlated random graph model. Our datasets consist of correlated random graph pairs (G_A, G_B) with identical marginal distributions and controllable correlation: we generate correlated graphs G_A and G_B with known alignment, apply a random permutation $\pi^\star \in \mathcal{S}_n$ to G_B to obtain G'_B, and use the triplets $(G_A, G'_B, \pi^\star)$ for supervised learning.

We employ three graph families—**Bernoulli**, **Erdős-Rényi**, and **Regular**—with parameters: **Number of nodes**: n; **Average degree**: d; **Noise level**: $p_{\text{noise}} \in [0, 1]$, see Section A.1 for precise definitions. The noise parameter controls edge correlation: the graphs G_A and G_B (before applying the random permutation) share $(1 - p_{\text{noise}})nd/2$ edges on average (with $p_{\text{noise}} = 0$ yielding isomorphic graphs). For low noise levels, we expect $\pi^\star = \pi^{A\to B}$, providing clean supervision. At high noise the planted permutation $\pi^\star$ may no longer be optimal, introducing label noise. We therefore report the number of common edges **nce** as our primary metric in addition to accuracy: **nce** is computed directly from the predicted alignment and the two graphs, and is *independent* of $\pi^\star$. A method that finds a better alignment than the planted one simply attains a higher **nce**.

3 Learning via Chaining GNNs

Overview. The chaining procedure iteratively refines alignment estimates through three operations: (1) computing node similarities, (2) extracting and scoring the current best permutation, and (3) using these scores to generate improved node features.

3.1 Chaining procedure

Step 1: Initial feature extraction and similarity computation. Given a mapping f that extracts node features from a graph's adjacency matrix $A \in \{0, 1\}^{n\times n}$ and outputs $f : \{0, 1\}^{n\times n} \to \mathbb{R}^{n\times d}$, we compute node feature matrices $f(A)$ and $f(B)$ for graphs G_A and G_B. The initial similarity matrix captures pairwise node similarities via their feature dot products:

$$S^{A\to B,(0)} = f(A)f(B)^T \in \mathbb{R}^{n\times n}. \tag{6}$$

Here, $S_{ij}^{A\to B,(0)}$ measures the similarity between node $i \in G_A$ and node $j \in G_B$ based on their learned features.

Step 2: Permutation extraction and node quality scoring. From a similarity matrix $S^{A\to B}$, we extract the best permutation estimate by solving the linear assignment problem: $\pi = \textbf{Proj}(S^{A\to B})$ where $\pi = \arg\max_{\pi\in\mathcal{S}_n} \sum_i S^{A\to B}_{i\pi(i)}$. This permutation $\pi : G_A \to G_B$ represents our current best guess for the optimal alignment $\pi^{A\to B}$. To evaluate alignment quality, we compute a score for each node i in graph A: score$(i) = \sum_j A_{ij}B_{\pi(i)\pi(j)}$. Intuitively, score$(i)$ counts the number of edges incident to node i that are correctly matched under the current permutation π—higher scores indicate better-aligned nodes (see Figure 1). We then rank nodes in G_A by decreasing score, obtaining a ranking $r^A \in \mathcal{S}_n$ such that:

$$\text{score}(r^A(1)) \geq \text{score}(r^A(2)) \geq \cdots \geq \text{score}(r^A(n)). \tag{7}$$

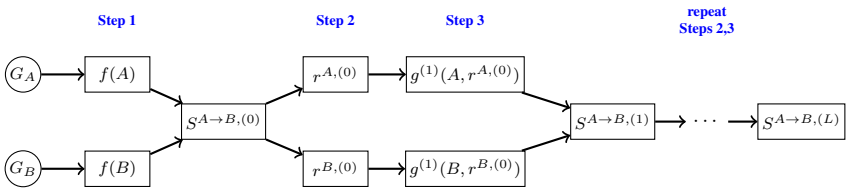


Figure 2: Overview of the chaining procedure. Starting from input graphs G_A and G_B, we first (1) extract features and compute similarities, then iteratively (2) rank nodes by alignment quality, and (3) use rankings to enhance features and similarities.

We transfer this ordering to G_B through the estimated permutation by setting $r^B(k) = \pi(r^A(k))$ for each rank k. Thus nodes with the same rank in r^A and r^B are matched by the current permutation: equivalently, $\pi(i) = r^B((r^A)^{-1}(i))$ (see Figure 1). The top-ranked pairs correspond to nodes whose incident edges are most consistent with the estimated alignment.

Symmetry, ties, and the choice of score. The score belongs to a matched *pair*: computed from the side of G_B, $\sum_y B_{\pi(i)y}A_{i\pi^{-1}(y)} = \text{score}(i)$, so both endpoints of a matched pair receive the same rank, and exchanging G_A and G_B (with $\pi \leftrightarrow \pi^{-1}$) yields the same rank fields up to the order of ties. Ties are frequent since scores are small integers; they are broken by the order returned by the sorting routine (`numpy.argsort`), which depends only on the score vector and uses no label information. We rank by the raw count rather than by a degree-normalized ratio because $\sum_i \text{score}(i) = 2\,\mathbf{nce}(\pi)$ is the per-node decomposition of the objective, and because the raw count promotes high-degree correctly matched nodes, which constrain the most neighboring assignments in the next stage; Section 5.1 shows that a chain retrained with the degree-normalized score performs on par, so this choice is second-order.

Step 3: Ranking-enhanced feature learning. We now incorporate the ranking information to compute improved node features. Using a mapping $g : \{0,1\}^{n\times n} \times \mathcal{S}_n \to \mathbb{R}^{n\times d}$ that takes both the graph structure and node rankings as input, we compute enhanced feature matrices $g(A, r^A)$ and $g(B, r^B)$. The new similarity matrix is:

$$S^{A\to B} = g(A, r^A)g(B, r^B)^T \in \mathbb{R}^{n\times n}. \tag{8}$$

This ranking-enhanced similarity matrix $S^{A\to B,(1)} = g(A, r^{A,(0)})g(B, r^{B,(0)})^T$ should be more informative than the initial $S^{A\to B,(0)}$ since it incorporates knowledge about which nodes align well. Consequently, we expect $\mathbf{Proj}(S^{A\to B,(1)})$ to be closer to the optimal $\pi^{A\to B}$ than $\mathbf{Proj}(S^{A\to B,(0)})$.

What the ranking provides. The next network does not reweight the previous similarities: it recomputes all n^2 pair similarities from (A, r^A) and (B, r^B) and solves a new assignment from scratch, so no match is frozen. What the ranking adds is evidence that did not exist before the projection. Matched pairs share a rank value across the two graphs, so a bilinear FGNN layer (Section 3.3) can count, for a candidate pair (u, v), the neighbors of u and of v that carry the same rank value. For the true partner this is the number of common neighbors among correctly matched pairs; for a wrong candidate it is close to zero. The high-ranked matches thus serve as witnesses for re-assigning the low-ranked nodes, as in seeded graph alignment, where a partially correct correspondence is extended through one- and two-hop witnesses [Kazemi et al., 2015, Lubars and Srikant, 2018, Mossel and Xu, 2020, Yu et al., 2021a]. Section 5.1 measures this correction directly.

Iterative refinement. We iterate steps 2 and 3 with different learned mappings $g^{(1)}, g^{(2)}, \ldots$ at each step, so that the chain of mappings f, r, $g^{(1)}$, $g^{(2)}$, $\ldots$ produces a sequence of similarity matrices $S^{A\to B,(0)}$, $S^{A\to B,(1)}$, $\ldots$ — each refining the previous via the discrete ranking signal. This creates a bootstrap effect where every step leverages the improved alignment from the previous one (Figure 2).

3.2 Training and inference with chained GNNs

The ranking step r is not differentiable, preventing end-to-end training. Instead, we train each GNN in the chain sequentially, which proves both practical and effective. This approach allows our method to explicitly learn from discrete permutation decisions at each step, which is crucial for the iterative improvement process.

Sequential training procedure. We implement $f, g^{(1)}, \ldots, g^{(k)}$ as graph neural networks and train them sequentially, each optimized to improve on the output of the previous stage. For training pairs (G_A, G_B) with planted permutation $\pi^\star$, we use the cross-entropy loss

$$\mathcal{L}(S^{A\to B}, \pi^\star) \;=\; -\sum_i \log\left(\mathrm{softmax}(S^{A\to B}_{i,\cdot})\right)_{\pi^\star(i)}, \tag{9}$$

which encourages the similarity matrix to assign high mass to the correct correspondences. The full schedule is: train f by minimizing $\mathcal{L}(S^{A\to B,(0)}, \pi^\star)$; then fix f, compute $r^{A,(0)}$ and $r^{B,(0)}$ on the training data and train $g^{(1)}$ by minimizing $\mathcal{L}(S^{A\to B,(1)}, \pi^\star)$; then fix $g^{(1)}$, compute $r^{A,(1)}, r^{B,(1)}$ and train $g^{(2)}$ by minimizing $\mathcal{L}(S^{A\to B,(2)}, \pi^\star)$; and so on for $g^{(3)}, \ldots, g^{(k)}$.

This sequential approach ensures that each GNN learns to improve upon the alignment quality achieved by all previous networks in the chain.

Why early errors do not compound. First, every stage is trained on the *actual* outputs of the previous stage on the training set, with the planted permutation as target: it sees at training time the same imperfect rankings that it receives at inference, including misassigned nodes, and the loss asks precisely for those to be corrected; this removes the train/test mismatch between stages that drives error accumulation in cascaded predictors [Ross et al., 2011]. Second, nothing is frozen: the ranking is an input feature, not a constraint, and each stage re-solves the full assignment. Third, errors arrive labeled as unreliable, since a misassigned node preserves few incident edges and sits at the bottom of the ordering. Appendix A.9 measures the resulting correction at inference.

Inference procedure. On a new pair (G_A, G_B), f produces $S^{A\to B,(0)}$ and alternating applications of r and $g^{(\ell)}$ yield refined similarity matrices $S^{A\to B,(1)}, \ldots, S^{A\to B,(L)}$. From any $S^{A\to B,(\ell)}$ we extract a candidate permutation $\pi^{(\ell)}$ via **Proj** or **FAQ** and rank candidates by **nce**$(\pi^{(\ell)})$ from (3). Along the chain, accuracy and **nce** of $\pi^{(\ell)}$ increase monotonically, and the stages correct rather than preserve: of the nodes that $\pi^{(\ell)}$ misassigns and ranks in the bottom half, $\pi^{(\ell+1)}$ reassigns up to 35% correctly, while it keeps 99% or more of the correctly assigned top-half nodes from the third stage on (sparse graphs at noise 0.2, Appendix A.9).

Looping at inference time. At inference, we can apply the final trained network $g^{(L)}$ more than once: each ranking-then-refinement pass produces a new candidate alignment, and we keep the one with the highest **nce** seen so far. We call this **looping**. It introduces no additional training, and because we only keep an iterate when its **nce** improves on the running best, the returned alignment is guaranteed (in **nce**) to be at least as good as the unlooped baseline. The rule only guarantees that looping cannot hurt. The gains come from re-applying a stage that was trained to correct its input. Section 5.1 shows that this gives substantial empirical gains on hard instances.

3.3 GNN architecture and expressiveness

Architecture choice and motivation. We implement all GNN mappings f, $g^{(1)}$, $g^{(2)}, \ldots, g^{(k)}$ using the same architecture, inspired by Folklore-type GNNs [Maron et al., 2019]. Unlike standard message-passing neural networks (MPNNs), this architecture operates on *node pairs* rather than individual nodes, which is what gives it 2-FWL-level expressivity and lets it distinguish nodes in regular graphs that MPNNs cannot.

Core architecture: Folklore-inspired residual layers. Our GNN's main building block is a residual layer that processes hidden states for all node pairs $(h^t_{i\to j})_{i,j} \in \mathbb{R}^{n\times n\times d}$, producing updated states $(h^{t+1}_{i\to j})_{i,j} \in \mathbb{R}^{n\times n\times d}$: $h^{t+1}_{i\to j} = h^t_{i\to j} + m_1\left(h^t_{i\to j}, \sum_\ell h^t_{i\to \ell} \odot m_0(h^t_{\ell\to j})\right)$, where $m_0 : \mathbb{R}^d \to \mathbb{R}^d$ and $m_1 : \mathbb{R}^{2d} \to \mathbb{R}^d$ are multilayer perceptrons (MLPs) with graph normalization layers, and $\odot$ denotes component-wise multiplication. We refer to Section A.6 for more details about our FGNN; in Section A.7 we prove that, despite the single-m_0 simplification, FGNN matches the 2-FWL expressivity of PPGN [Maron et al., 2019] up to a doubling of depth.

4 Related work: state-of-the-art and learning limitations

Additional related work on machine learning approaches to graph matching is discussed in Section A.10. In this section, we restrict our attention to the combinatorial optimization perspective.

Table 2: Accuracy (**acc**) and number of common edges (**nce**) for Erdős-Rényi and regular graphs as a function of the noise p_{noise}. Each cell is **acc** / **nce**. ChFGNN denotes our chained FGNNs; **Proj** and **FAQ** are post-processing decoders. The **nce** metric does not depend on the planted permutation $\pi^\star$ and is therefore unaffected by label imperfection at high noise.

SPARSE ERDŐS-RÉNYI, AVERAGE DEGREE 4 ($\mathbf{NCE}_{\max} \approx 1000$)								
p_{NOISE}	0	0.05	0.1	0.15	0.2	0.25	0.3	0.35
PROJ(D_{cx})	**0.98/997**	**0.97/950**	0.90/853	0.59/499	0.23/195	0.09/130	0.04/115	0.02/112
FAQ(D_{cx})	**0.98/997**	**0.98/950**	**0.96/898**	**0.95/847**	0.73/723	0.13/504	0.04/487	0.02/485
FGNN **PROJ**	**0.98/997**	0.94/**925**	0.74/674	0.44/365	0.23/193	0.12/134	0.06/114	0.03/103
FGNN **FAQ**	**0.98/997**	**0.98/950**	**0.96/898**	**0.95/847**	0.81/755	0.24/535	0.07/494	0.03/485
CHFGNN **PROJ**	**0.98/997**	**0.98/950**	**0.96/898**	**0.94/845**	**0.91/790**	0.82/**720**	0.49/549	0.08/367
CHFGNN **FAQ**	**0.98/997**	**0.98/950**	**0.96/899**	**0.95/849**	**0.93/800**	**0.85/742**	0.52/638	0.09/546
DENSE ERDŐS-RÉNYI, AVERAGE DEGREE 80 ($\mathbf{NCE}_{\max} \approx 20{,}000$)								
p_{NOISE}	0	0.05	0.1	0.15	0.2	0.25	0.3	0.35
PROJ(D_{cx})	**1.00/19964**	**1.00/18987**	**1.00/17966**	0.61/8700	0.14/3888	0.04/3646	0.02/3633	0.01/3624
FAQ(D_{cx})	**1.00/19964**	**1.00/18987**	**1.00/17968**	**1.00/16990**	**1.00/15972**	0.21/7922	0.01/6272	0.01/6276
FGNN **PROJ**	**1.00/19964**	**1.00/18979**	0.73/11254	0.28/4674	0.10/3651	0.04/3521	0.02/3517	0.01/3505
FGNN **FAQ**	**1.00/19964**	**1.00/18987**	**1.00/17968**	**1.00/16990**	0.95/15390	0.14/7031	0.01/6259	0.01/6254
CHFGNN **PROJ**	**1.00/19964**	**1.00/18987**	0.94/16241	0.83/13028	0.68/10291	0.37/6574	0.02/3690	0.01/3591
CHFGNN **FAQ**	**1.00/19964**	**1.00/18987**	**1.00/17968**	**1.00/16990**	**0.99/15972**	**0.62/11577**	0.01/6263	0.01/6255
REGULAR RANDOM GRAPHS, DEGREE 10 ($\mathbf{NCE}_{\max} = 2500$)								
p_{NOISE}	0		0.05		0.1		0.15	0.2
PROJ(D_{cx})	0.002/51		0.002/51		0.003/50		0.001/49	0.002/50
FAQ(D_{cx})	0.002/385		0.003/425		0.003/456		0.002/369	0.003/496
FGNN **PROJ**	**1.00/2500**		0.31/405		0.03/113		0.005/108	0.003/106
FGNN **FAQ**	**1.00/2500**		**0.95/2059**		0.10/912		0.005/837	0.002/838
CHFGNN **PROJ**	**1.00/2500**		**0.95/2034**		0.54/1135		0.009/281	0.003/95
CHFGNN **FAQ**	**1.00/2500**		**0.95/2059**		**0.56/1383**		0.008/871	0.003/836

Non-learning methods. Among traditional optimization approaches, **FAQ** represents the state-of-the-art for correlated random graphs [Lyzinski et al., 2015], outperforming the convex relaxation, GLAG algorithm [Fiori et al., 2013], PATH algorithm [Zaslavskiy et al., 2008], Umeyama's spectral method [Umeyama, 1988], and linear programming approaches [Almohamad and Duffuaa, 1993]. More recent papers [Xu et al., 2019, Bommakanti et al., 2024] propose new algorithms for GAP, but their comparisons with **FAQ** underestimate it, likely because they use a suboptimal initialization (see Section A.2). Optimal-transport relaxations form another classical family: the Gromov–Wasserstein solvers SGWL [Xu et al., 2019], BAPG [Li et al., 2023] and FGWAlign [Tang et al., 2025]. Evaluated on the same test pairs as all other methods (Section A.4), BAPG and FGWAlign match **FAQ**(D_{cx}) on dense graphs, fail earlier on sparse graphs, and are at chance accuracy on regular graphs.

Learning approaches and their limitations. Recent GNN-based methods for graph alignment include SeedGNN [Yu et al., 2023], PGM [Kazemi et al., 2015], MGCN [Chen et al., 2020], and MGNN [Wang et al., 2021]. The closest to our chaining procedure is Deep Graph Matching Consensus [Fey et al., 2020], which also iteratively refines a similarity matrix, but with MPNN backbones and a differentiable consensus rule in the semantic setting where rich node features are available; our setting is purely structural and uses a non-differentiable ranking signal. For Erdős–Rényi graphs in our (purely structural) setting, none of the prior GNN methods reach positive accuracy at noise levels where **FAQ**(D_{cx}) still recovers a meaningful alignment (see Section A.4).

These comparisons motivate **FAQ**(D_{cx}) as our main classical reference point in the purely structural correlated-graph setting. Our goal is not to replace **FAQ** but to provide it, through chaining, with similarity matrices that improve upon both the convex relaxation and prior learning approaches.

