Training Neural Networks to Predict Graphs.

Who is afraid of the big bad NP-hardness?

Paul Krzakala

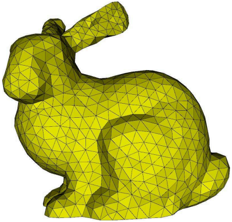
Ecole Polytechnique (CMAP) & Télécom Paris (LTCI).

Introduction

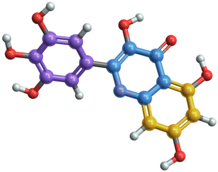
The infamous "Graphs are everywhere" slide

Social networks

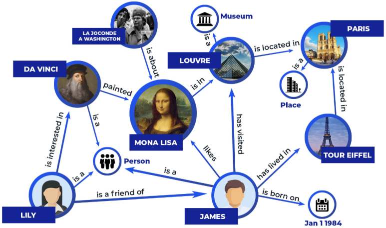
Maps



3D Mesh



Molecules

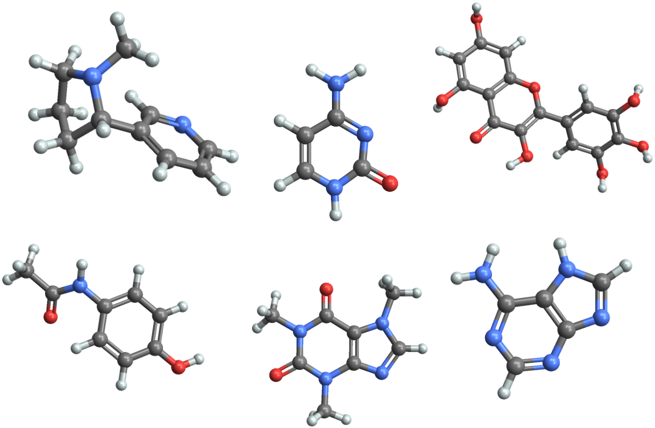


Knowledge Graph

Let's be more precise: the data

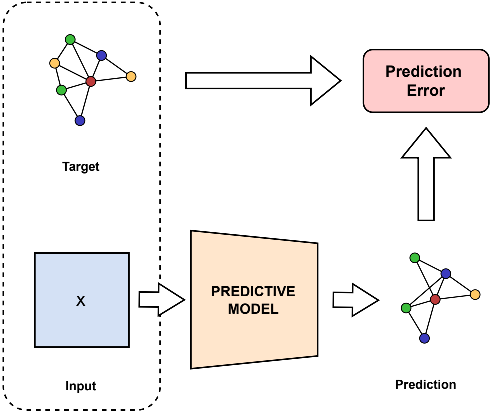
- Dataset is made of **many** graphs (> 10000, can be millions)
- Each graph is **small** (< 100 nodes, typically)

Ex: molecular datasets...



Let's be more precise: the tasks

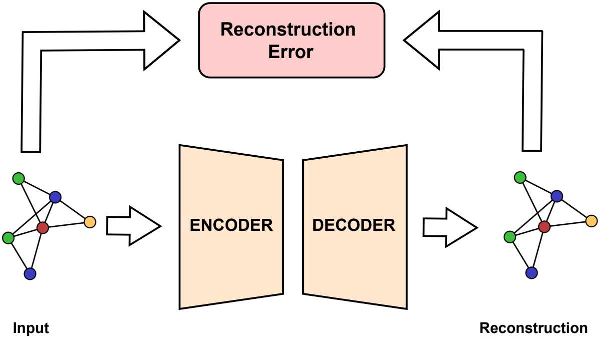
Any task where the **output** is a graph. Ex: **graph prediction** ...



krzakala2024any2graph, **krzakala2024any2graph**, Neurips 2024.

Let's be more precise: the tasks

Any task where the **output** is a graph. Ex: **graph AutoEncoder** ...



krzakala2025quest, **krzakala2025quest**, Preprint 2025.

Challenges

Minor challenge: The size

We need to handle **different sizes** with **fixed model** (and in parallel).

Small graphs : it is easy to pick a **max size** and use **padding** .

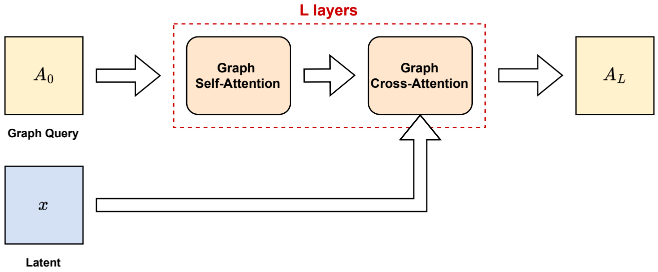
$$\underbrace{\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}}_{A} \iff \underbrace{\begin{pmatrix} 0 & 1 & 0 & \cdot & \cdot \\ 1 & 0 & 1 & \cdot & \cdot \\ 0 & 1 & 0 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}}_{(A,h)} \tag{1}$$

The new vector h indicates which nodes are real.