5 Empirical results and comparison to FAQ

We evaluate our chaining procedure against **FAQ** Vogelstein et al. [2015], the strongest method on all our datasets (Sections A.3 and A.4 compare with more recent algorithms, including the Gromov–Wasserstein solvers BAPG and FGWAlign). We compare non-neural baselines using the convex relaxation against our chained FGNNs (i.e. $f, g^{(1)}, \ldots, g^{(L)}$); an architecture ablation against

single-step MPNN backbones (SAGE, GIN, GAT) appears in Table 3. All methods can be combined with **Proj** and **FAQ** as a post-processing step to extract a permutation (Section 2.2).

5.1 Main results on synthetic datasets

Table 2 presents results on graphs with 500 nodes across noise levels, for sparse Erdős-Rényi graphs (average degree 4), dense Erdős-Rényi graphs (average degree 80), and regular graphs (degree 10); higher p_{noise} means more corrupted alignments.

Sparse and dense Erdős-Rényi graphs. For both sparse and dense graphs, our chained FGNNs significantly outperform all baselines, particularly at challenging noise levels. At $p_{\text{noise}} = 0.25$, chained FGNNs with **FAQ** post-processing achieve 85% accuracy on sparse graphs, compared to 13% for the non-neural **FAQ** baseline. Note that p_{noise}=0.2 corresponds to the setting of Yu et al. [2023] where none of the GNN-based methods they evaluate achieve positive accuracy (see Section A.4); the architecture ablation in Table 3 confirms that single-step MPNN backbones (SAGE, GIN, GAT) likewise collapse in this regime, while chaining with our FGNN backbone reaches 0.93.

Regular graphs: a particularly challenging case. Regular graphs are a regime where standard approaches break. The convex relaxation (4) admits the barycenter $D_{\text{cx}} = J := \frac{1}{n}\mathbf{1}\mathbf{1}^\top$ as a solution; the Frank–Wolfe gradient of $\langle AD, DB\rangle$ at $D = J$ equals $2d^2J$ on a d-regular pair (since $AJ = JB = dJ$), so the linear-assignment subproblem at the first **FAQ** iteration is degenerate — every permutation matrix achieves the same inner product — and **FAQ**'s trajectory from this initialisation is determined by the LAP solver's tie-breaking rather than by the graph data. MPNNs face a separate expressivity obstruction: bounded by 1-WL [Xu et al., 2018], they collapse to identical node embeddings on a regular graph with constant initial features, even though a random d-regular graph has trivial automorphism group almost surely, so their similarity matrices are uninformative not because of symmetry but because 1-WL fails to refine. Table 2 shows that only our chained FGNN gets a meaningful result (56% accuracy at $p_{\text{noise}} = 0.1$, where every other method essentially fails); both the 2-FWL backbone (expressivity beyond 1-WL) and the chaining (refinement of the noisy alignment) are needed, an empirical gap that we do not formally bound.

Why the gains are larger on sparse graphs. The ranking relies on two sources of contrast, and both fade as the graph densifies. The first is degree heterogeneity: in an Erdős–Rényi graph of average degree d, degrees vary by a relative amount of about $1/\sqrt{d}$, so sparse graphs offer many nodes that stand out for the first stage, while dense graphs approach the regular case where none does. The second is the evidence that an already-matched neighbor pair gives about a candidate pair: it supports the true candidate with probability $\lambda(1 - p_{\text{noise}})$ but a wrong one with probability λ^2, where $\lambda = d/n$, a ratio $(1 - p_{\text{noise}})/\lambda$ that falls as d grows. On dense graphs the transition to failure is therefore sharper and the noise window in which chaining improves on **FAQ**(D_{cx}) is narrower (Appendix A.9).

Ranking ablations. Table 15 (Appendix A.9) isolates what the ranking contributes, on the sparse chain trained at $p_{\text{noise}} = 0.22$ with **FAQ** decoding on 30 fixed test pairs per noise level. *Removing* the rank field reduces the chain to independently trained single FGNNs, so the gap between the ChFGNN and FGNN rows of Table 2 measures the contribution of the ranking. Ranking nodes by $\text{score}(i)/\deg(i)$ instead of $\text{score}(i)$ and retraining the chain gives comparable accuracies at every noise level (0.924 against 0.928 at $p_{\text{noise}} = 0.2$, 0.840 against 0.884 at 0.25), so the normalization is a second-order choice. Applying the normalized ranking at inference to the chain trained with the raw score, without retraining, collapses accuracy at $p_{\text{noise}} \geq 0.2$: each stage relies on the ordering convention it was trained with. A *random order* keeps the transported matching (matched nodes still share a rank value) but destroys the confidence information: at inference it is catastrophic (0.346 at $p_{\text{noise}} = 0.2$), and a chain retrained end-to-end on random orders still improves stage over stage but never catches the score-ordered chain, with a gap that widens with noise (0.802 against 0.928 at 0.2, 0.331 against 0.884 at 0.25). The confidence ordering is thus the component that keeps the chain accurate in the regime where chaining matters. Per-stage measurements (Appendix A.9) show that the stages correct rather than propagate errors: at test noise 0.2, the probability that a bottom-half misassigned node is corrected by the next stage rises from 0.14 to 0.35 as the reliable set grows, while top-half correct matches are retained with probability 0.99–1.00 from the third stage on.

Architecture ablation: chaining versus 2-FWL expressiveness. To disentangle the contribution of the FGNN backbone from the chaining procedure, we re-train the entire pipeline with three standard message-passing GNN backbones in place of FGNN: SAGE, GIN, and GAT. All backbones use the

Table 3: Architecture ablation on sparse Erdős-Rényi (n=500, d=4), **FAQ** decoding. Each cell is **acc** / **nce**. Ch· denotes the chained variant of each backbone. Chaining improves every backbone; only ChFGNN survives at $p_{\text{noise}} \geq 0.20$.

p_{NOISE}	0	0.05	0.1	0.15	0.2	0.25
SAGE-**FAQ**	**0.98/997**	**0.97/946**	0.13/497	0.05/477	0.02/474	0.01/473
CHSAGE-**FAQ**	**0.98/997**	**0.97/951**	0.69/756	0.11/494	0.04/476	0.02/475
GIN-**FAQ**	**0.98/997**	**0.98/951**	0.93/**879**	0.25/538	0.04/478	0.02/477
CHGIN-**FAQ**	**0.98/997**	**0.98/951**	**0.95/889**	0.34/575	0.06/483	0.03/477
GAT-**FAQ**	**0.98/997**	**0.97/951**	0.79/805	0.14/503	0.04/479	0.02/478
CHGAT-**FAQ**	**0.98/997**	**0.97/951**	**0.96/893**	0.92/**833**	0.24/537	0.05/486
CHFGNN-**FAQ**	**0.98/997**	**0.98/950**	**0.96/899**	**0.95/849**	**0.93/800**	**0.85/730**

same hidden width and number of residual blocks. Table 3 reports both metrics on sparse Erdős-Rényi graphs (n=500, d=4) with **FAQ** decoding (**Proj** decoding gives the same ordering). Two findings stand out: *(i)* chaining improves every backbone, e.g. ChGAT→0.92 vs. GAT→0.14 at p_{noise}=0.15; *(ii)* 2-FWL expressiveness is needed in the harder regime — only ChFGNN maintains accuracy at $p_{\text{noise}} \geq 0.20$. On dense Erdős-Rényi and regular graphs we were unable to successfully train any of the MPNN backbones: regular graphs are theoretically beyond MPNN expressivity Xu et al. [2018].

Training strategy: noise-level selection. Figure 8 (appendix) shows that intermediate training noise levels (around $p_{\text{noise}} = 0.22$ for sparse graphs) generalize best: too-easy instances do not generalize to harder cases and too-hard instances are suboptimal on easier ones. The training noise of each graph family is chosen on the validation split only, before evaluating on the test pairs of Table 2; the appendix tables sweep training noise levels without this selection and, unless stated, without the final **nce**-based inference loop.

Looping: enhanced inference without additional training. The last network $g^{(L)}$ can be re-applied at inference for free. Since it was supervised on the output of $g^{(L-1)}$, further applications are formally out of distribution, so we treat looping as an empirically grounded heuristic with a safety net: after each pass we recompute **nce** and keep the candidate only if it improves the running best (capped at N_{loop}=100), so that the returned alignment is at least as good (in **nce**) as the unlooped one. Table 19 (appendix) shows substantial gains on hard instances (up to +0.26 accuracy at p_{noise}=0.30, L+1=16) at negligible compute overhead; the number of loops scales with difficulty (avg. 15 at p_{noise}=0.15 vs. 91 at 0.30).

5.2 Computational cost and scalability

A fair hardware-independent comparison of running times between **FAQ**(D_{cx}) and our chained GNN procedure is challenging, so we focus on inference complexity and iteration counts rather than claiming algorithmic speedups from wall-clock time. While our method requires an initial GPU-based training phase, this is assumed to be completed before solving new instances. For **FAQ**(D_{cx}), each gradient step involves solving a linear assignment problem ($O(n^3)$), and total runtime depends on the number of gradient ascent iterations. Our chaining procedure has two main costs as n grows: (i) the bilinear pair aggregation in the graph layer, with arithmetic cost $O(n^3 d)$ and activation storage $O(n^2 d)$ per layer when implemented as matrix products/tensor contractions; and (ii) computing ranks via a projection **Proj** of the similarity matrix in each iteration, an $O(n^3)$ CPU operation. Table 14 (appendix) reports the average number of gradient ascent iterations in **FAQ**, starting from either D_{cx} or the similarity matrix produced by our chained FGNN. The iteration count is substantially lower with the chained FGNN, indicating that the similarity matrix from chaining provides a more accurate initialization than D_{cx}. A wall-clock time–performance trade-off against **FAQ**(J), **FAQ**(D_{cx}) and FUGAL (Pareto curves) is reported in Appendix A.5. **Memory is the binding constraint.** The order-2 pair state (0.26 GB per stored activation at $n = 500$, $d = 256$, single precision) grows quadratically with n. The released implementation runs end-to-end on the largest real graphs of this paper ($n = 1004$ and $n = 1174$), but this dense implementation, does not exploit graph sparsity, preventing our method to scale to graphs orders of magnitude larger.

Table 4: Accuracy (**acc**) and number of common edges (**nce**) on noisy versions of real-world networks; each cell is **acc** / **nce**. ChFGNN-ER4 is the model trained on sparse Erdős–Rényi graphs (d=4); ChFGNN is trained on the target network and noise level.

METHOD	YEAST25LC		CA-NETSCIENCE		INF-EUROROAD	
NOISE q	5%	10%	10%	20%	10%	20%
FAQ(D_{cx})	0.50/7660	0.45/7245	0.65/822	0.46/687	0.56/1170	0.11/940
CHFGNN-ER4	0.48/7693	0.42/7297	0.64/818	0.44/688	0.40/1111	0.08/**970**
CHFGNN	0.54/7732	0.51/**7404**	0.65/824	0.57/**724**	0.64/**1213**	0.15/**963**

5.3 Results on real graphs.

We evaluate on three real-world datasets (biology, social networks and road networks; details in Section A.3). **Yeast** [Vijayan and Milenkovic, 2018] is a protein–protein interaction (PPI) network (1,004 proteins, 8,323 trusted interactions) with five noisy variants obtained by adding $q\%$ low-confidence edges ($q \in \{5, 10, 15, 20, 25\}$). Since the base graph is always an induced subgraph of each variant, the maximum number of common edges is $8{,}323$ and the true node correspondence is known. In Table 8 (appendix), all methods recover nearly the maximum number of common edges (within 3%), but high **nce** does not always translate into high node accuracy because the base graph has a large automorphism group; **nce** is thus the more reliable metric, and **FAQ** is already near-optimal on this benchmark. For harder benchmarks, we apply the edge-addition–removal noise model (Section 2.3) to the yeast PPI with $q = 25\%$, the **ca-netscience** coauthorship network [Newman, 2006], and the **inf-euroroad** road network [Šubelj and Bajec, 2011], evaluating both zero-shot transferred and dataset-specific FGNNs. Table 4 shows that dataset-specific ChFGNNs achieve the best **nce** in most high-noise settings, improving over **FAQ**(D_{cx}) by about 2%. The transferred ChFGNN-ER4 model is more mixed: it sometimes improves **nce** but underperforms **FAQ**(D_{cx}) in node accuracy, so results for zero-shot transfer are limited. The smaller gains have two causes. Distribution shift: the real graphs are far from the synthetic training model in degree spread and clustering (Table 7, appendix), so transfer is reliable only when the target resembles the training distribution, as for ca-netscience at 10% noise where the transferred model is within one point of the dataset-specific one, and dataset-specific training, which only requires simulating the noise model on the target graph without external labels, is otherwise needed. Evaluation ambiguity: on ca-netscience at 10% noise the chain reaches **nce** $= 824$ against 826 for **FAQ** started from the true alignment (Table 9), yet its node accuracy is 0.65. The accuracy is limited by symmetries of the graph: 48% of the nodes of ca-netscience have an exact twin, a node with the same neighbors (typically a coauthor of the same papers), and exchanging twins changes no **nce**, so the expected accuracy of any method that only sees the structure is at most 0.71 on this graph. Exact twins alone bound the accuracy by 0.85 on yeast and 0.98 on inf-euroroad; there the limiting factor is the near-twins, nodes that can be exchanged with another node by changing at most two edges of the graph, which the noise makes indistinguishable in practice: 57% of the nodes of yeast and 77% of those of inf-euroroad are in this case (Table 7). We therefore report both metrics and do not read much into accuracy differences of a few points.

6 Conclusion and limitations

We introduced a chained-GNN procedure for combinatorial graph alignment that learns initializations for **FAQ** and improves over **FAQ**(D_{cx}) on hard sparse correlated Erdős–Rényi instances and on the regular-graph regime among the methods we evaluate, including Gromov–Wasserstein solvers; its ablations identify the confidence ordering of the ranking as the component that makes the chain reach useful accuracy in a feasible number of stages. It extends naturally to the seeded GAP. **Limitations:** each FGNN layer has $O(n^2 d)$ activation memory, capping training around $n \approx 1000$ on an 80 GB GPU in our implementation, and the dense pair state cannot exploit graph sparsity; real-world gains over a properly initialized **FAQ** are consistent across all six settings but smaller than on synthetic graphs, and they require training on the target graph rather than zero-shot transfer; FGNN inherits the 2-FWL ceiling [Maron et al., 2019] and cannot separate strongly-regular graphs of identical parameters, a barrier we have not hit empirically.

Acknowledgments and Disclosure of Funding

We thank the anonymous reviewers and the area chair, whose questions on the mechanism of the ranking feedback, on robustness and on baselines led to the analyses of Section 5.1 and Appendix A.9. We are grateful to the CLEPS infrastructure from Inria Paris for providing resources and support. This work was granted access to the HPC resources of IDRIS under the allocation 2025-AD010613995R2 and A0191016874 made by GENCI, on the A100 and H100 partitions of the Jean Zay supercomputer.

References

Ralph Abboud, İsmail İlkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2112–2118, 2021.

HA Almohamad and Salih O Duffuaa. A linear programming approach for the weighted graph matching problem. *IEEE Transactions on pattern analysis and machine intelligence*, 15(5): 522–525, 1993.

Waiss Azizian and Marc Lelarge. Expressive power of invariant and equivariant graph neural networks. In *International Conference on Learning Representations*, 2021.

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

Aditya Bommakanti, Harshith R Vonteri, Konstantinos Skitsas, Sayan Ranu, Davide Mottin, and Panagiotis Karras. Fugal: Feature-fortified unrestricted graph alignment. *Advances in Neural Information Processing Systems*, 37:19523–19546, 2024.

Rainer E Burkard, Stefan E Karisch, and Franz Rendl. Qaplib–a quadratic assignment problem library. *Journal of Global optimization*, 10:391–403, 1997.

Rainer E. Burkard, Eranda Çela, Panos M. Pardalos, and Leonidas S. Pitsoulis. *The Quadratic Assignment Problem*, pages 1713–1809. Springer US, Boston, MA, 1998. ISBN 978-1-4613-0303-9. doi: 10.1007/978-1-4613-0303-9_27. URL https://doi.org/10.1007/978-1-4613-0303-9_27.

Tianle Cai, Shengjie Luo, Keyulu Xu, Di He, Tie-yan Liu, and Liwei Wang. Graphnorm: A principled approach to accelerating graph neural network training. In *International Conference on Machine Learning*, pages 1204–1215. PMLR, 2021.

Hongxu Chen, Hongzhi Yin, Xiangguo Sun, Tong Chen, Bogdan Gabrys, and Katarzyna Musial. Multi-level graph convolutional networks for cross-platform anchor link prediction. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1503–1511, 2020.

D Conte, P Foggia, C Sansone, and M Vento. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition & Artificial Intelligence*, 18(3), 2004.

Pierluigi Crescenzi, Viggo Kann, and M Halldórsson. A compendium of np optimization problems. Technical report and online compendium, 1995.

Daniel Cullina and Negar Kiyavash. Improved achievability and converse bounds for erdos-renyi graph matching. *SIGMETRICS Perform. Eval. Rev.*, 44(1):63–72, jun 2016. ISSN 0163-5999. doi: 10.1145/2964791.2901460. URL https://doi.org/10.1145/2964791.2901460.

Jian Ding and Hang Du. Matching recovery threshold for correlated random graphs. *The Annals of Statistics*, 51(4):1718–1743, 2023.

Jian Ding and Zhangsong Li. A polynomial-time iterative algorithm for random graph matching with non-vanishing correlation. *arXiv preprint arXiv:2306.00266*, 2023.

Jian Ding, Zongming Ma, Yihong Wu, and Jiaming Xu. Efficient random graph matching via degree profiles. *Probability Theory and Related Fields*, 179:29–115, 2021.

Zhou Fan, Cheng Mao, Yihong Wu, and Jiaming Xu. Spectral graph matching and regularized quadratic relaxations ii: Erdős-rényi graphs and universality. *Foundations of Computational Mathematics*, 23(5):1567–1617, 2023.

Matthias Fey, Jan E Lenssen, Christopher Morris, Jonathan Masci, and Nils M Kriege. Deep graph matching consensus. *ICLR*, 2020.

Marcelo Fiori, Pablo Sprechmann, Joshua Vogelstein, Pablo Musé, and Guillermo Sapiro. Robust multimodal graph matching: Sparse coding meets graph matching. *Advances in Neural Information Processing Systems*, 26, 2013.

Donniell E Fishkind, Sancar Adali, Heather G Patsolic, Lingyao Meng, Digvijay Singh, Vince Lyzinski, and Carey E Priebe. Seeded graph matching. *Pattern recognition*, 87:203–215, 2019.

Rémi Flamary, Nicolas Courty, Alexandre Gramfort, Mokhtar Z. Alaya, Aurélie Boisbunon, Stanislas Chambon, Laetitia Chapel, Adrien Corenflos, Kilian Fatras, Nemo Fournier, Léo Gautheron, Nathalie T.H. Gayraud, Hicham Janati, Alain Rakotomamonjy, Ievgen Redko, Antoine Rolet, Antony Schutz, Vivien Seguy, Danica J. Sutherland, Romain Tavenard, Alexander Tong, and Titouan Vayer. POT: Python optimal transport. *Journal of Machine Learning Research*, 22(78): 1–8, 2021.

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

Luca Ganassali, Marc Lelarge, and Laurent Massoulié. Correlation detection in trees for planted graph alignment. *The Annals of Applied Probability*, 34(3):2799–2843, 2024a.

Luca Ganassali, Laurent Massoulié, and Guilhem Semerjian. Statistical limits of correlation detection in trees. *The Annals of Applied Probability*, 34(4):3701–3734, 2024b.

Quankai Gao, Fudong Wang, Nan Xue, Jin-Gang Yu, and Gui-Song Xia. Deep graph matching under quadratic constraint. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5069–5078, 2021.

Bo Jiang, Pengfei Sun, and Bin Luo. Glmnet: Graph learning-matching convolutional networks for feature matching. *Pattern Recognition*, 121:108167, 2022.

Ehsan Kazemi, S Hamed Hassani, and Matthias Grossglauser. Growing a graph matching from a handful of seeds. *Proceedings of the VLDB Endowment*, 8(10):1010–1021, 2015.

H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. doi: https://doi.org/10.1002/nav.3800020109. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800020109`.

Adrien Lagesse and Marc Lelarge. Graph alignment for benchmarking graph neural networks and learning positional encodings. *arXiv preprint arXiv:2505.13087*, 2025.

Jiajin Li, Jianheng Tang, Lemin Kong, Huikang Liu, Jia Li, Anthony Man-Cho So, and Jose Blanchet. A convergent single-loop algorithm for relaxation of Gromov-Wasserstein in graph data. In *International Conference on Learning Representations*, 2023.

Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. In *International conference on machine learning*, pages 3835–3845. PMLR, 2019.

Joseph Lubars and R. Srikant. Correcting the output of approximate graph matching algorithms. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 1745–1753, 2018.

Vince Lyzinski, Donniell E Fishkind, Marcelo Fiori, Joshua T Vogelstein, Carey E Priebe, and Guillermo Sapiro. Graph matching: Relax at your own risk. *IEEE transactions on pattern analysis and machine intelligence*, 38(1):60–73, 2015.

Cheng Mao, Yihong Wu, Jiaming Xu, and Sophie 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.

Haggai Maron and Yaron Lipman. (probably) concave graph matching. *Advances in Neural Information Processing Systems*, 31, 2018.

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. *Advances in neural information processing systems*, 32, 2019.

Facundo Mémoli. Gromov–wasserstein distances and the metric approach to object matching. *Foundations of computational mathematics*, 11:417–487, 2011.

Elchanan Mossel and Jiaming Xu. Seeded graph matching via large neighborhood statistics. *Random Structures & Algorithms*, 57(3):570–611, 2020.

Andrea Muratori and Guilhem Semerjian. Faster algorithms for the alignment of sparse correlated Erdős–Rényi random graphs. *arXiv preprint arXiv:2405.08421*, 2024.

Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 111–125. IEEE, 2008.

M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Phys. Rev. E*, 74:036104, Sep 2006. doi: 10.1103/PhysRevE.74.036104. URL `https://link.aps.org/doi/10.1103/PhysRevE.74.036104`.

Alex Nowak, Soledad Villar, Afonso S Bandeira, and Joan Bruna. Revised note on learning quadratic assignment with graph neural networks. In *2018 IEEE Data Science Workshop (DSW)*, pages 1–5. IEEE, 2018.

Panos M Pardalos and Stephen A Vavasis. Quadratic programming with one negative eigenvalue is np-hard. *Journal of Global optimization*, 1(1):15–22, 1991.

Giovanni Piccioli, Guilhem Semerjian, Gabriele Sicuro, and Lenka Zdeborová. Aligning random graphs with a sub-tree similarity message-passing algorithm. *Journal of Statistical Mechanics: Theory and Experiment*, 2022(6):063401, 2022.

Michal Rolínek, Paul Swoboda, Dominik Zietlow, Anselm Paulus, Vít Musil, and Georg Martius. Deep graph matching via blackbox differentiation of combinatorial solvers. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVIII 16*, pages 407–424. Springer, 2020.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 627–635, 2011.

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Random features strengthen graph neural networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pages 333–341, 2021.

Jadranka Skorin-Kapov. Tabu search applied to the quadratic assignment problem. *ORSA Journal on computing*, 2(1):33–45, 1990.

Lovro Šubelj and Marko Bajec. Robust network community detection using balanced propagation. *The European Physical Journal B*, 81(3):353–362, 2011.

Hui Sun, Wenju Zhou, and Minrui Fei. A survey on graph matching in computer vision. In *2020 13th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pages 225–230. IEEE, 2020.

Éric Taillard. Robust taboo search for the quadratic assignment problem. *Parallel computing*, 17 (4-5):443–455, 1991.

Jianheng Tang, Xi Zhao, Lemin Kong, Xiaofang Zhou, and Jia Li. Fused Gromov-Wasserstein alignment for graph edit distance computation and beyond. *Proceedings of the VLDB Endowment*, 18(10):3641–3654, 2025.

G. Tinhofer. A note on compact graphs. *Discrete Applied Mathematics*, 30(2):253–264, 1991. ISSN 0166-218X. doi: https://doi.org/10.1016/0166-218X(91)90049-3. URL `https://www.sciencedirect.com/science/article/pii/0166218X91900493`.

Shinji Umeyama. An eigendecomposition approach to weighted graph matching problems. *IEEE transactions on pattern analysis and machine intelligence*, 10(5):695–703, 1988.

Vipin Vijayan and Tijana Milenkovic. Multiple network alignment via multimagna++. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 15(5):1669–1682, September 2018. ISSN 1545-5963. doi: 10.1109/TCBB.2017.2740381. URL `https://doi.org/10.1109/TCBB.2017.2740381`.

Joshua T Vogelstein, John M Conroy, Vince Lyzinski, Louis J Podrazik, Steven G Kratzer, Eric T Harley, Donniell E Fishkind, R Jacob Vogelstein, and Carey E Priebe. Fast approximate quadratic programming for graph matching. *PLOS one*, 10(4):e0121002, 2015.

Runzhong Wang, Junchi Yan, and Xiaokang Yang. Neural graph matching network: Learning lawler's quadratic assignment problem with extension to hypergraph and multiple-graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9):5261–5279, 2021.

Hongteng Xu, Dixin Luo, and Lawrence Carin. Scalable gromov-wasserstein learning for graph partitioning and matching. *Advances in neural information processing systems*, 32, 2019.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2018.

Liren Yu, Jiaming Xu, and Xiaojun Lin. Graph matching with partially-correct seeds. *Journal of Machine Learning Research*, 22(280):1–54, 2021a.

Liren Yu, Jiaming Xu, and Xiaojun Lin. Seedgnn: graph neural network for supervised seeded graph matching. In *International Conference on Machine Learning*, pages 40390–40411. PMLR, 2023.

Tianshu Yu, Runzhong Wang, Junchi Yan, and Baoxin Li. Deep latent graph matching. In *International Conference on Machine Learning*, pages 12187–12197. PMLR, 2021b.

Andrei Zanfir and Cristian Sminchisescu. Deep learning of graph matching. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2684–2693, 2018.

Mikhail Zaslavskiy, Francis Bach, and Jean-Philippe Vert. A path following algorithm for the graph matching problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12): 2227–2242, 2008.

A Appendix

A.1 Correlated random graphs

In this section, we present the mathematical details for the various correlated random graphs model used in this paper.

Bernoulli graphs. We start with the model considered in [Lyzinski et al., 2015]. Given n the number of nodes, $\rho \in [0,1]$ and a symmetric hollow matrix $\Lambda \in [0,1]^{n\times n}$, define $\mathcal{E} = \{\{i,j\}, i \in [n], j \in [n], i \neq j\}$. Two random graphs $G_A = (V_A, E_A)$ and $G_B = (V_B, E_B)$ are ρ-correlated Bernoulli(Λ) distributed, if for all $\{i,j\} \in \mathcal{E}$, the random variables (matrix entries) A_{ij} and B_{ij} are such that $B_{ij} \sim$ Bernoulli(Λ_{ij}) independently drawn and then conditioning on B, we have $A_{ij} \sim$ Bernoulli($\rho B_{ij} + (1-\rho)\Lambda_{ij}$) independently drawn. Note that the marginal distributions of A and B are both Bernoulli(Λ), so the laws of A and B are the same (but correlated).

In our experiments in Sections A.14.1 and A.14.2, we consider the same case as in [Lyzinski et al., 2015]: $n = 150$ vertices, the entries of the matrix Λ are i.i.d. uniform in $[\alpha, 1-\alpha]$ with $\alpha = 0.1$, and we vary ρ.

Erdős-Rényi graphs. The Erdős-Rényi model is a special case of the Bernoulli model where Λ is the matrix with all entries equal to λ. To be consistent with the main notation, we define $p_{\text{noise}} = (1-\lambda)(1-\rho)$ where ρ was the correlation above and $\lambda = d/n$ where d is the average degree of the graph. Hence the random graphs G_A and G_B are correlated Erdős-Rényi graphs when $\mathbb{P}(A_{i,j} = B_{i,j} = 1) = \frac{d}{n}(1-p_{\text{noise}})$ and $\mathbb{P}(A_{i,j} = 0,\, B_{i,j} = 1) = \mathbb{P}(A_{i,j} = 1,\, B_{i,j} = 0) = \frac{d}{n}p_{\text{noise}}$.

Regular graphs. In this case, we first generate G_A as a uniform regular graph with degree d and then we generate G_B by applying edgeswap to G_A: if $\{i,j\}$ and $\{k,\ell\}$ are two edges of G_A then we swap them to $\{i,\ell\}$ and $\{k,j\}$ with probability p_{noise}.

A.2 Limitations in Reported FAQ Performance in Prior Work

Several recent studies on graph alignment do not provide a fair or representative comparison against the Fast Approximate Quadratic Assignment (**FAQ**) algorithm [Vogelstein et al., 2015]. In particular, **FAQ**'s performance is frequently underestimated due to the use of suboptimal initialization strategies. When properly initialized, **FAQ** remains a remarkably strong and competitive baseline.

The original **FAQ** implementation is provided in MATLAB in the authors' repository[2] and dates back nearly a decade. A more recent and widely used implementation is available in SciPy via `scipy.optimize.quadratic_assignment(method='faq')`. While convenient, the SciPy implementation does not expose the theoretically motivated initialization scheme recommended in the literature.

[2] `https://github.com/jovo/FastApproximateQAP/`

Table 5: Statistics of synthetic datasets.

name	average degree	number of nodes	used for comparison with	sizes of train/val
Bernoulli	70	150	**FAQ**(D_{cx}) [Lyzinski et al., 2015]	2000/200
Sparse Erdős-Rényi (ER 4)	4	500	MPNN [Yu et al., 2023]	200/100
Dense Erdős-Rényi (ER 80)	80	500	MPNN [Yu et al., 2023]	200/100
Large Erdős-Rényi	3	1000	Bayesian message passing [Muratori and Semerjian, 2024]	200/100
Regular	10	500	new	200/100

Table 6: Summary statistics of the real-world graphs.

Dataset	# Nodes	# Edges	Avg. Degree	size train/valid
Yeast PPI [Vijayan and Milenkovic, 2018]	1,004	8,323	16.58	20/20
ca-netscience[Newman, 2006]	379	914	4.82	200/20
inf-euroroad [Šubelj and Bajec, 2011]	1,174	1,417	2.41	20/5

Table 7: Structure and symmetries of the real-world graphs. Degree spread is the standard deviation of the degrees divided by their mean, and clustering the average clustering coefficient; both are compared with an Erdős–Rényi graph of the same size and average degree. A node has a *twin* when another node has the same neighbors (or, for two adjacent nodes, the same other neighbors): exchanging twins is an automorphism of the graph, so it leaves **nce** unchanged for every alignment, and the accuracy ceiling is the expected accuracy of a method that cannot tell twins apart (one correct node per twin class). The last column counts the nodes that can be exchanged with another node at the cost of at most two edges of the graph.

Dataset	Degree spread graph / ER	Clustering graph / ER	Twins	Acc. ceiling	Swap ≤ 2 edges
Yeast PPI	1.12 / 0.24	0.65 / 0.02	24%	0.85	57%
ca-netscience	0.81 / 0.45	0.74 / 0.01	48%	0.71	83%
inf-euroroad	0.49 / 0.64	0.02 / 0.002	4%	0.98	77%

Specifically, [Lyzinski et al., 2015] shows that initializing the indefinite relaxation (i.e., **FAQ**) with the solution of the convex relaxation D_{cx}, leads to improved convergence and empirical performance. Their theoretical analysis and experiments both support this strategy.

The corresponding MATLAB implementation computes this initialization using a Frank–Wolfe procedure (see `relaxed_normAPPB_FW_seeds.m` in the original repository). In contrast, SciPy's default behavior initializes FAQ with the uniform doubly stochastic matrix, which is known to be inferior to the convex-optimum initialization.

To ensure a fair evaluation, we therefore explicitly compute the convex relaxation using our own Frank–Wolfe solver and pass the resulting matrix D_{cx} as the initialization for **FAQ**. This matches the procedure advocated in [Lyzinski et al., 2015] and the original MATLAB implementation.

Below, we empirically assessed the impact of initialization on three real-world datasets (biology, social networks, and transportation) previously used in [Bommakanti et al., 2024] to introduce the algorithm FUGAL. Across all datasets, we observe that:

1. Our **FAQ** results differ substantially from those reported in [Bommakanti et al., 2024];
2. Properly initialized **FAQ** consistently outperforms FUGAL and all competing methods reported in Bommakanti et al. [2024]; and
3. Our chained FGNN models further outperform this strengthened **FAQ** baseline.

We also re-evaluated FAQ on the synthetic benchmarks used in SeedGNN [Yu et al., 2023]. Although SeedGNN compares against PGM [Kazemi et al., 2015], SGM [Fishkind et al., 2019], and MGCN [Chen et al., 2020], it does not include **FAQ**. Using the convex-optimum initialization, we find that **FAQ** outperforms SeedGNN and all of its reported competitors, while our chained FGNNs again achieve the best overall performance.

Taken together, these results indicate that prior studies likely underestimate **FAQ** due to inadequate initialization, and that properly configured **FAQ** should be regarded as a strong and necessary baseline for graph alignment research.

A.3 Recent results on real graphs

The real-world networks in Section 5.3 are standard benchmarks for graph alignment. We apply the same noising procedure as for Erdős–Rényi graphs (described above), using the original graph's

average degree d. Table 6 gives the sizes of the training and validation sets used for training our chained FGNNs.

Table 8 reports results for SGWL [Xu et al., 2019] and FUGAL [Bommakanti et al., 2024]. SGWL results come from the original paper; FUGAL results were obtained using the authors' code (available at `https://github.com/idea-iitd/Fugal`). Our FUGAL performance matches [Bommakanti et al., 2024], but our **FAQ** results do not. We attribute the discrepancy to poor initialization: using the uninformative barycenter $J = \frac{1}{n}\mathbf{1}\mathbf{1}^\top$ reproduces the degraded **FAQ** performance reported in [Bommakanti et al., 2024].

On noisy real datasets (Table 9), FUGAL never matches **FAQ**, contrary to the claims in [Bommakanti et al., 2024]. To compute the maximum number of common edges, we run **FAQ** initialized with the true permutation (prior to noising).

A.4 More recent results on synthetic graphs

Because FUGAL [Bommakanti et al., 2024] reports stronger performance than SGWL [Xu et al., 2019], we evaluated FUGAL on our synthetic benchmarks using the authors' public implementation (`https://github.com/idea-iitd/Fugal`). We tested multiple hyperparameter configurations to ensure a fair comparison. Nevertheless, across all settings we considered, FUGAL consistently underperformed both **FAQ** (with convex-optimum initialization D_{cx}) and our chained FGNN (ChFGNN) models.

Table 10 reports the number of correctly matched common edges (**nce**). Properly initialized **FAQ** substantially outperforms FUGAL in all regimes, while ChFGNN achieves the best overall performance.

Gromov–Wasserstein solvers: BAPG and FGWAlign. We also evaluate two optimal-transport baselines: BAPG [Li et al., 2023], a single-loop Bregman alternating projected gradient method for the Gromov–Wasserstein relaxation, and FGWAlign [Tang et al., 2025], a fused Gromov–Wasserstein solver designed for graph edit distance. For equal-size unlabeled graphs, the graph edit distance minimized by FGWAlign equals $|E_A| + |E_B| - 2\,\mathbf{nce}(\pi)$ over permutations, so it directly optimizes our metric. BAPG follows the protocol of its authors through its implementation in the POT library [Flamary et al., 2021]: raw adjacency matrices, uniform marginals, step size $\rho = 0.1$, 2000 iterations, tolerance 10^{-6}, square loss; we decode its transport plan with the Hungarian algorithm, a stronger extraction than the row-wise argmax used in the original paper. FGWAlign uses the authors' code with their default full protocol (patience 15, top-k 5) in its sparse mode, since the dense code path overflows single precision at $n = 500$; the two modes have the same optimum over permutations, since every edge or non-edge mismatch count is an affine function of $\mathbf{nce}(\pi)$ once the two edge counts are fixed. Both solvers are run on exactly the same test pairs as every other method (30 pairs per cell for sparse and regular graphs, 10 for dense), so that the comparison is paired per graph pair; the per-pair results are released with the code.

Table 11 reports accuracy and **nce** for all methods on these pairs, reproduced with the released code. The means differ slightly from Table 2, which used 100 pairs per cell from another dataset. BAPG and FGWAlign behave as one: calling a pair solved when accuracy exceeds 0.5, the two

Table 8: Accuracy (**acc**) and number of common edges (**nce**) on the Yeast PPI networks. ChFGNN ER4 is our model trained on Erdős-Rényi graphs with average degree 4, while ChFGNN is trained on the pairs obtained with the three first networks.