Minor challenge: architectures

- Many works on graph encoding models $x = f_\theta(A)$
- Few works on graph decoding models $A = f_\theta(x)$
- **Large, unexplored, design space** .

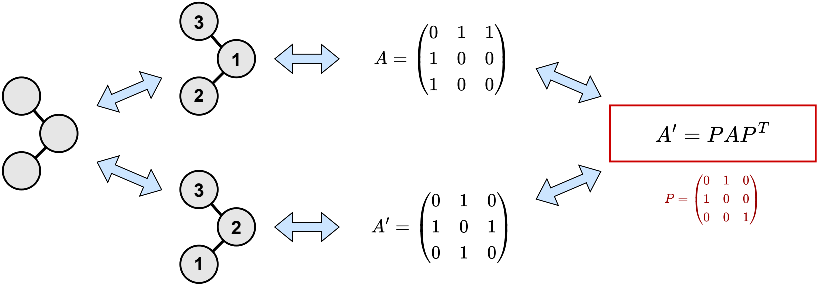
High level idea:



Just need to generalize self/cross attention to graphs.

Major challenge: Permutation Invariance

All models should be **invariant to node reordering** .



In particular, the loss should be invariant:

$$\forall P \in \sigma_n, \quad \mathcal{L}(A, A^*) = \mathcal{L}(A, P[A^*]) \tag{2}$$

where $P[A] = PAP^T$.

Major challenge: Permutation Invariance

Theorem

If $\mathcal{L}(A, A^*)$ *satisfies:*

(i) **Permutation invariance:** $\forall P \in \sigma_n, \quad \mathcal{L}(A, A^*) = \mathcal{L}(A, P[A^*])$,

(ii) **Separability:** $\mathcal{L}(A, A^*) = 0 \implies \exists P \in \sigma_n, \quad A = P[A^*]$,

Then, there exists a base loss $\mathcal{L}_0$ *such that:*

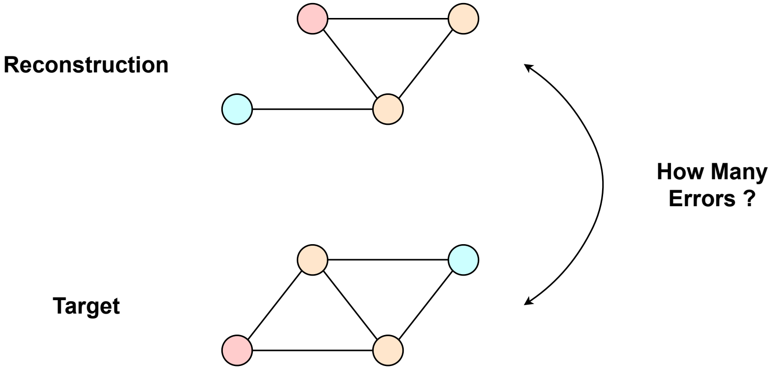
$$\mathcal{L}(A, A^*) = \min_P \mathcal{L}_0(A, P[A^*]) \tag{3}$$

and solving the optimization problem is **NP-hard** .

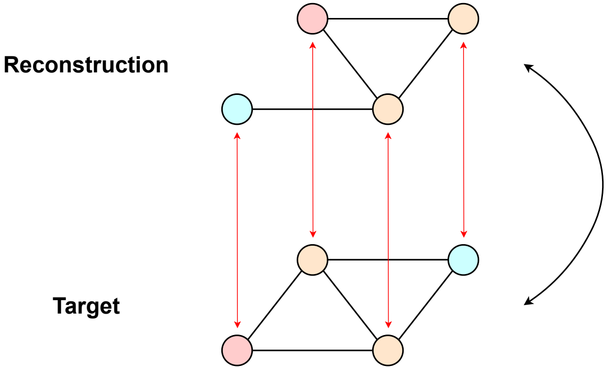
Example: $\mathcal{L}_0(A, A^*) = ||A - A^*||_F^2$.

"Any reasonable loss rewrites as a graph matching problem!"

Major challenge: Permutation Invariance



Major challenge: Permutation Invariance



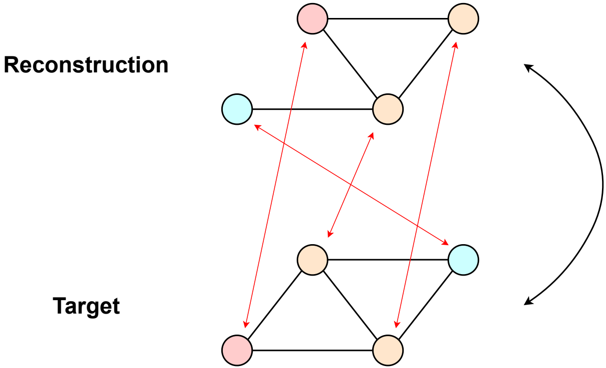
Non-Optimal Matching

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

4 Errors ?

(3 Nodes, 1 Edge)