YEAST PPI NETWORKS (ACC / NCE)

METHOD	5% CONF	10% CONF	15% CONF	20% CONF	25% CONF
FAQ(J)	37.5 / 7383	34.4 / 7245	29.1 / 6807	23.9 / 6689	36.4 / 7383
SGWL	83.6 / –	– / –	66.6 / –	– / –	58.8 / –
FUGAL	83.0 / 8311	77.7 / 8231	74.3 / 8172	70.9 / 8148	68.6 / 8095
FAQ(D_{cx})	84.2 / 8323	82.6 / 8317	78.0 / 8289	77.0 / 8294	76.1 / 8306
CHFGNN ER4	80.3 / 8300	75.3 / 8288	67.2 / 8252	63.1 / 8213	53.1 / 8080
CHFGNN	TRAINING	TRAINING	TRAINING	72.2 / 8300	69.8 / 8291

Table 9: Accuracy (**acc**) and number of common edges (**nce**) on noisy versions of real-world networks. Each network is corrupted by adding noise at different levels. ChFGNN ER4 is trained on Erdős-Rényi graphs, while ChFGNN is trained on the specific network and noise level. In bold if gain in **nce** is larger than 2%.

REAL-WORLD NETWORKS WITH ADDED NOISE (ACC / NCE)

METHOD	YEAST25LC		CA-NETSCIENCE		INF-EUROROAD	
	5%	10%	10%	20%	10%	20%
FUGAL	53.1 / 7480	44.6 / 7035	60.3 / 794	37.7 / 629	18.3 / 818	2.9 / 714
FAQ(D_{cx})	49.8 / 7660	44.7 / 7245	65.2 / 822	45.6 / 687	55.8 / 1170	10.9 / 940
CHFGNN ER4	47.6 / 7693	42.3 / 7297	63.5 / 818	44.1 / 688	40.0 / 1111	7.5 / **970**
CHFGNN	54.1 / 7732	51.3 / **7404**	65.4 / 824	57.0 / **724**	63.5 / **1213**	15.4 / **963**
MAX **NCE**	7909	7498	826	730	1272	1137

Table 10: Number of common edges (NCE) recovered on synthetic graphs.

Datasets	**FUGAL**	**FAQ**(D_{cx})	**ChFGNN**
ER sparse (noise 0.1)	560	898	899
ER sparse (noise 0.2)	396	723	800
ER dense (noise 0.1)	17968	17968	17968
ER dense (noise 0.2)	5853	15972	15876
Regular (noise 0.05)	131	425	2059

solvers solve or fail the same pairs on 469 of the 470 synthetic pairs (240 sparse, 80 dense, 150 regular). On dense graphs they coincide with **FAQ**(D_{cx}) and collapse at the same noise level; on sparse graphs they degrade earlier than **FAQ**(D_{cx}); on regular graphs both are at chance accuracy, like every relaxation-based method we tested. FGWAlign does salvage more common edges on regular graphs (**nce** ≈ 820 of 2500, against ≈ 50 for BAPG and **Proj**(D_{cx})) without recovering node identities. Across all 470 pairs, ChFGNN **FAQ** solves 192 pairs that BAPG does not and misses a single pair that BAPG solves (dense, $p_{\text{noise}} = 0.3$); Figure 3 shows the paired, per-pair view, together with the bimodality of all methods at their transition.

The problem of graph alignment for correlated Erdős Rényi random graphs has been studied empirically with Message Passing GNN (MPNN) in [Yu et al., 2023] when a seed of matched vertices is given in addition to the 2 graphs. We are reproducing their results taken from `https://github.com/Leron33/SeedGNN` corresponding to Figure 6 in [Yu et al., 2023]. SeedGNN refers to [Yu et al., 2023], PGM to [Kazemi et al., 2015], SGM to [Fishkind et al., 2019] and MGCN to [Chen et al., 2020].

At 0% seeds (the purely structural setting studied in this paper) and noise 0.2, none of the four seeded methods in Table 12 exceeds 0.4% accuracy on either sparse or dense graphs. By contrast, on the same regime, **FAQ**(D_{cx}) from Section 5 reaches 73% (sparse) and 100% (dense), and our chained FGNNs reach 93% and 99% respectively.

A.5 Time–performance trade-off

For iterative algorithms, the computation time can be controlled by adjusting the number of iterations. This applies to gradient-based methods such as the Frank–Wolfe algorithm, used to compute either the convex relaxation D_{cx} or the indefinite relaxation **FAQ**. The recently proposed FUGAL algorithm [Bommakanti et al., 2024] is also iterative. Finally, our chained FGNNs naturally define an iterative procedure, where we may bound the number of chaining steps. Figure 4 displays the resulting Pareto curves, showing the trade-off between performance and runtime for each method on sparse Erdős–Rényi graphs with noise level $p_{\text{noise}} = 0.2$.

For FUGAL, we used the authors' implementation (available at `https://github.com/idea-iitd/Fugal`), specifically the `predict_alignment` routine with hy-

Table 11: Accuracy (**acc**) and number of common edges (**nce**) of the Gromov–Wasserstein solvers BAPG and FGWAlign, compared with the convex-relaxation baselines and our chained FGNNs on identical seeded test pairs ($n = 500$). Each cell is **acc** / **nce**; best accuracy per column in bold.

SPARSE ERDŐS-RÉNYI, AVERAGE DEGREE 4 (30 PAIRS PER CELL)								
p_{NOISE}	0	0.05	0.1	0.15	0.2	0.25	0.3	0.35
PROJ(D_{cx})	**0.98/994**	**0.98/944**	0.91/849	0.58/482	0.22/195	0.088/131	0.040/117	0.020/117
FAQ(D_{cx})	**0.98/994**	**0.97/945**	**0.97/895**	0.94/841	0.64/683	0.12/499	0.037/484	0.015/481
BAPG	**0.98/994**	0.89/893	0.71/772	0.35/624	0.13/548	0.062/528	0.020/520	0.009/520
FGWALIGN	**0.98/994**	0.88/886	0.70/767	0.34/618	0.13/546	0.062/528	0.021/520	0.010/520
CHFGNN **PROJ**	**0.98/994**	**0.98/945**	**0.97/894**	**0.95/840**	0.91/783	0.85/722	0.40/451	0.035/261
CHFGNN **FAQ**	**0.98/994**	**0.98/945**	**0.97/895**	**0.95/843**	**0.93/792**	**0.88/744**	**0.44/608**	0.033/520
DENSE ERDŐS-RÉNYI, AVERAGE DEGREE 80 (10 PAIRS PER CELL)								
p_{NOISE}	0	0.05	0.1	0.15	0.2	0.25	0.3	0.35
PROJ(D_{cx})	**1.00/19906**	**1.00/18904**	**1.00/17882**	0.67/9827	0.17/3975	0.049/3664	0.024/3646	0.012/3604
FAQ(D_{cx})	**1.00/19906**	**1.00/18904**	**1.00/17896**	**1.00/16912**	**1.00/15899**	0.32/8854	0.014/6226	0.006/6218
BAPG	**1.00/19906**	**1.00/18904**	**1.00/17896**	**1.00/16912**	**1.00/15899**	0.32/8832	0.13/7018	0.011/6208
FGWALIGN	**1.00/19906**	**1.00/18904**	**1.00/17869**	**1.00/16842**	**1.00/15784**	0.32/8595	0.13/6981	0.014/6208
CHFGNN **PROJ**	**1.00/19906**	**1.00/18904**	0.95/16584	0.82/12699	0.73/10333	0.49/7026	0.082/4278	0.014/4037
CHFGNN **FAQ**	**1.00/19906**	**1.00/18904**	**1.00/17896**	**1.00/16912**	**1.00/15899**	**0.90/14050**	**0.22/7800**	0.009/6272
REGULAR RANDOM GRAPHS, DEGREE 10 (30 PAIRS PER CELL)								
p_{NOISE}	0	0.05	0.1	0.15	0.2			
PROJ(D_{cx})	0.002/51	0.002/51	0.002/51	0.002/50	0.002/50			
FAQ(D_{cx})	0.002/623	0.002/491	0.002/571	0.002/646	0.002/409			
BAPG	0.002/51	0.002/51	0.002/51	0.002/50	0.002/50			
FGWALIGN	0.003/821	0.002/820	0.003/821	0.002/821	0.002/821			
CHFGNN **PROJ**	**1.00/2500**	0.72/1329	0.27/518	0.005/296	0.004/290			
CHFGNN **FAQ**	**1.00/2500**	**0.96/2052**	**0.60/1416**	0.003/865	0.003/869			

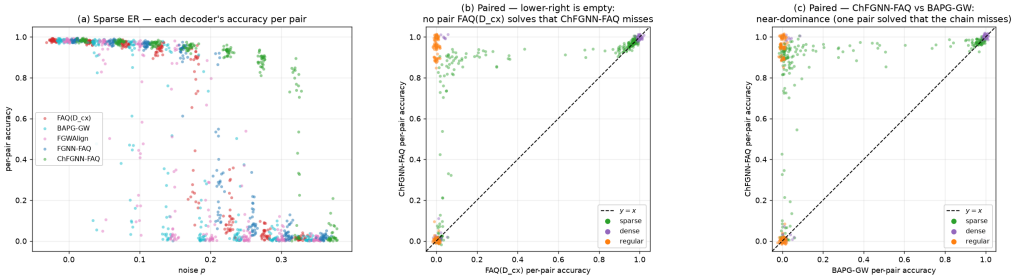


Figure 3: Per-pair analysis on the test pairs. (a) Per-pair accuracy of each decoder on sparse Erdős–Rényi graphs as a function of the noise: the transition is bimodal (pairs are either solved or not) and the two Gromov–Wasserstein solvers transition earliest. (b) ChFGNN **FAQ** against **FAQ**(D_{cx}) on every synthetic pair: the empty lower-right region means that no pair solved by the baseline is missed by the chain. (c) The same paired view against BAPG: a single pair (dense, $p_{\text{noise}} = 0.3$) is solved by BAPG and missed by the chain.

perparameter `mu = 1`. For **FAQ**(J), we relied on the SciPy implementation `scipy.optimize.quadratic_assignment(method='faq')`. For **FAQ**(D_{cx}), we used our own Frank–Wolfe implementation to compute D_{cx} before passing it to **FAQ**.

We emphasize that **FAQ** and FUGAL run on CPU, whereas our chained FGNNs run on GPU. Runtimes correspond to computing the number of common edges (**nce**) over 100 graph pairs of size $n = 500$. Thus Figure 4 is an implementation-level wall-clock comparison, not a hardware-agnostic efficiency claim. The more robust conclusion is that the chained FGNN reaches higher **nce** with a small number of learned refinement steps, while **FAQ**(D_{cx}) remains sensitive to the number and implementation quality of Frank–Wolfe iterations.

FUGAL is substantially faster than **FAQ**(D_{cx}). We lack an efficient implementation of the Frank–Wolfe solver required for the convex-relaxation initialization D_{cx}, and our implementation prioritizes correctness over speed. The subsequent **FAQ** step uses SciPy's efficient routine

Table 12: Accuracy (%) on sparse (average degree 4) and dense (average degree 80) Erdős Rényi random graphs at noise 0.2, as a function of the seed fraction. Note the different seed scales for the two regimes.

Sparse, average degree 4											
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
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
Dense, average degree 80											
Fraction of Seeds	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
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

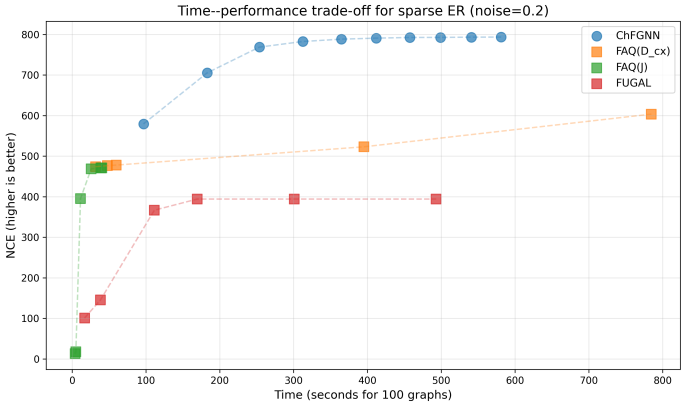


Figure 4: Number of common edges (**nce**) recovered in sparse Erdős–Rényi graphs as a function of wall-clock computation time allocated to each implementation: FUGAL and **FAQ** variants run on CPU, while ChFGNN runs on GPU. The plot should be read as an implementation-level trade-off, not as a hardware-independent speed comparison.

`quadratic_assignment`. We expect that **FAQ**(D_{cx}) could be made faster with an optimized implementation.

Per-pair runtimes of the baselines. Table 13 complements the Pareto curves with per-pair inference times measured on a single CPU core, for the Gromov–Wasserstein solvers of Section A.4 and, on the same laptop, for **FAQ**(D_{cx}) (dominated by our unoptimized Frank–Wolfe solver for D_{cx}) and for the full chain run on CPU. None of the baselines has a training phase; the chain's training is a one-off cost discussed in Section 5.2. For BAPG the wall-clock time of every pair is stored with the released results and averaged over the test pairs of each family; for FGWAlign only the per-cell averages printed during the runs were kept, and the table reports their range across noise levels. The memory footprint of all these methods is $O(n^2)$ (dense transport plans or cost products for the optimal-transport solvers, dense doubly stochastic iterates for **FAQ**); the chain's constant is larger by the feature dimension of the pair state (256 in our models, Section 5.2).

Table 14 reports the average number of Frank–Wolfe iterations of **FAQ** on sparse Erdős–Rényi graphs when it is started from D_{cx} or from the similarity matrix of the chained FGNN: the learned initialization substantially lowers the number of iterations.

Table 13: Inference time per graph pair at $n = 500$ on a single laptop CPU core (seconds). The GPU timings of the chain are given by the Pareto curves of Figure 4 (a few seconds per pair).

Method	sparse (d=4)	dense (d=80)	regular (d=10)
BAPG	7	3	< 0.1
FGWAlign	≈ 70	150–210	85–95
FAQ(D_{cx})	20–40	–	–
ChFGNN **FAQ** (full chain, CPU only)	≈ 90	–	–

Table 14: Average number of gradient projections (**Proj**) in the Frank-Wolfe algorithm **FAQ**, with initialization from either D_{cx} or the similarity matrix produced by the chained FGNNs.

ER 4 - NOISE	0.15	0.2	0.25	0.3	0.35
FAQ(D_{CX})	15.6	31.4	25.8	24.2	25.6
CHFGNN **FAQ**	4.1	6.5	8.3	15.1	19.7

A.6 GNN architecture and expressiveness

The choice of a more expressive architecture is crucial for our approach. The success of the chaining procedure critically depends on producing a high-quality initial similarity matrix $S^{A\to B}(0)$ to bootstrap the iterative refinement process. Standard MPNNs, which aggregate only local neighborhood information, would produce similarity matrices based on limited local features—insufficient for capturing the global structural patterns needed for effective graph alignment. This limitation has been observed in prior work: Nowak et al. [2018] implemented a similar initial step using MPNNs with limited success, while Azizian and Lelarge [2021] demonstrated the superiority of Folklore-type GNNs for this task. As we show in Section 5, combining our expressive architecture with the chaining procedure yields substantial performance improvements over single-step approaches.

Core architecture: Folklore-inspired residual layers. Our GNN's main building block is a residual layer that processes hidden states for all node pairs $(h^t_{i\to j})_{i,j} \in \mathbb{R}^{n\times n\times d}$, producing updated states $(h^{t+1}_{i\to j})_{i,j} \in \mathbb{R}^{n\times n\times d}$:

$$h^{t+1}_{i\to j} = h^t_{i\to j} + m_1\left(h^t_{i\to j}, \sum_\ell h^t_{i\to \ell} \odot m_0(h^t_{\ell\to j})\right), \tag{10}$$

where $m_0 : \mathbb{R}^d \to \mathbb{R}^d$ and $m_1 : \mathbb{R}^{2d} \to \mathbb{R}^d$ are multilayer perceptrons (MLPs) with graph normalization layers, and $\odot$ denotes component-wise multiplication.

This design differs from the original Folklore-type GNN [Maron et al., 2019] in three ways:

- **Residual connections**: The skip connection $h^t_{i\to j} + (\cdot)$ is a faithful neuralization of the 2-FWL update rule, which carries the colour from step t forward and aggregates new evidence on top of it. We initially adopted residuals for the usual training-stability reasons, but they turn out to also be the structurally correct choice from the 2-FWL perspective. Stable optimization is a side benefit, not the primary motivation.
- **Graph normalization**: Inspired by Cai et al. [2021], this ensures well-behaved tensor magnitudes across different graph sizes.
- **Single MLP in the multiplicative branch**: PPGN Maron et al. [2019] applies an MLP to *both* arguments of the component-wise product to argue, via universal approximation of power-sum polynomials, that each layer is as expressive as one 2-FWL step. We apply m_0 on only one branch, reducing per-layer compute and parameter count at the cost of a strict layer-by-layer expressivity guarantee: a single FGNN layer is not as expressive as a single PPGN layer. We show in Section A.7 (Theorem A.4) that this is recovered with doubled depth: a $2K$-layer FGNN of width $4d$ exactly simulates any K-layer PPGN of width d, and FGNN therefore inherits the full 2-FWL expressivity of PPGN.

Input and output transformations. The complete architecture consists of three main components:

1. **Input embedding**: The adjacency matrix $A \in \{0,1\}^{n\times n}$ is embedded into the initial hidden state $h^0_{i\to j} \in \mathbb{R}^{n\times n\times d}$ using a learned embedding layer that encodes edge presence/absence.
2. **Residual processing**: Multiple residual layers (10) transform the node-pair representations, capturing complex structural relationships.
3. **Node feature extraction**: The final tensor $h^k_{i\to j} \in \mathbb{R}^{n\times n\times d}$ is converted to node features $\mathbb{R}^{n\times d}$ via max-pooling over the second spatial dimension: $\text{node}_i = \max_j h^k_{i\to j}$.

Ranking integration for chained networks. The networks $g^{(1)}, g^{(2)}, \ldots$ must incorporate ranking information in addition to graph structure. For a graph of size n, the node at position $k \in \{0,\ldots,n-1\}$ of the ordering by increasing score is represented by its relative rank $\tilde{r} = k/n \in [0,1)$ (the implementation stores the increasing order, which is equivalent to the decreasing order of (7)). Only this relative value is used, never the absolute position, which is why a model trained on $n = 500$ can be evaluated on larger graphs. Ties between equal scores are ordered by the sorting routine (`numpy.argsort`) applied to the score vector, without any use of labels or of the planted permutation. In a chained FGNN, the input of the pair tensor has two channels: the adjacency matrix, and a second channel whose diagonal entry (j,j) carries the relative rank $\tilde{r}_j$ of node j (zero elsewhere), so that rank information enters the pair state on the diagonal and spreads through the residual pair-message-passing layers. After max-pooling, the node feature is concatenated with a sinusoidal positional encoding of the same relative rank (followed by a small MLP and a normalization), and $S^{A\to B}$ is the dot product of these concatenated node features. The similarity therefore decomposes as a structural term plus a learned kernel of the two relative ranks, but the structural term itself depends on the full rank fields through the pair tensor, so the similarity is not merely an additive sum of a rank-independent structural score and a rank score.