Major challenge: Permutation Invariance

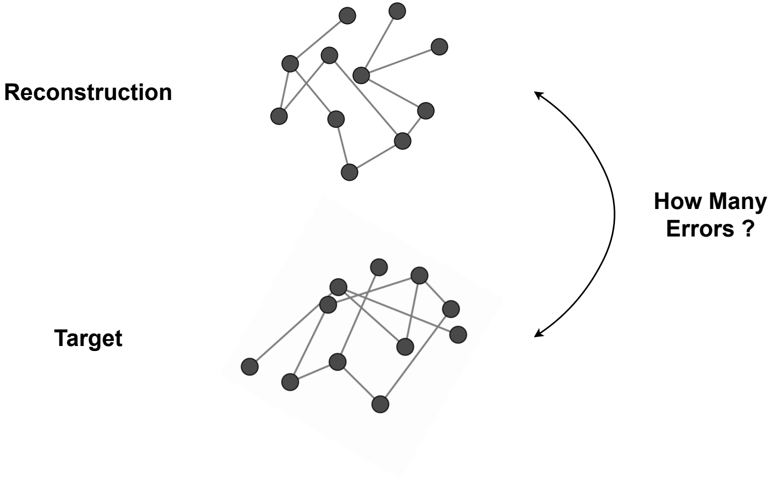


Optimal Matching

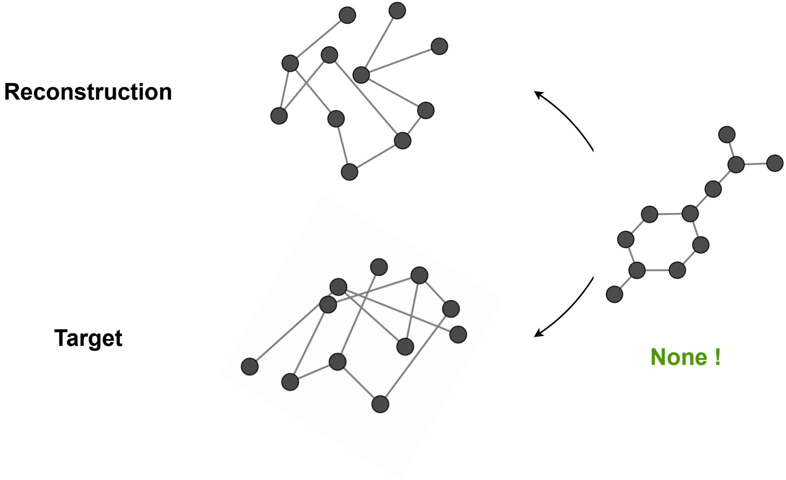
$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

1 Error !

Major challenge: Permutation Invariance



Major challenge: Permutation Invariance



Existing alternatives

Graph Canonization

Main idea:

1. Re-order the nodes in a **canonical manner**
2. Reframe graph prediction as **sequence prediction**

Pros: leverage NLP litterature.

Cons:

- If the ordering is not unique, the training is **noisy** .
- The training is **biased** (model must "retro-engineer" the algorithm).

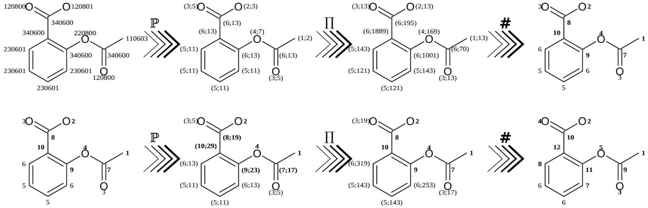


Figure 1: SMILES canonical ordering algorithm.

Generative modeling

In deterministic setting, the **loss** needs to be invariant:

$$\min \mathcal{L}(f_\theta(x), y^*) \quad + \text{ensure that } \mathcal{L} \text{ is invariant} \tag{4}$$

Invariance in a generative model, the **distribution** needs to be invariant:

$$\max \log P_\theta(y^*|x) \quad + \text{ensure that } P_\theta(y|x) \text{ is invariant} \tag{5}$$

Easy to achieve! For instance with a **permutation equivariant** denoiser g_θ:

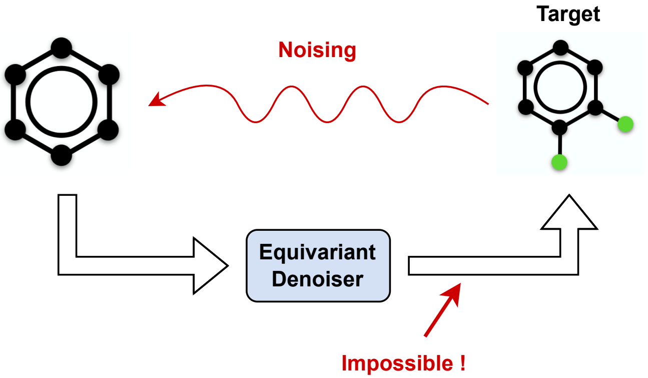
$$Y \sim P_\theta \iff Y = g_\theta(Z),\ Z \sim \text{Unif} \tag{6}$$

Graph Generative modeling is a hot topic. Limitations:

- Inference can be slow
- Can be hard to train
- Symmetry can be a problem

Curie's Principle :

An equivariant function can only make the input "more symmetric".



Lawrence, Hannah, et al. "Improving equivariant networks with probabilistic symmetry breaking."

Node-Level Models

Any model that rely on **local operations** (e.g. GNNs).

Note: this is not always an option (e.g. graph prediction).

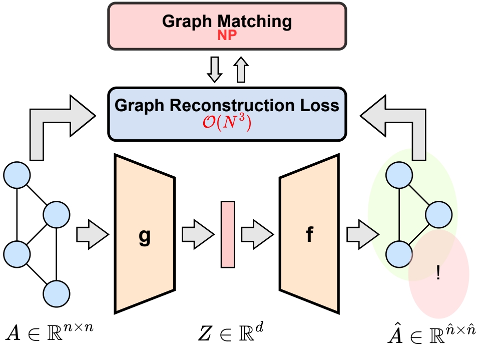


Figure 2: Naive graph-level auto-encoder.

Node-Level Models

Any model that rely on **local operations** (e.g. GNNs).

Note: this is not always an option (e.g. graph prediction).

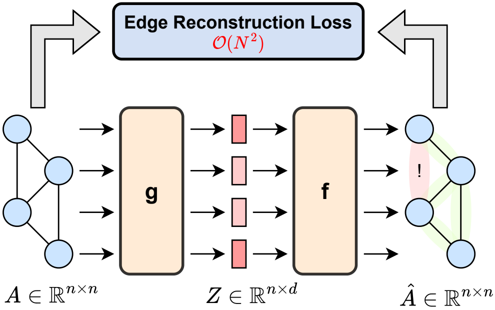


Figure 3: Node-level auto-encoder (+ Aggregation for graph-level embedding).

Direct Approach

Is it reasonable?

Is it reasonable ?

Matching **arbitrary** graphs is NP, BUT:

- Matching **trees** is $O(\log(n))$.
- Matching **planar graphs** is $O(n)$.
- Matching **Interval graphs** is $O(n^2)$.
- Matching **graphs of degree k** is $O(n^k)$.

And many more data distributions!

Is it reasonable ?

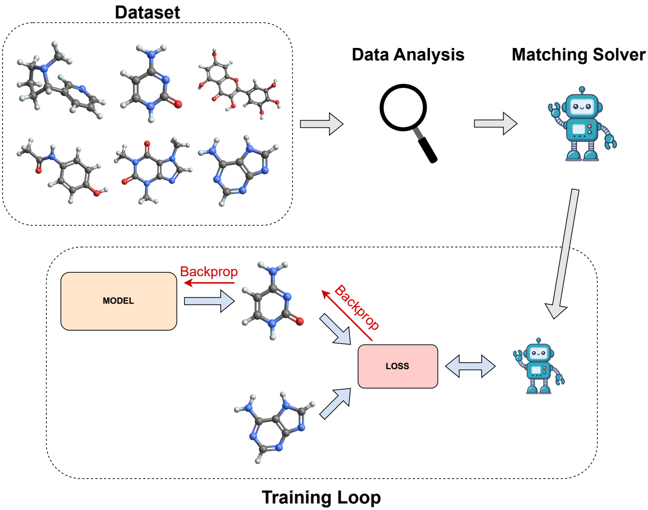
Similarly, **Graph Isomorphism** is NP, but in practice...

FRACTION OF IDENTIFIABLE GRAPHS FOR k WL ITERATIONS

Dataset	Identifiable Graphs		
	k=1	k=2	k=3
DD	100.00	100.00	100.00
ENZYMES	100.00	100.00	100.00
MUTAG	32.32	92.68	96.34
NCI1	94.18	99.47	100.00
NCI109	94.91	99.40	100.00
PROTEINS	100.00	100.00	100.00
COLLAB	100.00	100.00	100.00
IMDB-B	100.00	100.00	100.00
IMDB-M	100.00	100.00	100.00
REDDIT-B	100.00	100.00	100.00
REDDIT-M-5K	100.00	100.00	100.00

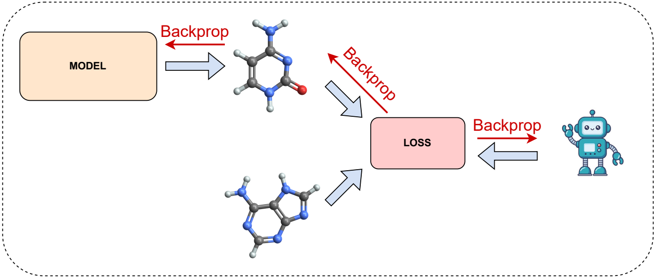
Figure 4: 1-WL Expressiveness Is (Almost) All You Need, Markus Zopf, 2021.

Is it reasonable ?



Any2graph: Deep end-to-end supervised graph prediction with an optimal transport loss, Krzakala et al, Neurips 2024.