This architecture provides the expressiveness needed to capture global graph properties while remaining trainable through the sequential training procedure described in Section 3.2. While scalability remains a limitation for very large graphs, the architecture proves highly effective for the graph sizes considered in our experiments (up to 1000 nodes).

A.7 Expressivity: FGNN matches PPGN up to depth doubling

The FGNN block of (10) applies a single MLP m_0 inside the bilinear pair-aggregation, in contrast with PPGN [Maron et al., 2019] which applies one MLP to *each* argument. This raises the concern that FGNN is strictly less expressive than PPGN at the layer level. We prove this is an artefact of the layer-by-layer comparison: at the level of the formal architecture (entry-wise MLPs, no graph normalisation), **any K-layer PPGN can be exactly simulated by a $2K$-layer FGNN of tensor-channel width $4d$**, and FGNN therefore inherits the full 2-FWL expressivity of PPGN.

MLP class. Throughout this section, the MLP class $\mathcal{M}$ consists of feed-forward networks with arbitrary finite depth, arbitrary finite hidden widths, and a continuous activation σ that satisfies the *identity-realisability* property below.

Definition A.1 (Identity-realisable activation)**.** An activation $\sigma : \mathbb{R} \to \mathbb{R}$ is identity-realisable if there exist constants $a, b, c \in \mathbb{R}$ such that $a\,\sigma(x) + b\,\sigma(-x) + c = x$ for all x.

ReLU obviously satisfies $\sigma(x) - \sigma(-x) = x$.

Lemma A.2 (MLP closure)**.** *Let σ be identity-realisable and $\mathcal{M}$ the corresponding MLP class. Then $\mathcal{M}$ is closed under: (1) affine pre-composition $f \circ A$; (2) affine post-composition $B \circ f$ (in particular $-f \in \mathcal{M}$); (3) parallel branching $x \mapsto (f_1(x), f_2(x))$; (4) composition $f \circ g$; (5) the identity map $\mathbb{R}^p \to \mathbb{R}^p$ is in $\mathcal{M}$, with one hidden layer of width $2p$. The constructed MLPs have width bounded by the sum of those of the constituents plus an additive constant depending only on input/output dimensions and on σ. The depth is bounded by the maximum of those of the constituents for operations (1)–(3) and (5), and by the sum of their depths for operation (4).*

Proof. (1)–(2) absorb A, B into the first/last affine layer of f. (3) places f_1, f_2 in parallel branches and concatenates their outputs (padding the shallower branch with identity gadgets to match depths). (4) stacks the layers. (5) follows from identity-realisability: write $x_i = a\sigma(x_i) + b\sigma(-x_i) + c$ component-wise, using $2p$ hidden units. □

Definition A.3 (PPGN block)**.** A PPGN block with parameters (M_1, M_2, M_3), where $M_1, M_2 : \mathbb{R}^d \to \mathbb{R}^d$ and $M_3 : \mathbb{R}^{2d} \to \mathbb{R}^d$ are MLPs in $\mathcal{M}$ applied entry-wise, maps $h \in \mathbb{R}^{n\times n\times d}$ to

$$\mathrm{PPGN}(h)_{ij} \;=\; M_3\Bigg(h_{ij},\; \sum_{\ell=1}^{n} M_1(h_{i\ell}) \odot M_2(h_{\ell j})\Bigg).$$

The FGNN block is the residual block of (10), with $m_0, m_1 \in \mathcal{M}$ applied entry-wise and *without* graph normalisation; we denote it $\mathrm{FGNN}(h)$. Graph normalisation is treated separately in Remark A.7.

Theorem A.4 (FGNN simulates PPGN with depth doubling)**.** *Let $K \in \mathbb{N}$ and let $\mathcal{P} = \mathrm{PPGN}_K \circ \cdots \circ \mathrm{PPGN}_1$ be a K-layer PPGN of channel dimension d with MLP parameters $(M_1^{(k)}, M_2^{(k)}, M_3^{(k)})$ for $k = 1, \ldots, K$. There exist FGNN blocks $\mathrm{FGNN}_1, \ldots, \mathrm{FGNN}_{2K}$ of channel dimension $d' = 4d$ such that, for every input $h \in \mathbb{R}^{n\times n\times d}$,*

$$\iota^{-1}(\mathrm{FGNN}_{2K} \circ \cdots \circ \mathrm{FGNN}_1(\iota(h))) \;=\; \mathcal{P}(h),$$

where $\iota : \mathbb{R}^d \hookrightarrow \mathbb{R}^{4d}$ embeds into the first channel block ($\iota(x) = (x, 0, 0, 0)$) and ι^{-1} extracts it. Moreover, the internal hidden widths of the FGNN MLPs are bounded above by a constant times $W^\star + d$, where $W^\star$ is the maximum hidden width of $M_1^{(k)}, M_2^{(k)}, M_3^{(k)}$ over all k, the constant depending only on σ. The additive d term reflects the identity gadgets on $\mathbb{R}^d$ used in the construction (each of width $2d$ by Lemma A.2(5)).

Proof. It suffices to simulate one PPGN block: the construction below maps ι-form inputs to ι-form outputs and therefore composes cleanly across layers ($K = 0$ is vacuous). Write every $x \in \mathbb{R}^{4d}$ as $x = (x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)})$ with each block in $\mathbb{R}^d$.

For a PPGN block with parameters (M_1, M_2, M_3), define the first FGNN block by

$$m_0^{(1)} \equiv 0, \qquad m_1^{(1)}(x, y) = \big(0,\; M_1(x^{(1)}),\; M_2(x^{(1)}),\; 0\big).$$

By Lemma A.2, $m_1^{(1)} \in \mathcal{M}$ (drop y via affine pre-composition; run M_1, M_2 in parallel; place in blocks c_2, c_3 via affine post-composition). The first residual yields

$$h_{ij}^1 = \iota(h)_{ij} + m_1^{(1)}(\iota(h)_{ij}, 0) = \big(h_{ij},\; M_1(h_{ij}),\; M_2(h_{ij}),\; 0\big).$$

For the second FGNN block, set $m_0^{(2)}(z) = (0, z^{(3)}, 0, 0)$ (an affine map; in $\mathcal{M}$ via Lemma A.2(5)+(2)). The bilinear sum then evaluates to

$$\begin{aligned}\sum_\ell h_{i\ell}^1 \odot m_0^{(2)}(h_{\ell j}^1) &= \sum_\ell \big(h_{i\ell}, M_1(h_{i\ell}), M_2(h_{i\ell}), 0\big) \odot \big(0, M_2(h_{\ell j}), 0, 0\big) \\ &= \Big(0,\; \underbrace{\textstyle\sum_\ell M_1(h_{i\ell}) \odot M_2(h_{\ell j})}_{=:\, S_{ij}},\; 0,\; 0\Big).\end{aligned}$$

Define $m_1^{(2)}(x, y) = \big(M_3(x^{(1)}, y^{(2)}) - x^{(1)},\; -x^{(2)},\; -x^{(3)},\; 0\big)$. By Lemma A.2 (extract $x^{(1)}, y^{(2)}$ via affine pre-composition, compose with M_3, subtract identity gadget on $x^{(1)}$; negate identity gadgets on $x^{(2)}, x^{(3)}$; combine via parallel branching), $m_1^{(2)} \in \mathcal{M}$. Applying the second residual,

$$h_{ij}^2 = h_{ij}^1 + m_1^{(2)}\big(h_{ij}^1, (0, S_{ij}, 0, 0)\big) = \big(M_3(h_{ij}, S_{ij}),\; 0,\; 0,\; 0\big) = \iota\big(\mathrm{PPGN}(h)_{ij}\big),$$

which is in ι-form, so the cycle composes for the next PPGN block. The width claim follows from Lemma A.2: each constructed MLP uses parallel copies of M_1, M_2, M_3, affine projections, and a constant number of identity gadgets at $2d$ each. □

Corollary A.5 (2-FWL expressivity)**.** *Equip PPGN with any graph-level read-out R (e.g. summation over the diagonal of the final-layer tensor followed by an MLP), and equip the simulating FGNN with the read-out $R \circ \iota^{-1}$ on the final $4d$-channel tensor. For any pair of graphs (G_1, G_2) that the 2-FWL test distinguishes, there exists a finite-depth FGNN with this read-out that produces different graph-level representations on G_1 and G_2.*

Proof. By Maron et al. [2019], a sufficiently deep PPGN with the chosen read-out distinguishes any 2-FWL-distinguishable pair. Theorem A.4 applied at that depth produces, after ι^{-1}, the same per-pair tensor as the PPGN; the read-outs then coincide. □

Remark A.6 (Converse direction). PPGN is at least as expressive as FGNN at the block level: setting $M_1 = \mathrm{id}_{\mathbb{R}^d}$ in Definition A.3 reproduces the FGNN bilinear term, and M_3 accommodates the residual connection. So FGNN $\equiv$ PPGN in expressivity, up to a constant factor in depth and tensor-channel width.

Remark A.7 (Graph normalisation: scope of the result). The FGNN of our experiments composes each m_0, m_1 with a graph-normalisation operator GN that subtracts a per-channel mean over all n^2 pair entries and divides by a per-channel standard deviation. GN is a tensor-wide reduction; it is not entry-wise and therefore not in $\mathcal{M}$. The GN-augmented FGNN is, strictly, outside the scope of the theorem above.

We do *not* prove that adding GN preserves the equivalence: a naive attempt fails because after GN the channel-block c_3 of h^1 contains a per-channel rescaling of $M_2(h)$, not $M_2(h)$, and the bilinear identity producing S_{ij} no longer holds verbatim. Whether some compensating construction recovers the equivalence with GN is left open. Theorem A.4 settles only the single-m_0 question — which is the substantive concern; GN is a separate practical ingredient. Empirically, the GN-augmented FGNN solves correlated regular graph alignment (Section 5), a regime where 1-WL provably fails, providing direct evidence that the trained network exhibits expressivity strictly above 1-WL.

A.8 Technical details for the GNN architecture and training

By default, we use MLPs for the functions m_0 and m_1 in (10) with 2 hidden layers of dimension 256. In all our experiments, we take a GNN with 2 residual layers. We used the Adam optimizer with a learning rate of 10^{-4} and the scheduler ReduceLROnPlateau from PyTorch with a patience parameter of 3.

For **Proj**, we use the function **linear_sum_assignment** from **scipy.optimize** and for **FAQ**, we use the function **quadratic_assignment** from the same library. SciPy is a set of open source (BSD licensed) scientific and numerical tools for Python. In order to compute D_{cx} solving (4), we implemented the Frank-Wolfe algorithm.

For the training and inference, we used Nvidia RTX8000 48GB and Nvidia A100 80GB. For graphs of size 500, we train on 200 graphs and validate on 100 graphs for 100 epochs. We run for $L = 15$ steps of chaining (obtaining 16 trained FGNNs: $f, g^{(1)}, \ldots, g^{(15)}$), each stage being trained for 100 epochs on the outputs of the previous stage; the reduced three-stage chain of Section 5.2 was trained in about two GPU-hours. The training noise levels of the released chains are 0.22 (sparse), 0.24 (dense) and 0.11 (regular). The PyTorch code, the pretrained chains and the per-pair results of all tables are available at `https://github.com/mlelarge/chaining-gnn-graph-alignment`; the degree-normalized and random-order ablations of Appendix A.9 are options of the released pipeline.

A.9 Ranking ablations and per-stage analysis

All experiments in this section use the sparse Erdős–Rényi benchmark ($n = 500$, $d = 4$) and a 30 test pairs per noise level (same as in Section A.4); the reference chain is the released chain trained at $p_{\text{noise}} = 0.22$ with the raw score.

Protocol. The *degree-normalized* variant ranks node i by $\text{score}(i)/\max(\deg_A(i), 1)$ instead of $\text{score}(i)$. The *random order* keeps the decoded matching but assigns the n rank values to the matched pairs in a uniformly random order, drawn independently at every stage; matched nodes still share a rank value, so the matching remains recoverable from the two rank fields, but the values no longer indicate confidence. Each variant is evaluated in two ways: *at inference*, the raw-trained chain is run with the modified ranking (a stress test, since the stages then receive an ordering convention they were not trained on); *retrained*, a new chain is trained from scratch with the modified ranking at every stage, with the same architecture, data and chain length as the reference chain. Test accuracies with **FAQ** decoding are given in Table 15.

Table 15: Ranking ablations on sparse Erdős–Rényi graphs (n=500, d=4): accuracy with **FAQ** decoding on 30 fixed test pairs per noise level. *Raw* is the released chain, trained at p_{noise}=0.22 with the raw score. *At inference*: the raw-trained chain run with the modified ranking. *Retrained*: a chain trained from scratch with the modified ranking at every stage, with the same architecture, data and chain length.

		DEGREE-NORMALIZED SCORE		RANDOM ORDER	
p_{NOISE}	RAW (TRAINED)	AT INFERENCE	RETRAINED	AT INFERENCE	RETRAINED
0.15	0.951	0.941	0.952	0.914	0.947
0.20	0.928	0.751	0.924	0.346	0.802
0.25	0.884	0.272	0.840	0.092	0.331
0.30	0.440	0.081	0.465	0.040	0.097
0.35	0.033	0.023	0.033	0.017	0.027

Training trajectories under the ablations. Table 16 gives the validation accuracy (with **Proj** decoding, at the training noise) of each stage of the three retrained chains. The first stage receives no ranking and is identical for the three chains. With either score-based ranking, the accuracy rises steeply over the first stages and saturates; the degree-normalized chain tracks the raw chain within six points at every stage. The random-order chain also improves monotonically, which confirms that the transported matching alone carries usable signal, but it plateaus far below the score-ordered chains and is still climbing by about 0.01 per stage at the end of the chain: the score order reaches by the third stage an accuracy that the random order does not reach after fifteen. This is the practical content of the ranking: the transported matching makes chaining work at all, and the confidence ordering makes it reach useful accuracy in a feasible number of stages.

Table 16: Per-stage validation accuracy (**Proj** decoding, training noise $p_{\text{noise}} = 0.22$, sparse Erdős–Rényi graphs) of chains trained with the raw score, the degree-normalized score and a random order. Stage 0 is the network f, which receives no ranking.

stage	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
raw score	.09	.18	.30	.43	.55	.66	.73	.77	.80	.82	.83	.84	.85	.86	.87
degree-normalized	.09	.17	.27	.37	.49	.59	.66	.71	.74	.76	.77	.79	.80	.81	.81
random order	.09	.12	.13	.15	.16	.18	.18	.19	.20	.21	.23	.23	.25	.26	.27

Correction and retention. To test directly whether later stages correct the errors of earlier ones rather than merely preserving good matches, we track every node of the 30 test pairs at test noise $p_{\text{noise}} = 0.2$ through the released chain (**Proj** decoding at each stage). For each transition $k \to k{+}1$ we report the *correction rate*, the probability that a node that is misassigned at stage k and lies in the bottom half of stage k's ranking is correctly assigned at stage $k{+}1$, and the *retention rate*, the probability that a node correctly assigned at stage k and lying in the top half of the ranking is still correct at stage $k{+}1$ (Table 17). At its peak, one stage corrects 35% of all bottom-ranked errors while retaining essentially every top-ranked correct match. The correction rate first rises, as the set of reliable matches grows, then falls as the remaining error pool becomes smaller and harder. This rise separates two readings of what the ranking does. If the ranking only broke ties among structurally indistinguishable nodes, or only sharpened the similarities of nodes that are already correctly matched, a wrong node at the bottom of the ranking would be corrected at a rate independent of how many reliable matches the previous stage provides. Instead, the probability that such a node is corrected grows with the number of reliable matches, from 0.14 to 0.35, which is what one expects if the reliable matches serve as evidence for re-solving the remaining nodes (Section 3). Consistently, the mean **nce** of the **Proj**-decoded alignment over the 30 pairs increases monotonically along the chain, $115 \to 226 \to 369 \to 522 \to 633 \to 701 \to 736 \to 756 \to 767 \to 771 \to 775 \to 777$, and then plateaus; see also Figure 10.

Train/test mismatch and chain length. Each chain is trained at a single noise level (sparse 0.22, dense 0.24, regular 0.11) and evaluated on the full test grid (Tables 20–25, Figure 8). Performance is stable at all noise levels at or below the training noise and degrades gracefully beyond it: chaining helps once the first stages reach a basin with enough reliable pairs, and otherwise falls back toward

Table 17: Correction and retention rates per stage of the released sparse chain at test noise $p_{\text{noise}} = 0.2$ (30 test pairs). acc@k is the **Proj** accuracy of stage k; counts in parentheses are corrected nodes over bottom-half misassigned nodes.

stage $k \to k{+}1$	acc@k	correction (bottom half, wrong at k)	retention (top half, correct at k)
$0 \to 1$	0.12	0.14 (1019/7123)	0.92
$1 \to 2$	0.23	0.25 (1698/6937)	0.97
$2 \to 3$	0.40	0.33 (2147/6506)	0.99
$3 \to 4$	0.58	0.35 (1883/5422)	1.00
$5 \to 6$	0.79	0.27 (815/2991)	1.00
$8 \to 9$	0.88	0.11 (182/1671)	1.00

single-stage behavior; in Table 2 it never falls below the single-FGNN baseline. Accuracy and **nce** saturate along the chain (Table 19, Figure 10), so beyond the plateau additional stages or loops cost computation but not accuracy; the number of stages is a compute budget rather than a sensitive hyperparameter, and the reduced three-stage chain of Section 5.2 already recovers most of the accuracy.

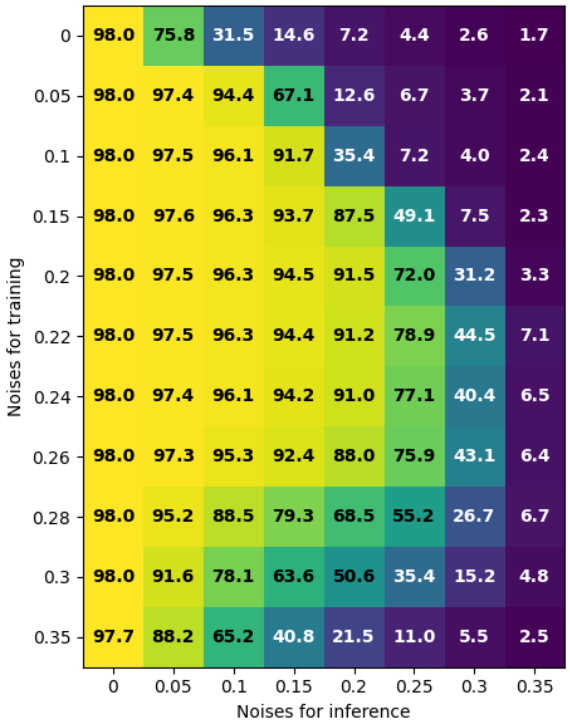


Figure 5: Each line corresponds to chained FGNNs trained at a given level of noise and evaluated across all different level of noises. Performances are **acc** (in %) for sparse Erdős-Rényi graphs with **Proj** as post-processing.