Is it reasonable ?



Training Loop

GRALE: The quest for the GRAph Level autoEncoder, Krzakala et al, Preprint 2025.

Relaxation

Graph matching problem:

$$\min_{P \in \sigma_n} L_0(A, A', P) \tag{7}$$

where σ_n is the set of **permutation matrices** :

$$\sigma_n = \{P \in \{0,1\}^{n \times n}, P\mathbf{1} = P^T\mathbf{1} = \mathbf{1}\} \tag{8}$$

We can relax it to the set of **doubly stochastic matrices** (convex hull):

$$\pi_n = \{T \in [0,1]^{n \times n}, T\mathbf{1} = T^T\mathbf{1} = \mathbf{1}\} \tag{9}$$

Ex:

$$\begin{pmatrix} 0 & \mathbf{1} & 0 \\ \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} \end{pmatrix} \in \sigma_3, \quad \begin{pmatrix} 0 & \mathbf{0.9} & 0.1 \\ \mathbf{0.9} & 0.1 & 0 \\ 0.1 & 0 & \mathbf{0.9} \end{pmatrix} \in \pi_3 \tag{10}$$

Choice of the relaxation

For $P \in \sigma_n$ permutation matrix, $L_0(A, A', P) =$

$$||A - PA'P^T||_F^2 = ||AP - PA'||_F^2 = \sum_{i,j,k,l} P_{i,k} P_{j,l} d(A_{i,j}, A'_{k,l})$$

For $T \in \pi_n$ matching matrix:

$$||A - TA'T^T||_F^2 \neq ||AT - TA'||_F^2 \neq \underbrace{\sum_{i,j,k,l} T_{i,k} T_{j,l} d(A_{i,j}, A'_{k,l})}_{\mathcal{L}_{GW}(A,A',T)}$$

How to choose the relaxation?

Theorem

$\mathcal{L}_{GW}$ is the only relaxation such that

$$\mathcal{L}(A, A', T) = 0 \iff \exists P \in \sigma_n, A = PA'P^T \tag{11}$$

For a **loss function** $\mathcal{L}_{GW}$ is the good choice.

Fused Gromov-Wasserstein

Known as **Gromov-Wasserstein** loss in Optimal-Transport [**peyre2016gromov**].

$$\mathcal{L}_{GW}(A, A', T) = \sum_{i,j,k,l} T_{i,k} T_{j,l} d(A_{i,j}, A'_{k,l}) \tag{12}$$

Interpretation: "Map $i \to k$ and $j \to l$ if $A_{i,j} \approx A'_{k,l}$"

The **Fused Gromov-Wasserstein** adds node features $F, F' \in \mathbb{R}^{n \times d}$ [**vayer2020fused**]:

$$\mathcal{L}_{FGW}(G, G', T) = \sum_{i,k} T_{i,k} d(F_i, F'_k) + \sum_{i,j,k,l} T_{i,k} T_{j,l} d(A_{i,j}, A'_{k,l}) \tag{13}$$

Interpretation: "Map $i \to k$ if $F_i \approx F'_k$"

Solver choice

Solver choice

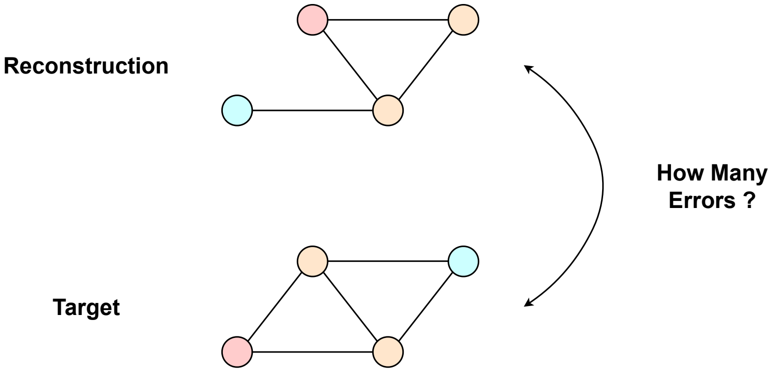
$\min_{T \in \sigma_n} \mathcal{L}_{FGW}(G, G', T)$ is still NP (non-convex QP).

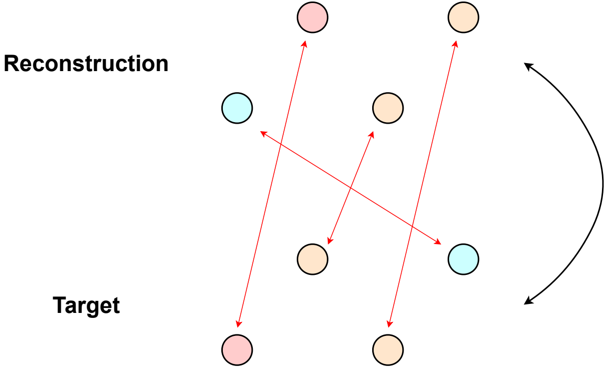
Conditionnal gradient solver: $\mathcal{O}(Kn^3)$ where K number of iterations.