Why the gains are larger on sparse graphs. The transition is sharper on dense graphs and the window where chaining helps is narrower. Two quantities explain this. In an Erdős–Rényi graph with average degree d and edge probability $\lambda = d/n$, the relative spread of degrees is $\sqrt{(1-\lambda)/d} \approx 1/\sqrt{d}$: large when sparse, so that heterogeneous degrees provide strong anchors, and shrinking as d grows, until the graph resembles the regular case where the first assignment is degenerate. The edge evidence used by the ranking behaves in the same way: a matched neighbor pair corroborates a true candidate with probability $\lambda(1-p_{\text{noise}})$ but a wrong one with probability λ^2, a contrast ratio $(1-p_{\text{noise}})/\lambda$ that shrinks as the graph densifies or as the noise grows. The advantage of chaining is therefore one of relative contrast, not of sparse instances being uniformly easier.

Uninformative rank fields. The random-order ablation keeps the matching in the rank fields. A *fully* random rank field, drawn independently for the two graphs, carries no information about the correspondence and cannot help in the following sense.

Proposition A.8. *Let the rank fields (r^A, r^B) given to a stage be drawn independently of $(G_A, G_B, \pi^\star)$. For any loss function, the minimal expected loss over predictors that observe (A, B, r^A, r^B) equals the minimal expected loss over predictors that observe (A, B) only.*

Proof. By independence, the conditional law of $\pi^\star$ given (A, B, r^A, r^B) equals its conditional law given (A, B). Hence a Bayes-optimal predictor for the enlarged observation can be chosen to depend on (A, B) only, and the two Bayes risks coincide; any (possibly randomized) predictor of the enlarged observation has expected loss at least this Bayes risk. □

The proposition is a statement about information, not about a fixed finite architecture: independent random inputs can still act as symmetry-breaking features for a GNN [Sato et al., 2021, Abboud et al., 2021], and selecting the best of several stages by **nce** is a separate effect. It covers neither the transported random order, which does carry the matching, nor the score order.

A.10 More related work

Supervised learning approaches for the graph matching problem have been extensively studied in the computer vision literature [Wang et al., 2021], [Rolínek et al., 2020], [Zanfir and Sminchisescu, 2018], [Gao et al., 2021], [Yu et al., 2021b], [Jiang et al., 2022]. [Fey et al., 2020] is closely related to our work and proposes a two-stage architecture similar to our chaining procedure with MPNNs. The first stage is the same as our first step but with an MPNN instead of our FGNN. Then the authors propose a differentiable, iterative refinement strategy to reach a consensus of matched nodes. All these works assume that non-topological node features are available and informative. This is a setting favorable to GNNs as node-based GNNs are effective in learning how to extract useful node representations from high-quality non-topological node features. In contrast, we focus on the pure combinatorial problem where no side information is available. In [Li et al., 2019], graph matching networks take a pair of graphs as input and compute a similarity score between them. This algorithm can be used to compute the value of the graph matching (1) but does not give the optimal permutation $\pi^{A\to B}$ between the two graphs which is the main focus of our work. [Lagesse and Lelarge, 2025] propose a benchmarking methodology for GNNs based on graph alignment, and show that the same task also serves as an effective objective for unsupervised GNN pre-training.

Regarding benchmarks for the combinatorial GAP, we are not aware of any publicly available dataset. The GAP can be seen as a particular version of the QAP and some algorithms designed for the GAP can be used for QAP instances (i.e. with weighted adjacency matrices). This is the case for the convex and indefinite relaxations presented in Section 2.2 which can be used with real-valued matrices. In particular, [Lyzinski et al., 2015] shows very good performances of **FAQ** on some QAP instances from [Burkard et al., 1997]. These instances are small (from 12 to 40 nodes) with full (integer-valued) matrices. They are very far from the distribution of correlated random graphs used for training in our work and we do not expect good performances for such out-of-distribution instances for any supervised learning algorithm.

A.11 Model-based versus simulation-based algorithms

As explained in Section 2.3, we train and evaluate our supervised learning algorithms on correlated random graphs. This choice connects our work to a rich theoretical literature on the correlated Erdős-Rényi random graph ensemble, which has been extensively studied from an information-theoretic perspective.

A.11.1 Theoretical Foundations and Limits

The theoretical analysis begins with Cullina and Kiyavash [2016], which establishes the information-theoretic limit for exact recovery of $\pi^\star$ as the number of nodes n tends to infinity. In the sparse regime, where the average degree d remains constant as $n \to \infty$, exact recovery becomes impossible. Subsequent work by Ganassali et al. [2021] and Ding and Du [2023] demonstrates that partial recovery of $\pi^\star$ is only possible when $p_{\text{noise}} < 1 - d^{-1}$.

For the correlated Erdős-Rényi ensemble, the joint probability distribution is given by:

$$\mathbb{P}(G_A, G_B) \propto \left(\frac{(1 - p_{\text{noise}})(n - d(1 + p_{\text{noise}}))}{d p_{\text{noise}}^2} \right)^{e(G_A \wedge G_B)},$$

where $e(G_A \wedge G_B) = \sum_{i<j} A_{ij} B_{ij}$ counts the common edges between graphs G_A and G_B. This distribution reveals a crucial insight: **the maximum a posteriori estimator of π^* given G_A and the permuted graph G'_B is exactly a solution of the GAP on the (G_A, G'_B) instance**.

A.11.2 Model-based Approaches: Achievements and Limitations

Recent theoretical advances have produced efficient polynomial-time algorithms [Ding et al., 2021, Fan et al., 2023, Ding and Li, 2023, Ganassali et al., 2024a, Piccioli et al., 2022] that approximate the probability distribution by exploiting structural properties like the local tree-like nature of sparse random graphs. These algorithms achieve partial recovery (positive accuracy) when p_{noise} is sufficiently small, though well below the information-theoretic threshold of $1 - d^{-1}$.

However, a fundamental **algorithmic threshold** appears to exist. Recent work [Mao et al., 2023, Ganassali et al., 2024b] suggests that no efficient algorithm can succeed for $p_{\text{noise}} > p_{\text{algo}} = 1 - \sqrt{\alpha} \approx 0.419$, where α is Otter's constant, even when the average degree d is large.

While these model-based algorithms provide theoretical guarantees for correlated Erdős-Rényi graphs, they suffer from significant practical limitations:

- **Narrow applicability**: Designed specifically for the correlated Erdős-Rényi model with no guarantees outside this distribution
- **Computational complexity**: Despite polynomial-time guarantees, running times are often impractical for real applications
- **Limited scalability**: Most implementations prioritize mathematical rigor over computational efficiency

Muratori and Semerjian [2024] represents a notable exception, focusing on making message-passing algorithms [Ganassali et al., 2024a, Piccioli et al., 2022] more scalable while maintaining theoretical guarantees.

A.11.3 FAQ: An Empirical Surprise

Remarkably, the **FAQ** algorithm—which was not designed specifically for any random graph model—empirically encounters the same algorithmic barrier predicted by theory. As shown in Figure 5, **FAQ**'s performance degrades sharply near p_{algo}, matching the theoretical predictions despite lacking formal guarantees for this setting. **FAQ** only underperforms compared to specialized message-passing methods Muratori and Semerjian [2024] when p_{noise} approaches p_{algo}.

This empirical observation suggests that **FAQ**, through its continuous relaxation approach, implicitly captures fundamental structural properties of the graph alignment problem that transcend specific random graph models.

A.11.4 Our Simulation-based Approach

To circumvent the computational challenges of maximizing the exact posterior (which, as shown above, corresponds exactly to solving the GAP), we adopt a **simulation-based approach**. Rather than deriving model-specific algorithms, we:

1. **Sample training data**: Generate pairs of graphs (G_A, G_B) with known alignment permutations $\pi^\star$
2. **Learn mappings**: Train neural networks to map graph pairs to similarity matrices $S^{A \to B} \in \mathbb{R}^{n \times n}$
3. **Extract solutions**: Convert similarity matrices to permutations via projection or as **FAQ** initialization

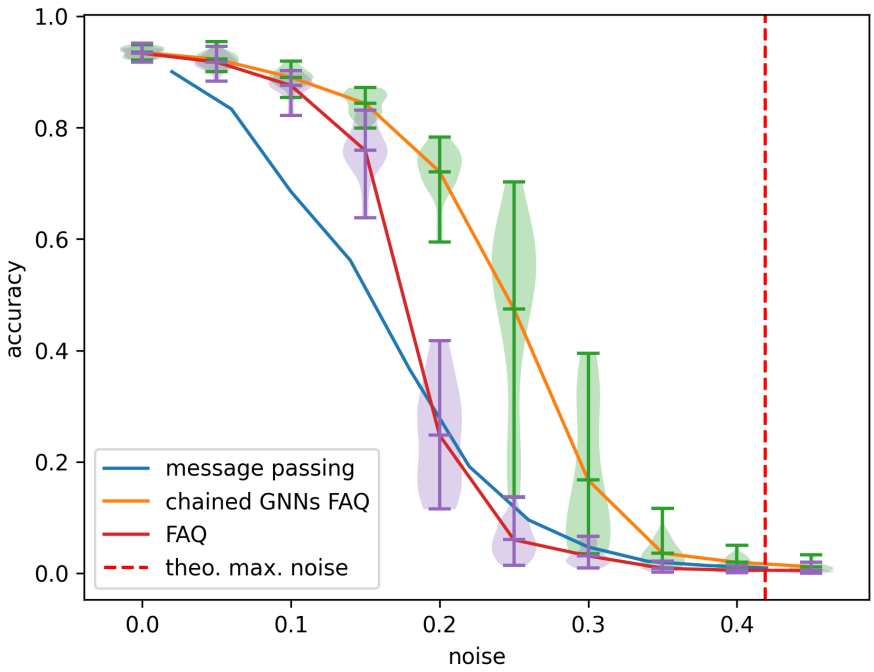


Figure 6: Accuracy **acc** as a function of the noise level for correlated Erdős-Rényi random graphs with size $n = 1000$ and average degree $d = 3$. Chained GNNs were trained at noise level 0.25 and **FAQ** is used as the last step for the inference. The red curve labeled FAQ corresponds to **FAQ**(D_{cx}) and the blue curve labeled message passing are results from [Muratori and Semerjian, 2024]. The dashed vertical line corresponds to the theoretical $p_{\text{algo}} = 1 - \sqrt{\alpha}$ above which no efficient algorithm is known to succeed.

Key Advantages. Our simulation-based approach offers several advantages over both model-based methods and traditional relaxations:

Supervised learning with ground truth: Unlike the convex relaxation (4), we have access to ground truth permutations during training, enabling more informative loss functions.

Better optimization objective: Instead of the Frobenius norm used in convex relaxation, we employ cross-entropy loss, which provides more informative gradients for discrete matching problems.

Generalization potential: While trained on specific distributions, our learned representations may capture general structural patterns applicable beyond the training distribution.

Hybrid capability: Our similarity matrices can initialize traditional solvers like **FAQ**, combining the benefits of learning and optimization approaches.

This simulation-based methodology bridges the gap between theoretical guarantees and practical performance, achieving strong empirical results while maintaining computational tractability.

A.12 Relating GAP to Gromov-Hausdorff, Gromov-Monge and Gromov-Wasserstein distances for finite metric spaces

We consider a simple case of discrete spaces with the same number of elements n and where $A, B \in \mathbb{R}^{n\times n}$ are the distance matrices of two finite metric spaces (X, d_X) and (Y, d_Y), i.e. $A_{ij} = d_X(x_i, x_j)$ and $B_{ij} = d_Y(y_i, y_j)$. Recall that we denote by $\mathcal{S}_n$ the set of permutation matrices and by $\mathcal{D}_n$ the set of doubly stochastic matrices. We also denote by $\mathcal{R}_n$ the set of matrices $R \in \{0, 1\}^{n\times n}$ such that $\sum_i R_{ij} \geq 1$ and $\sum_j R_{ij} \geq 1$.

The **Gromov-Hausdorff distance** for finite metric spaces can be written as:

$$\mathrm{GH}_L(A,B) = \min_{R\in\mathcal{R}_n} \max_{i,j,k,\ell} L(A_{ik},B_{j\ell})R_{ij}R_{k\ell} \tag{11}$$

where $L(a,b) \geq 0$. It is often desirable to smooth the max operator in (11) to a sum. This can be done by considering the related problem:

$$\mathrm{GM}_L(A,B) = \min_{R\in\mathcal{R}_n} \sum_{i,j,k,\ell} L(A_{ik},B_{j\ell})R_{ij}R_{k\ell} \tag{12}$$

Note that if we replace the constraint $R \in \mathcal{R}_n$ by $P \in \mathcal{S}_n$, we get the permutation-constrained version

$$\mathrm{GM}_L^{\mathcal{S}}(A,B) = \min_{P\in\mathcal{S}_n} \sum_{i,j,k,\ell} L(A_{ik},B_{j\ell})P_{ij}P_{k\ell} = \min_{\pi\in\mathcal{S}_n} \sum_{i,j} L\left(A_{ij},B_{\pi(i)\pi(j)}\right),$$

which is called **Gromov-Monge distance**.

The **Gromov-Wasserstein distance** is a relaxation of the Gromov-Hausdorff distance and is defined in Mémoli [2011]:

$$\mathrm{GW}_L(A,B,p,q) = \min_{T\in\mathcal{C}_{p,q}} \sum_{i,j,k,\ell} L(A_{ik},B_{j\ell})T_{ij}T_{k\ell}, \tag{13}$$

where p,q are probability distributions on X, Y and the minimum is taken over $\mathcal{C}_{p,q} = \{T \in \mathbb{R}_+^{n\times n},\, T\mathbf{1}=p,\, T^T\mathbf{1}=q\}$. Taking $p=q=\mathbf{1}/n$ the uniform distribution, we have $\mathcal{C}_{p,q} = \frac{1}{n}\mathcal{D}_n$ and $\mathrm{GW}_L(A,B,\mathbf{1}/n,\mathbf{1}/n)$ is a relaxed version of $\mathrm{GM}_L(A,B)$. We typically consider $L(a,b)=|a-b|^2$, and then we get:

$$\mathrm{GM}_{L^2}^{\mathcal{S}}(A,B) = \min_{\pi\in\mathcal{S}_n} \sum_{i,j} (A_{ij} - B_{\pi(i)\pi(j)})^2,$$

and with the simplified notation $\mathrm{GW}_{L^2}(A,B) = \mathrm{GW}_{L^2}(A,B,\mathbf{1}/n,\mathbf{1}/n)$,

$$\begin{aligned} n^2\mathrm{GW}_{L^2}(A,B) &= \min_{D\in\mathcal{D}_n} \sum_{i,j,k,\ell} (A_{ik} - B_{j\ell})^2 D_{ij}D_{k\ell} \\ &= \min_{D\in\mathcal{D}_n} \sum_{i,k} A_{ik}^2 + \sum_{j\ell} B_{j\ell}^2 - 2\sum_{i,j,k,\ell} A_{ik}B_{j\ell}D_{ij}D_{k\ell} \\ &= \|A\|_F^2 + \|B\|_F^2 - 2\max_{D\in\mathcal{D}_n} \langle AD, DB\rangle. \end{aligned}$$

In the particular case where A and B are positive semidefinite matrices, i.e. $A = U^TU$ and $B = V^TV$, we have: $\langle AD, DB\rangle = \|UDV^T\|_F^2$ which is a convex function of D and is always maximized at an extremal point of its constraint polytope $\mathcal{D}_n$. By Birkhoff's theorem, the extremal points of $\mathcal{D}_n$ are permutation matrices. Therefore, we have: $n^2\mathrm{GW}_{L^2}(A,B) = \mathrm{GM}_{L^2}^{\mathcal{S}}(A,B)$ in this case. Maron and Lipman [2018] shows that a similar result holds for Euclidean distances, when $A_{ij} = \|x_i - x_j\|_2$ and $B_{ij} = \|y_i - y_j\|_2$. Hence, we have:

Proposition A.9. *For A, B Euclidean distance matrices, the indefinite relaxation* (5) *is tight and solves the GAP* (1)*. In this case, the GAP computes the Gromov-Monge distance and the indefinite relaxation computes n^2 times the Gromov-Wasserstein distance under uniform marginals.*

A.13 Notations used in tables

- **acc FAQ**(D_{cx}) means the accuracy of **FAQ** algorithm initialized with D_{cx}.
- **acc** ChFGNN **Proj** means the accuracy of our chained FGNNs with **Proj** as the last step.
- **acc** ChFGNN **FAQ** means the accuracy of our chained FGNNs with **FAQ** as the last step.
- **nce FAQ**(D_{cx}) means the number of common edges found by **FAQ** algorithm initialized with D_{cx}.
- **nce FAQ**($\pi^\star$) means the number of common edges found by **FAQ** algorithm initialized with $\pi^\star$.
- **nce** ChFGNN **Proj** means the number of common edges found by our chained FGNNs with **Proj** as the last step.
- **nce** ChFGNN **FAQ** means the number of common edges found by our chained FGNNs with **FAQ** as the last step.

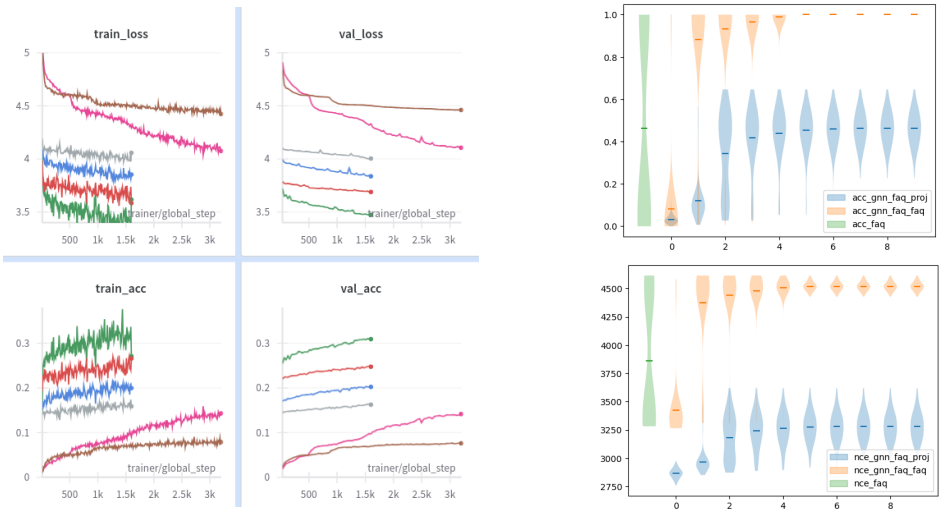


Figure 7: Left: Training chained GNNs. Each color corresponds to a different training and GNN: the first training (brown) reaches an accuracy below 0.1. The second training (magenta) uses as input the output of the first training and get an accuracy ≈ 0.15. The remaining trainings using the output of the previous training as input and reach higher and higher accuracy.
Right top: **acc** bottom: **nce**. First Violin plot (**_faq** green) for **FAQ**, then all other Violin plots correspond to a different number of iterations $N_{\max} = 0, 1, \ldots, 9$ of the chaining procedure (**_gnn_faq_proj** blue with **Proj** and **_gnn_faq_faq** orange with **FAQ**).

A.14 Bernoulli graphs: Generalization properties for chained GNNs

A.14.1 Training chained GNNs

For the same dataset as in Lyzinski et al. [2015] (see Bernoulli graphs in Section A.1), we plot in Figure 6 the training curves for the chained GNNs for the loss and the accuracy on the training set and the validation set. We do not see any overfitting here as values are similar on both sides. Chaining is very effective in this case: while the first training (brown) corresponding to the mapping f in (6) saturates at an accuracy below 0.1, the second training (magenta) corresponding to the mapping $g^{(1)}$ in (8) reaches a much higher accuracy because it uses the information about the graph matching contained in the output of the first training f. The curves for the remaining trainings are indeed ordered. This is due to the fact that for the training of $g^{(k+1)}$, we initialized it with the weights obtained after the training of $g^{(k)}$ in order to speed up the training. Since we observe a saturation in the learning of the $g^{(k)}$ for $k \geq 2$, we stop the training after half the number of epochs used for f and $g^{(1)}$.

A.14.2 Efficient inference for chained FGNNs

These results suggest an extreme form of looping: since $g^{(1)}$ allows to improve the accuracy of an initial guess (given by f), we can keep only the GNNs f and $g^{(1)}$, and we loop through $g^{(1)}$ for a fixed number of steps $N_{\max}$. Figure 6 gives the accuracy **acc** and the number of common edges **nce** defined in (3) for the inference procedure as a function of the number of iterations $N_{\max}$ made on $g^{(1)}$. We give (in blue) the performances of **Proj**, and (in orange) the performances of **FAQ** applied on the similarity matrix obtained after L loops. We see that the Frank-Wolfe algorithm used in **FAQ** used as the last step of our chaining procedure is crucial to get better performances. Indeed, we need only $N_{\max} = 5$ loops in order to get a perfect accuracy **acc** $= 1$ with **FAQ**.

We also give (in green) the performance of **FAQ** applied on the matrix D_{cx} (the default option in the **FAQ** algorithm). Indeed, **FAQ**(D_{cx}) is able to find the correct permutation for the graph matching in 13% of the cases and is stuck in a local maxima with a very small (less than 20%) accuracy otherwise. This bimodal behavior is due to the fact that D_{cx} gives very little information about the correct

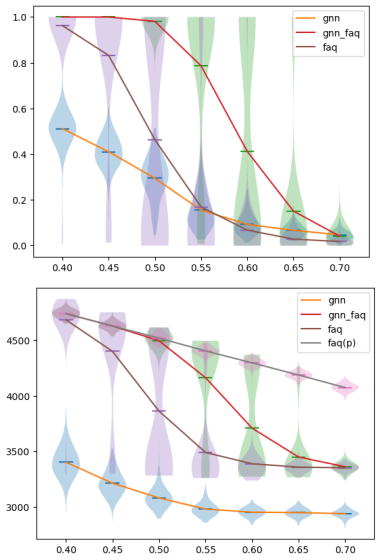


Figure 8: Bernoulli graphs: **acc** (top) and **nce** (bottom) as a function of the noise level. Chained FGNNs were trained at noise level 0.5. **gnn** (resp. **gnn_faq**) for chained FGNNs with **Proj** (resp. **FAQ**) as the last step. **faq** for **FAQ**(D_{cx}) and **faq(p)** for **FAQ**$(\pi^\star)$.

permutation. In contrast, **the chaining procedure was able to learn a much better initialization than D_{cx} for FAQ allowing to improve the accuracy from 50% to an exact accuracy.**

We now explore the generalization properties of the chaining procedure by applying the inference procedure described in Section A.14.2 on datasets with different noise levels. The level of noise used during training (described in Section A.14.1) is 0.5. Figure 7 gives the accuracy **acc** and the number of common edges **nce** for the inference procedure as a function of the noise level. We stop the inference loop when the **nce** obtained after applying **FAQ** to the similarity matrix is not increasing anymore. The red curve gives the performances of our chaining procedure with **FAQ** as the last step, the orange curve gives the performances of our chaining procedure with **Proj** as the last step. We compare our chaining procedure to **FAQ**(D_{cx}) in brown and to **FAQ**$(\pi^\star)$ in grey which corresponds to the maximum number of common edges for these noise levels. The curve for the accuracy of **FAQ**(D_{cx}) is similar to the one obtained in Lyzinski et al. [2015]. Our chaining procedure is able to generalize to noise levels different from the one used during training and outperforms **FAQ**(D_{cx}) in all cases. Indeed with a noise level less than 0.5, our chaining procedure recovers the correct permutation for the graph matching problem. Note that we did not try to optimize the performances of our chaining procedure with **Proj** as the last step, and they are indeed increasing if we allow for more loops.

	noise	0.4	0.45	0.5	0.55	0.6	0.65	0.7
	acc Proj(D_{cx})	0.3428	0.1956	0.1209	0.0815	0.0552	0.0411	0.0309
	acc FAQ(D_{cx})	1.0	0.9954	0.9531	0.6910	0.2621	0.0959	0.0225
	nce Proj(D_{cx})	3147.7	3000.0	2960.2	2945.8	2942.0	2942.4	2933.6
	nce FAQ(D_{cx})	4737.8	4622.8	4462.2	4056.1	3564.9	3408.0	3352.8
training 0.5	**acc** ChFGNN **Proj**	0.9994	0.9962	0.9639	0.7842	0.3400	0.1442	0.0737
	acc ChFGNN **FAQ**	1.0	1.0	1.0	0.9915	0.8949	0.5105	0.1267
	nce ChFGNN **Proj**	4736.1	4617.1	4439.4	4025.8	3319.0	3085.4	3038.0
	nce ChFGNN **FAQ**	4737.8	4629.0	4520.0	4395.8	4188.4	3747.2	3413.5

Table 18: Accuracy **acc** and number of common edges **nce** for Bernoulli graphs as a function of the noise p_{noise}.

A.15 Additional results for sparse Erdős-Rényi graphs

Table 19: Accuracy (**acc**) for sparse Erdős-Rényi graphs as a function of the number $(L+1)$ of trained FGNNs. Parentheses report the gain from extending inference to $N_{\text{loop}} = 60$ loops instead of stopping after the $L+1$ trained chain steps. Last line: number of loops for chained FGNNs as a function of the noise p_{noise} to get optimal **nce** (with a maximal $N_{\text{loop}} = 100$).

NOISE	0.15	0.2	0.25	0.3	0.35
L+1=2	0.28 (+0.02)	0.15 (+0.02)	0.08 (+0.01)	0.04 (+0.01)	0.02 (+0.00)
L+1=6	0.59 (+0.01)	0.43 (+0.06)	0.21 (+0.11)	0.07 (+0.05)	0.03 (+0.01)
L+1=10	0.85 (+0.01)	0.72 (+0.06)	0.43 (+0.13)	0.11 (+0.19)	0.04 (+0.03)
L+1=14	0.91 (+0.00)	0.86 (+0.01)	0.57 (+0.13)	0.16 (+0.21)	0.04 (+0.04)
L+1=16	0.92 (+0.00)	0.88 (+0.01)	0.61 (+0.12)	0.19 (+0.26)	0.04 (+0.04)
#LOOP	15	23	88	91	73

Figure 9 gives the performance of our chained GNNs trained at noise level 0.25 for sparse Erdős-Rényi graphs with average degree $d = 4$ and size $n = 500$. We observe that our chaining procedure is able to generalize to noise levels different from the one used during training and outperforms **FAQ**(D_{cx}) in all cases.

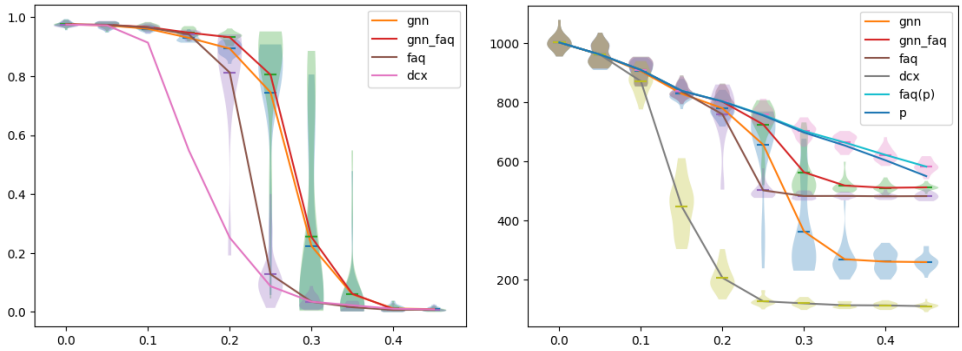


Figure 9: Sparse Erdős-Rényi graphs: **acc** (top) and **nce** (bottom) as a function of the noise level. Chained FGNNs were trained at noise level 0.25. **gnn** (resp. **gnn_faq**) for chained FGNNs with **Proj** (resp. **FAQ**) as the last step. **faq** for **FAQ**(D_{cx}) and **faq(p)** for **FAQ**$(\pi^\star)$.

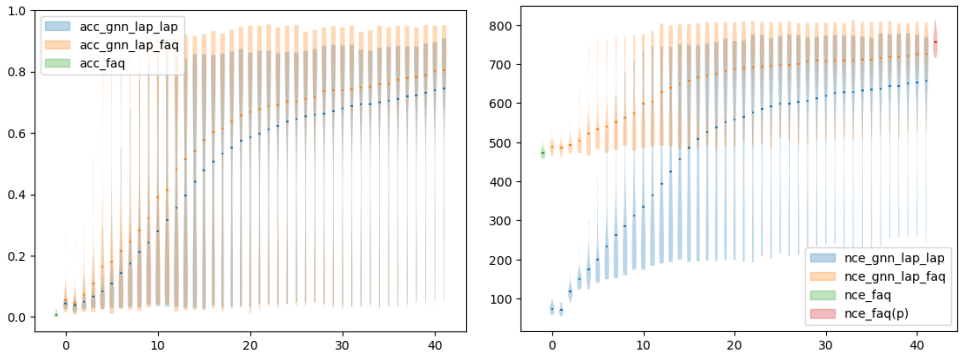


Figure 10: Sparse Erdős-Rényi graphs: **acc** (top) and **nce** (bottom) as a function of the number of iterations L at inference.

Table 20: Accuracy (**acc**) defined in (3) for sparse Erdős-Rényi graphs as a function of the noise p_{noise}. FGNN refers to the architecture in Section A.6 and ChFGNN to our chained FGNNs. **Proj** and **FAQ** are used to produce a permutation (from the similarity matrix computed).

ER 4 (**acc**)	NOISE	0	0.05	0.1	0.15	0.2	0.25	0.3	0.35
BASELINES	**PROJ**(D_{CX})	98.0	97.3	90.3	59.3	23.3	9.1	4.0	2.0
	FAQ(D_{CX})	98.0	97.5	96.3	94.6	72.9	13.0	3.7	1.7
TRAINING 0.05	CHFGNN **PROJ**	97.9	97.3	94.3	67.1	12.5	6.72	3.70	2.13
	CHFGNN **FAQ**	97.9	97.5	96.2	72.9	43.5	10.8	4.04	1.69
TRAINING 0.10	CHFGNN **PROJ**	97.9	97.5	96.1	91.6	35.4	7.24	4.03	2.36
	CHFGNN **FAQ**	97.9	97.5	96.4	94.0	38.6	12.3	4.73	1.91
TRAINING 0.15	CHFGNN **PROJ**	97.9	97.5	96.3	93.7	87.4	49.1	7.51	2.28
	CHFGNN **FAQ**	97.9	97.5	96.4	94.3	90.3	54.7	9.71	1.79
TRAINING 0.20	CHFGNN **PROJ**	97.9	97.4	96.3	94.5	91.4	72.0	31.2	3.30
	CHFGNN **FAQ**	97.9	97.5	96.4	95.2	93.1	76.3	35.0	3.39
TRAINING 0.22	CHFGNN **PROJ**	97.9	97.5	96.2	94.4	91.1	78.9	44.5	7.11
	CHFGNN **FAQ**	97.9	97.5	96.4	95.3	93.1	82.1	48.3	7.78
TRAINING 0.24	CHFGNN **PROJ**	97.9	97.4	96.1	94.1	91.0	77.0	40.3	6.52
	CHFGNN **FAQ**	97.9	97.5	96.4	95.3	93.3	80.1	43.3	6.93
TRAINING 0.26	CHFGNN **PROJ**	97.9	97.3	95.2	92.3	88.0	75.8	43.1	6.43
	CHFGNN **FAQ**	97.9	97.5	96.4	95.2	93.2	82.5	48.2	6.87
TRAINING 0.28	CHFGNN **PROJ**	97.9	95.2	88.4	79.3	68.4	55.1	26.7	6.69
	CHFGNN **FAQ**	97.9	97.5	96.3	94.9	92.7	82.9	38.7	7.62
TRAINING 0.30	CHFGNN **PROJ**	97.9	91.5	78.0	63.6	50.6	35.4	15.1	4.76
	CHFGNN **FAQ**	97.9	97.4	96.2	94.8	92.1	73.5	23.4	5.00
TRAINING 0.35	CHFGNN **PROJ**	97.6	88.2	65.1	40.8	21.5	10.9	5.46	2.54
	CHFGNN **FAQ**	97.9	97.4	96.2	94.5	68.0	19.4	5.79	2.00

Each line in Tables 20 and 21 corresponds to a chained FGNN trained at a given level of noise (given on the left) and tested for all different noises.

Table 21: Number of common edges (**nce**) defined in (3) for sparse Erdős-Rényi graphs as a function of the noise p_{noise}. FGNN refers to the architecture in Section A.6 and ChFGNN to our chained FGNNs. **Proj** and **FAQ** are used to produce a permutation (from the similarity matrix computed).

ER 4 (**NCE**)	NOISE	0	0.05	0.1	0.15	0.2	0.25	0.3	0.35
BASELINES	**PROJ**(D_{cx})	997	950	853	499	195	130	115	112
	FAQ(D_{cx})	997	950	898	847	723	504	487	485
TRAINING 0.05	CHFGNN **PROJ**	997	950	885	630	116	95	87	83
	CHFGNN **FAQ**	997	950	898	761	607	495	485	481
TRAINING 0.10	CHFGNN **PROJ**	997	950	897	828	370	99	90	86
	CHFGNN **FAQ**	997	950	899	845	606	501	487	483
TRAINING 0.15	CHFGNN **PROJ**	996	950	898	840	768	511	254	86
	CHFGNN **FAQ**	997	950	899	846	791	651	520	484
TRAINING 0.20	CHFGNN **PROJ**	996	950	898	846	792	665	456	338
	CHFGNN **FAQ**	997	950	899	849	800	715	596	529
TRAINING 0.22	CHFGNN **PROJ**	997	950	898	845	790	694	503	319
	CHFGNN **FAQ**	997	950	899	849	800	730	626	534
TRAINING 0.24	CHFGNN **PROJ**	997	950	897	844	789	686	480	296
	CHFGNN **FAQ**	997	950	899	849	800	726	613	527
TRAINING 0.26	CHFGNN **PROJ**	997	949	892	834	770	672	499	338
	CHFGNN **FAQ**	997	950	899	849	800	731	626	537
TRAINING 0.28	CHFGNN **PROJ**	996	934	836	724	612	504	374	311
	CHFGNN **FAQ**	997	950	899	848	799	732	599	530
TRAINING 0.30	CHFGNN **PROJ**	996	897	726	566	446	345	271	246
	CHFGNN **FAQ**	997	950	898	848	797	704	552	513
TRAINING 0.35	CHFGNN **PROJ**	995	860	578	347	219	173	159	134
	CHFGNN **FAQ**	997	950	898	847	702	524	494	489

A.16 Additional results for dense Erdős-Rényi graphs

For the correlated dense Erdős-Rényi graphs, we used the same dataset as in Yu et al. [2023] with 500 nodes and an average degree of 80. Again, with a noise level of 20%, our chaining GNNs clearly outperform the existing learning algorithms, as we obtain a perfect accuracy (as opposed to an accuracy of zero in Yu et al. [2023] and Chen et al. [2020] without any seed). We see in Table 22 that in this dense setting, **FAQ**(D_{cx}) is very competitive but is still slightly outperformed by our chaining FGNNs (orange curve with **Proj** and red curve with **FAQ**, top). In terms of number of common edges, our chained FGNNs does not perform well with **Proj** but performs best with **FAQ**, see Table 22 where the level of noise used for training was 24%.

Figure 11 gives the performance of our chained GNNs trained at noise level 0.24 for dense Erdős-Rényi graphs with average degree $d = 80$ and size $n = 500$.

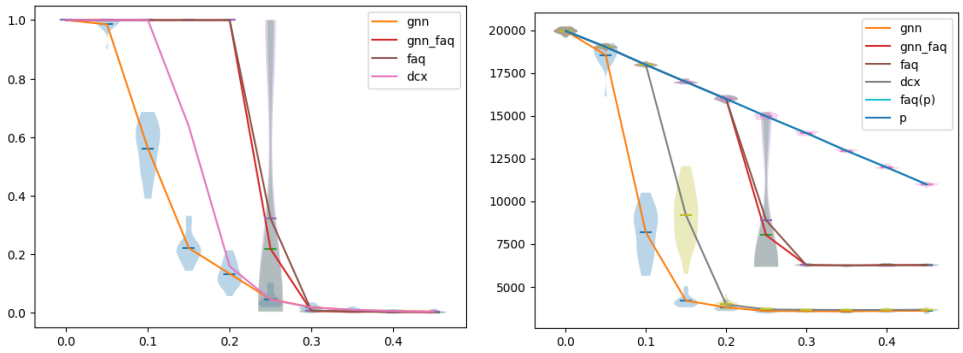


Figure 11: Dense Erdős-Rényi graphs: **acc** (top) and **nce** (bottom) as a function of the noise level. Chained FGNNs were trained at noise level 0.24. **gnn** (resp. **gnn_faq**) for chained FGNNs with **Proj** (resp. **FAQ**) as the last step. **faq** for **FAQ**(D_{cx}) and **faq(p)** for **FAQ**($\pi^\star$).