Initialization is important!

Example: use the **optimal node matching** .

$$T_0 = \arg\min \sum_{i,k} T_{i,k} d(F_i, F'_k) + \sum_{i,j,k,l} T_{i,k} T_{j,l} d(A_{i,j}, A'_{k,l}) \tag{14}$$


Reconstruction
Target
How Many
Errors ?

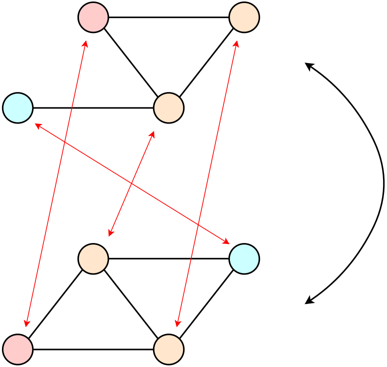


Optimal Matching

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Reconstruction

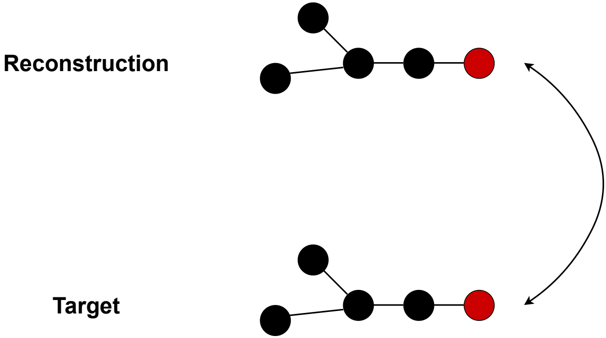
Target



Optimal Matching

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

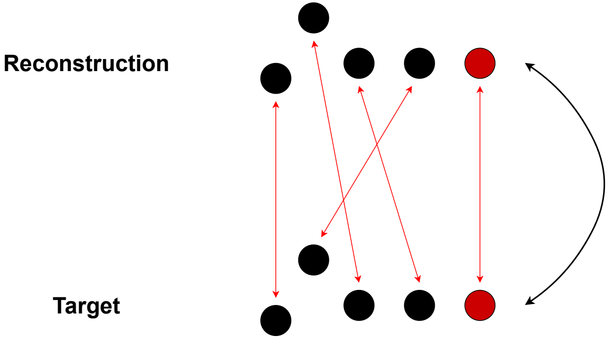
1 Error !



Optimal Matching

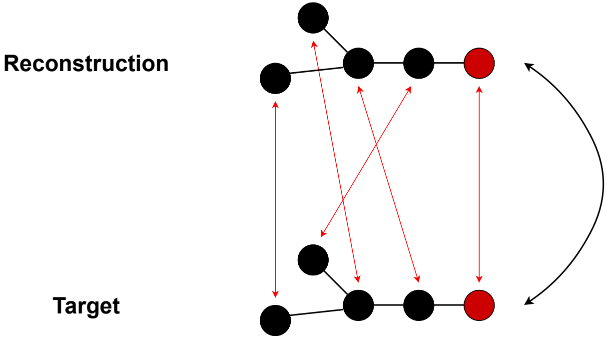
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

0 Errors !



Optimal Matching

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

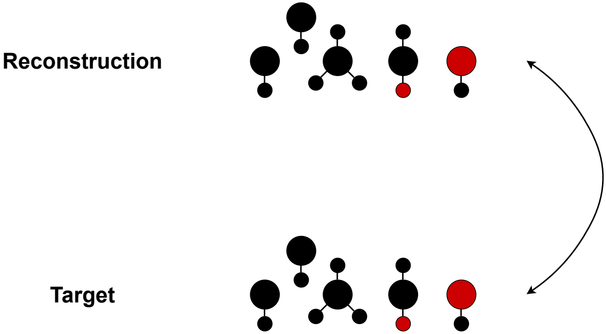


Non-Optimal Matching

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

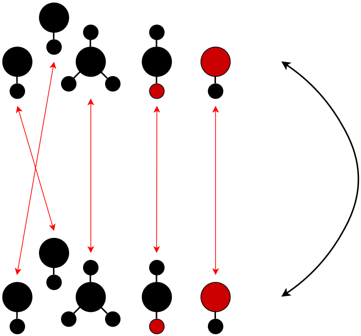
6 Errors ?

(0 Nodes, 6 Edge)


Reconstruction
Target

Reconstruction

Target

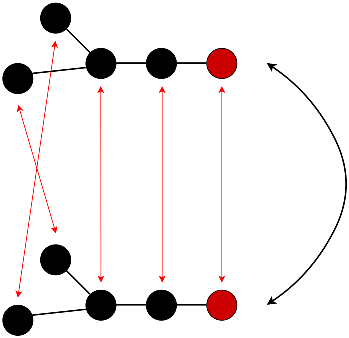


Optimal Matching

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Reconstruction

Target



Optimal Matching

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

0 Errors !

Feature diffusion

In practice: feature augmentation with **message passing** .

$$\tilde{F} = [F, AF] \tag{15}$$

Note: similar to 1-step of Weisfeiler-Lehman test.