Table 22: Accuracy (**acc**) defined in (3) for dense Erdős-Rényi graphs as a function of the noise p_{noise}. FGNN refers to the architecture in Section A.6 and ChFGNN to our chained FGNNs. **Proj** and **FAQ** are used to produce a permutation (from the similarity matrix computed).

ER 80 (ACC)	NOISE	0	0.05	0.1	0.15	0.2	0.25	0.3	0.35
BASELINES	**PROJ**(D_{CX})	100.	100.	100.	60.8	14.3	4.3	1.8	1.1
	FAQ(D_{CX})	100.	100.	100.	100.	100.	21.2	0.9	0.5
TRAINING 0.05	CHFGNN **PROJ**	100.	100.	99.8	16.0	5.95	2.76	1.77	1.01
	CHFGNN **FAQ**	100.	100.	100.	100.	54.4	1.16	0.72	0.47
TRAINING 0.10	CHFGNN **PROJ**	100.	100.	100.	88.9	7.49	3.47	1.99	1.16
	CHFGNN **FAQ**	100.	100.	100.	89.0	80.0	6.58	0.85	0.52
TRAINING 0.15	CHFGNN **PROJ**	100.	100.	99.9	99.9	75.0	3.57	2.14	1.21
	CHFGNN **FAQ**	100.	100.	100.	100.	75.1	3.85	0.85	0.55
TRAINING 0.20	CHFGNN **PROJ**	100.	100.	100.	99.9	94.0	22.9	2.06	1.22
	CHFGNN **FAQ**	100.	100.	100.	100.	95.0	22.7	0.83	0.52
TRAINING 0.22	CHFGNN **PROJ**	100.	100.	100.	99.9	97.9	49.5	2.21	1.25
	CHFGNN **FAQ**	100.	100.	100.	100.	99.0	50.5	1.00	0.52
TRAINING 0.24	CHFGNN **PROJ**	100.	99.9	93.5	83.4	67.4	34.0	2.16	1.33
	CHFGNN **FAQ**	100.	100.	100.	100.	98.1	57.6	0.96	0.55
TRAINING 0.26	CHFGNN **PROJ**	100.	99.9	78.3	39.4	13.0	3.91	2.07	1.29
	CHFGNN **FAQ**	100.	100.	100.	100.	94.1	5.75	0.82	0.54
TRAINING 0.28	CHFGNN **PROJ**	100.	99.8	70.7	31.2	9.88	3.94	2.01	1.19
	CHFGNN **FAQ**	100.	100.	100.	100.	84.7	12.7	0.83	0.51
TRAINING 0.30	CHFGNN **PROJ**	100.	99.5	62.3	24.2	8.27	3.28	1.92	1.18
	CHFGNN **FAQ**	100.	100.	100.	100.	80.7	3.24	0.81	0.49
TRAINING 0.35	CHFGNN **PROJ**	100.	96.1	47.8	18.3	6.86	3.47	1.92	1.14
	CHFGNN **FAQ**	100.	100.	100.	100.	69.7	8.52	0.77	0.52

Each line in Tables 22 and 23 corresponds to a chained FGNN trained at a given level of noise (given on the left) and tested for all different noises.

Table 23: Number of common edges (**nce**) defined in (3) for dense Erdős-Rényi graphs as a function of the noise p_{noise}. FGNN refers to the architecture in Section A.6 and ChFGNN to our chained FGNNs. **Proj** and **FAQ** are used to produce a permutation (from the similarity matrix computed).

ER 80 (NCE)	NOISE	0	0.05	0.1	0.15	0.2	0.25	0.3	0.35
BASELINES	**PROJ**(D_{CX})	19964	18987	17966	8700	3888	3646	3633	3624
	FAQ(D_{CX})	19964	18987	17968	16990	15972	7922	6272	6276
TRAINING 0.05	CHFGNN **PROJ**	19964	18987	17941	3794	3457	3421	3429	3411
	CHFGNN **FAQ**	19964	18987	17968	16990	11408	6244	6252	6253
TRAINING 0.10	CHFGNN **PROJ**	19964	18987	17968	15479	3522	3459	3453	3449
	CHFGNN **FAQ**	19964	18987	17968	15811	13935	6681	6251	6257
TRAINING 0.15	CHFGNN **PROJ**	19964	18987	17967	16989	12842	3470	3469	3456
	CHFGNN **FAQ**	19964	18987	17968	16990	13544	6421	6254	6256
TRAINING 0.20	CHFGNN **PROJ**	19964	18987	17968	16990	15216	6113	3483	3472
	CHFGNN **FAQ**	19964	18987	17968	16990	15487	8172	6258	6254
TRAINING 0.22	CHFGNN **PROJ**	19964	18987	17968	16987	15701	9189	3644	3628
	CHFGNN **FAQ**	19964	18987	17968	16990	15876	10614	6257	6263
TRAINING 0.24	CHFGNN **PROJ**	19964	18969	16241	13028	9561	6166	3615	3591
	CHFGNN **FAQ**	19964	18987	17968	16990	15779	11227	6258	6255
TRAINING 0.26	CHFGNN **PROJ**	19964	18976	12528	5795	3925	3626	3591	3545
	CHFGNN **FAQ**	19964	18987	17968	16990	15388	6515	6257	6257
TRAINING 0.28	CHFGNN **PROJ**	19964	18948	10846	4975	3756	3587	3542	3512
	CHFGNN **FAQ**	19964	18987	17968	16990	14424	7207	6253	6256
TRAINING 0.30	CHFGNN **PROJ**	19964	18861	9289	4419	3651	3489	3478	3472
	CHFGNN **FAQ**	19964	18987	17968	16990	14032	6354	6254	6258
TRAINING 0.35	CHFGNN **PROJ**	19964	17850	6943	4003	3578	3512	3492	3461
	CHFGNN **FAQ**	19964	18987	17968	16990	12877	6853	6254	6256

A.17 Additional results for regular graphs

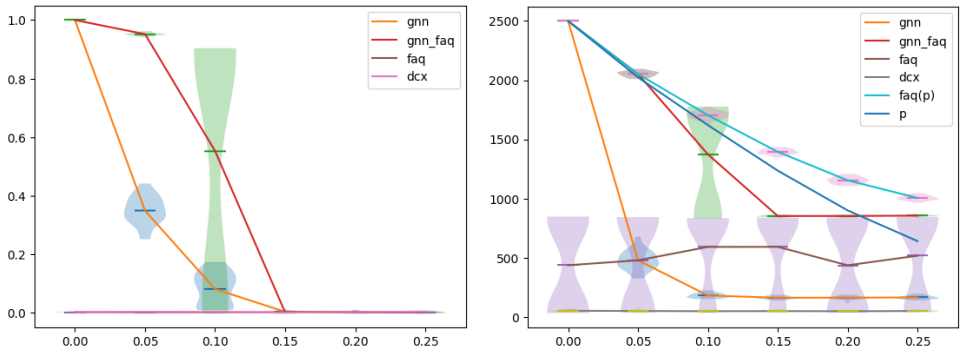


Figure 12: Regular graphs: **acc** (top) and **nce** (bottom) as a function of the noise level. Chained FGNNs were trained at noise level 0.1. **gnn** (resp. **gnn_faq**) for chained FGNNs with **Proj** (resp. **FAQ**) as the last step. **faq** for **FAQ**(D_{CX}), **faq(p)** for **FAQ**$(\pi^\star)$, and **p** for **nce**$(\pi^\star)$.

Finally, we propose a new dataset of regular graphs with 500 nodes and an average degree of 10. This is a particularly challenging setting. Indeed, Table 24 shows that **FAQ**(D_{CX}) always fails to solve the graph matching problem here. The obstruction is not specific to regularity: it is the same obstruction captured by fractional isomorphism and by the 1-WL ceiling of MPNNs Xu et al. [2018].

Theorem A.10. *Tinhofer [1991] G_A and G_B are fractionally isomorphic, i.e.* $\min_{D\in\mathcal{D}_n} \|AD - DB\|_F^2 = 0$*, if and only if 1-WL does not distinguish G_A and G_B.*

Proposition A.11 (MPNN ceiling under fractional isomorphism)**.** *Consider a Siamese MPNN encoder with shared parameters and identical initial node features on two fractionally isomorphic graphs G_A and G_B. Then the final node embeddings are constant on the common stable 1-WL colour classes of the two graphs. Consequently any similarity matrix $S_{ij} = s(h_i^A, h_j^B)$ computed only from these embeddings is block-constant on pairs of 1-WL colour classes. A **Proj** decoder, or a **FAQ** initialization based only on S, can therefore use at most the common 1-WL partition and has no graph-dependent signal for distinguishing vertices inside a colour class.*

Proof. Run 1-WL on both graphs from the same constant initial colour. By Theorem A.10, fractional isomorphism is equivalent to 1-WL indistinguishability, so the two graphs have the same stable colour histogram and the same common 1-WL colour classes up to relabelling of colours. The standard MPNN-versus-1-WL induction says that, for any fixed MPNN parameters, two vertices with the same 1-WL colour after t refinement rounds have the same MPNN embedding after t message-passing layers. Hence the final embeddings are constant on each stable colour class, with corresponding classes in G_A and G_B receiving the same embedding. Any pairwise score $s(h_i^A, h_j^B)$ therefore depends only on the colour classes of i and j, not on their identities inside those classes. The assignment objective built from S cannot distinguish permutations that only rearrange vertices within a common colour class. □

For regular graphs with constant features, the stable 1-WL partition has a single colour class, so Proposition A.11 reduces to a constant similarity matrix: an unchained MPNN gives no more informative initialization than the barycenter $D_{\text{cx}} = J$. The proposition does not rule out rank-augmented chained MPNNs, because the rank vector deliberately injects non-constant positional information after the first decoding step; empirically, however, our regular-graph experiments indicate that the 2-FWL FGNN backbone is the effective choice. In contrast, our FGNN architecture defined in Section A.6 is able to deal with regular graphs and our chaining procedure learns the correct information about the graph matching problem when the noise is low enough.

Note that we are in a setting where $\textbf{FAQ}(\pi^\star) \neq \pi^*$ as soon as the noise level is above 5% so that $\pi^\star \neq \pi^{A\to B}$. In this case, we believe that $\pi^{A\to B} = \textbf{FAQ}(\pi^\star)$ (but to check it we should solve the graph matching problem!). In Figure 12, the training was done with a noise level of 10% so that labels were noisy. Still performances of our chained FGNNs with **FAQ** are very good. We do not know of any other algorithm working in this setting.

Table 24: Accuracy (**acc**) defined in (3) for Regular graphs as a function of the noise p_{noise}. FGNN refers to the architecture in Section A.6 and ChFGNN to our chained FGNNs. **Proj** and **FAQ** are used to produce a permutation (from the similarity matrix computed).

REGULAR RANDOM GRAPHS WITH DEGREE 10

REGULAR (**ACC**)	NOISE	0	0.05	0.1	0.15	0.2
BASELINES	**PROJ**(D_{CX})	0.2	0.2	0.3	0.1	0.2
	FAQ(D_{CX})	0.2	0.2	0.2	0.2	0.2
TRAINING 0.05	CHFGNN **PROJ**	100.	95.2	2.60	0.67	0.27
	CHFGNN **FAQ**	100.	95.6	8.31	0.49	0.24
TRAINING 0.07	CHFGNN **PROJ**	100.	95.3	34.6	0.70	0.27
	CHFGNN **FAQ**	100.	95.6	36.0	0.54	0.25
TRAINING 0.09	CHFGNN **PROJ**	100.	95.2	54.4	0.86	0.34
	CHFGNN **FAQ**	100.	95.6	55.6	0.78	0.22
TRAINING 0.11	CHFGNN **PROJ**	100.	72.4	30.5	0.86	0.27
	CHFGNN **FAQ**	100.	95.6	61.8	0.70	0.25
TRAINING 0.13	CHFGNN **PROJ**	79.2	16.9	2.13	0.55	0.25
	CHFGNN **FAQ**	100.	95.6	2.14	0.37	0.24
TRAINING 0.15	CHFGNN **PROJ**	60.4	13.3	1.69	0.52	0.30
	CHFGNN **FAQ**	100.	95.6	1.37	0.34	0.21

Each line in Tables 24 and 25 corresponds to a chained FGNN trained at a given level of noise (given on the left) and tested for all different noises.

Table 25: Number of common edges (**nce**) defined in (3) for Regular graphs as a function of the noise p_{noise}. FGNN refers to the architecture in Section A.6 and ChFGNN to our chained FGNNs. **Proj** and **FAQ** are used to produce a permutation (from the similarity matrix computed).

REGULAR RANDOM GRAPHS WITH DEGREE 10						
REGULAR (**NCE**)	NOISE	0	0.05	0.1	0.15	0.2
BASELINES	**PROJ**(D_{CX})	51	51	50	49	50
	FAQ(D_{CX})	385	425	456	369	496
TRAINING 0.05	CHFGNN **PROJ**	2500	2034	178	101	100
	CHFGNN **FAQ**	2500	2059	901	835	835
TRAINING 0.07	CHFGNN **PROJ**	2500	2036	741	103	172
	CHFGNN **FAQ**	2500	2059	1193	836	852
TRAINING 0.09	CHFGNN **PROJ**	2500	2034	1105	281	95
	CHFGNN **FAQ**	2500	2059	1381	871	836
TRAINING 0.11	CHFGNN **PROJ**	2500	1343	563	192	114
	CHFGNN **FAQ**	2500	2059	1438	850	837
TRAINING 0.13	CHFGNN **PROJ**	1608	210	108	88	71
	CHFGNN **FAQ**	2500	2059	841	836	834
TRAINING 0.15	CHFGNN **PROJ**	984	163	96	86	87
	CHFGNN **FAQ**	2500	2059	837	836	836

A.18 LLM Usage

Large language models (LLMs) were employed in this work to assist with grammatical and syntactic corrections, to improve the clarity and readability of sentences and paragraphs, to support the generation of illustrative figures, to help refine one proof, and as coding assistants for the experimental pipeline and the ablation scripts. LLMs did not contribute to the core methodology or originality of the research.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and Section 5 state precisely the regimes in which our method matches, exceeds, or only modestly improves on **FAQ** (with concrete numbers), and Section 6 notes that broader transfer to other CO problems is left to future work. We do not claim to "close" the GNN-vs-classical gap on CO in general; we claim to do so for graph alignment on correlated random graphs. The claims about the mechanism of the ranking feedback are backed by the ablations of Section 5.1 and Appendix A.9.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 6 contains an explicit *Limitations* paragraph: (i) each FGNN layer has $O(n^2 d)$ activation memory, which caps practical training around $n \approx 1000$ on an 80 GB GPU in our implementation, and we do not claim scalability to much larger graphs (Section 5.2); (ii) the training cost grows linearly with the number of links; (iii) real-world gains over a properly initialized **FAQ** are modest and zero-shot transfer is unreliable (Section 5.3); (iv) FGNN inherits the 2-FWL ceiling [Maron et al., 2019] and cannot separate strongly-regular graphs of identical parameters — a theoretical barrier we have not hit empirically. A footnote in Section 3 discloses a default parameter of the implementation found after submission, together with the ablation showing that it does not affect the results.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: Appendix A.7 states the assumptions for the FGNN–PPGN simulation theorem and gives a complete proof. Other theoretical facts used in the paper rely on previously known properties of **FAQ**, 1-WL/fractional isomorphism, and 2-FWL expressivity, with citations where used.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The chaining procedure is described in Section 3, the FGNN architecture in Section 3.3 and Section A.6, training hyperparameters in Section A.8, and the synthetic-data generators in Section A.1. The PyTorch code, the pretrained chains and the per-pair results are publicly available at `https://github.com/mlelarge/chaining-gnn-graph-alignment`.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The PyTorch implementation, the pretrained chains (as GitHub releases) and the per-pair results behind the reproduced tables are publicly available at `https://github.com/mlelarge/chaining-gnn-graph-alignment`, with one `make` target per table. Synthetic datasets are generated by the code itself; the real-world graphs (Yeast PPI [Vijayan and Milenkovic, 2018], ca-netscience [Newman, 2006], inf-euroroad [Šubelj and Bajec, 2011]) are public benchmarks.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section A.8 specifies the optimiser (Adam, learning rate 10^{-4}, ReduceLROnPlateau scheduler with patience 3), architecture (2 residual layers, hidden dimension 256, m_0 and m_1 each a 2-hidden-layer MLP), chain length (L=15), training/validation split (200/100 graphs of size n=500), and number of epochs per chain link.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Variability across evaluation graph pairs is reported as confidence intervals or violin plots in Figures 5, 6, 7, 9, 10, 11, and 12. For readability, the tables (Tables 2–4) report only the mean values over evaluation graph pairs (typically 100 per setting); the corresponding figures show the spread. The released per-pair results allow the computation of confidence intervals for every reproduced cell, and Figure 3 shows the per-pair distributions, which are bimodal at the phase transitions.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section A.8 reports the GPUs used (NVIDIA RTX 8000 48 GB and A100 80 GB) and training schedule (100 epochs per chain link, L=15 chain links, batch sizes set to fit memory at n=500); Section 5.2 separates the one-off training cost (about two GPU-hours for a three-link chain) from the inference cost and memory, and Appendix A.5 reports per-pair runtimes of all baselines.

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work is purely methodological, involves no human subjects, uses only public datasets that we cite, and does not release any model with foreseeable misuse risk.

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Graph alignment has positive applications in computational biology (PPI network comparison), entity resolution across knowledge graphs, and computer vision. We see no direct path from improved structural graph alignment to negative societal impact (no surveillance pipeline, no generative misuse vector); we therefore do not foreground a dedicated impact section.

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: We do not release pretrained generative models, language models, or scraped data. The released code trains small (<1M parameter) FGNNs on synthetic graphs.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: SciPy (BSD-3, used for `linear_sum_assignment` and `quadratic_assignment`) and PyTorch (BSD-3) are cited where used. The three real-world datasets are public benchmarks with the original references [Vijayan and Milenkovic, 2018, Newman, 2006, Šubelj and Bajec, 2011] given in Section 5.3 and Section A.3.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The public repository includes a README documenting how to reproduce every table (synthetic and real-world experiments, baselines and ablations), the pretrained chains and the per-pair results.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or research with human subjects.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve human subjects.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: LLMs were used for grammatical/syntactic editing, to help draft illustrative figures, to help refine one proof, and as coding assistants for the experimental pipeline, not as part of the methodology. This is disclosed in Section A.18.