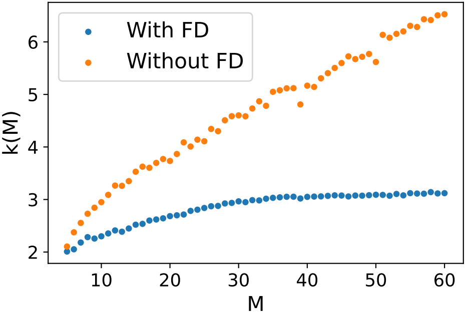
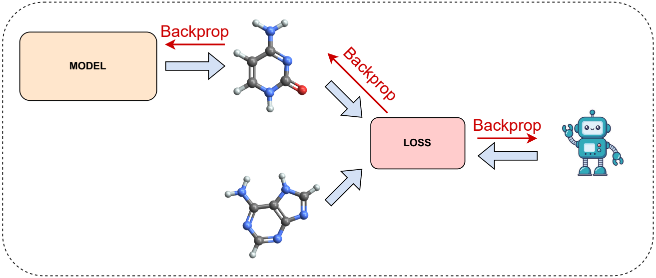


Figure 5: Solver iterations (k) vs average graph size (M).

Solver Learning

Solver Learning



Training Loop

GRALE: The quest for the GRAph Level autoEncoder, Krzakala et al, Preprint 2025.

Main idea

Parametrize the solver:

$$T = M_\theta(A, A') \tag{16}$$

Must be **differentiable** !

Then change the "naive" loss...

$$\min_{T^* \in \pi_n} \mathcal{L}_{FGW}\big(A_\theta(x), A^*, T^*)\big) \tag{17}$$

With the solver-free loss:

$$\mathcal{L}_{FGW}\big(A_\theta(x), A^*, M_\theta(A_\theta(x), A^*)\big) \tag{18}$$

Note: this is an **upper bound** of the original loss.

Main idea

All **existing methods** that train a solver M_θ are **supervised**

$$KL\big(M_\theta(A, A')||T^*\big) \quad \text{where} \quad T^* = \underset{T^* \in \pi_n}{\arg\min}\, \mathcal{L}\big(A, A', T^*\big) \tag{19}$$

Instead we train it **end-to-end without supervision** :

$$\mathcal{L}_{FGW}\big(A_\theta(x), A^*, M_\theta(A_\theta(x), A^*)\big) \tag{20}$$

This works because we picked the **"right" relaxation** $\mathcal{L}_{FGW}$.

Parametrizing the solver

" $M_\theta(A, A')$ = **Feature extraction** + **Node Matching** "

$$M_\theta(A, A') = \text{Sinkhorn}(F_\theta(A), F_\theta(A')) \tag{21}$$

where

$$\text{Sinkhorn}(F, F') = \arg\min_{T \in \pi_n} \sum_{i,k} T_{i,k} d(F_i, F'_k) + \epsilon H(T) \tag{22}$$

Sinkhorn is **differentiable** .

Conclusion

GRALE

GRAph Level autoEncoder (GRALE)

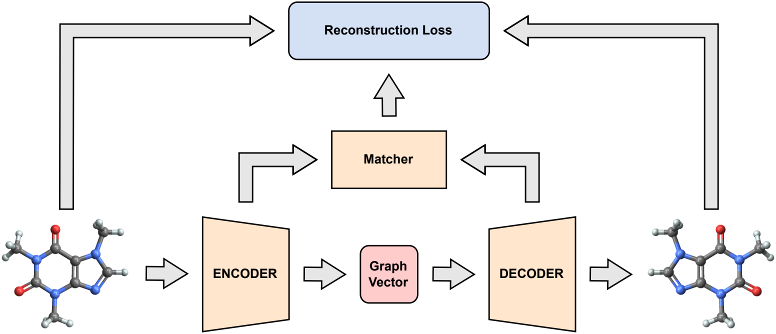
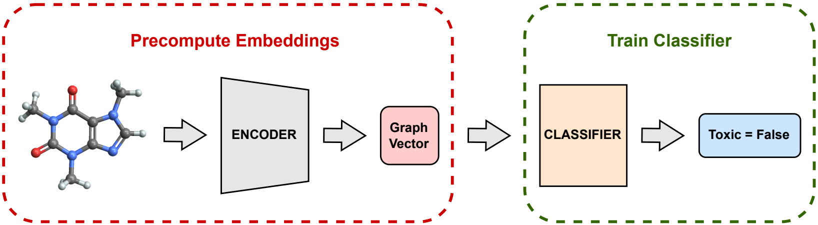


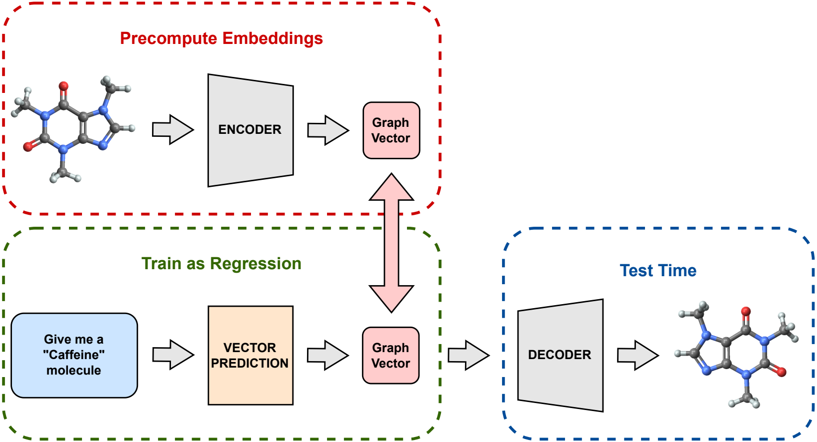
Figure 6: The quest for the GRAph Level autoEncoder (GRALE), Krzakala et al, Preprint 2025.

Application 1: Graph Classification



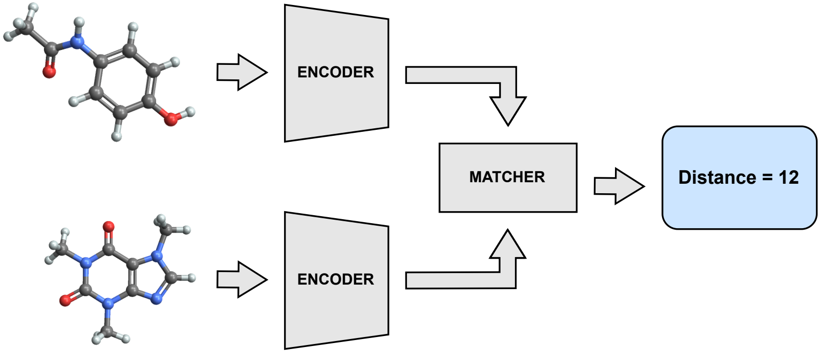
Takeway: Outperforms **node-level AutoEncoder + Aggregation** .

Application 2: Graph Prediction



Takeway: **SOTA** , often by a large margin.

Application 3: Graph Matching



Takeway: find **better matchings** than existing solvers, and **faster** .

See Mazelet et al. [**mazelet2025unsupervised**]

Application 4: Graph Interpolation

Classical **Fréchet Mean is intractable** .

$$A_t = \arg\min_A \; t d(A, A_1) + (1-t) d(A, A_0) \tag{23}$$

Lightspeed interpolation in the **latent space** :

$$A_t = f\Big(t g(A_1) + (1-t) g(A_0)\Big) \tag{24}$$

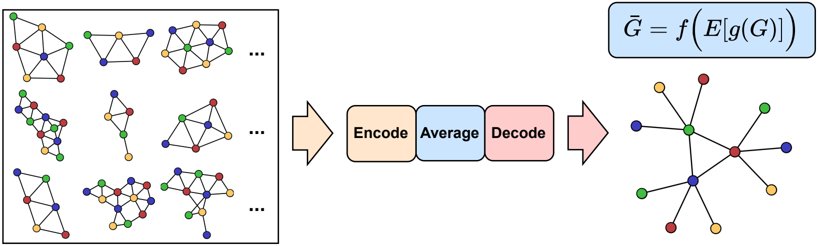


Figure 7: Compute the "average" of 10,000 graphs in seconds with GRALE.

Click here for the animation

Final remarks

Direct approach or alternatives?

Recall the existing alternatives:

i) Graph **Canonization** + Sequence modeling

→ Very **strong baseline** (leverages NLP literature)

ii) **Node-level** models

→ Most **data-efficient**, but not optimal for graph-level tasks

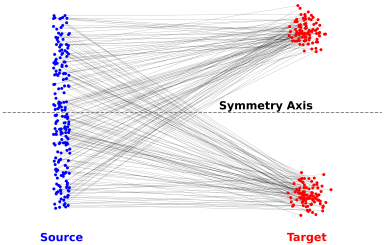
iii) **Generative** modeling

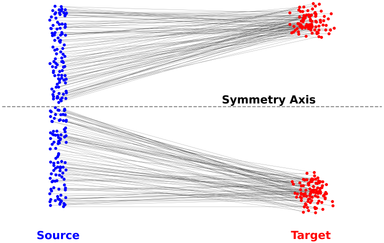
→ **Promising**, hot topic with open questions

The matcher learning is a **new tool in the toolbox**.

Graph matching in Flow models

In presence of **symmetry** naive interpolation paths are not straight.





$$X_t = (1-t)X_0 + tX_1$$

$$X_t = (1-t)X_0 + t(g \cdot X_1)$$

$$g = \arg\min_{g \in G} ||X_0 - g \cdot X_1||$$

For graphs: graph matching problem!

Thank you for your attention!

Looking for a **postdoc** !

Any2Graph Paper

GRALE Paper

References i