A framework for differentiable Supervised Graph Prediction

Paul Krzakala [1 2] **Junjie Yang** [1] **Rémi Flamary** [2] **Florence d'Alché-Buc** [1] **Charlotte Laclau** [1] **Matthieu Labeau** [1]

Abstract

We introduce a general framework to train a deep neural network to output a graph from a variety of input modalities. The framework is built using a novel Optimal Transport loss that exhibits all necessary properties (permutation invariance and differentiability) and allows for handling graphs of any size. We showcase the versatility and state-of-the-art performances of the proposed approach on various real-world tasks and a novel challenging synthetic dataset.

1. Introduction

This work addresses Supervised Graph Prediction (SGP), a supervised task where the output of the predictive model is a graph. In contrast to supervised tasks involving graphs as input that benefit from the hegemony of Graph Neural Networks, there is no broadly accepted framework to address SGP with deep learning. This can be explained by the discrete nature of the output set which makes it challenging to parameterize with a neural network. Consequently, most existing methods rely on domain-specific heuristics, such as introducing an ad hoc ordering of the nodes, to simplify the task [1] [2] [3]. Another line of work is to leverage energy-based models [4] or surrogate regression methods [5] to circumvent the difficulty of directly predicting a graph. However, these approaches typically suffer from an expensive decoding step at inference time. To the best of our knowledge, the graph barycenter of Brogat-Motte et al. [6] and the Relationformer model [7] are the only approaches that can tackle a variety of SGP tasks in a fully end-to-end manner. Unfortunately, those methods suffer from some limitations which we discuss in detail in section 3.

Contributions In this paper, we introduce Any2Graph, a general framework for end-to-end SGP. Any2Graph leverages a novel, fully differentiable, OT-based loss. The framework is completed with a novel synthetic dataset suited to the evaluation of SGP methods. We demonstrate that our approach is versatile and achieves state-of-the-art performance for diverse real-world tasks, such as constructing maps from satellite images (Sat2Graph) or predicting molecules from fingerprints (Fingerprint2Graph).

[1]Télécom Paris, IP Paris, LTCI [2]Ecole polytechnique, IP Paris, CMAP UMR 7641. Correspondence to: Krzakala Paul <paul.krzakala@telecom-paris.fr>.

Published at the 2^{nd} Differentiable Almost Everything Workshop at the 41^{st} International Conference on Machine Learning, Vienna, Austria. July 2024. Copyright 2024 by the author(s).

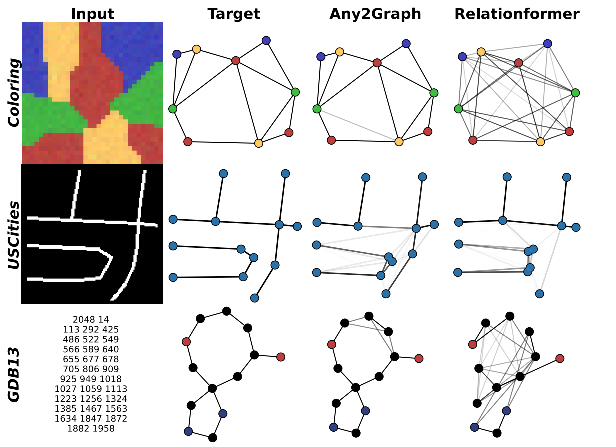

Figure 1: A sample of predictions from Any2Graph (our model) and its direct competitor Relationformer for different SGP tasks. Any2Graph can adapt to a variety of input modalities and output graphs. We extend Relationformer so that it can process inputs that are not images, but its loss limits its performances when the nodes are not uniquely identified by their features (e.g. for molecules). We provide more qualitative results in appendix G.

Notations. An attributed graph g with m nodes can be represented by a tuple $(\mathbf{F}, \mathbf{A})$ where $\mathbf{F} \in \mathbb{R}^{m\times d}$ encodes node features and $\mathbf{A} \in \mathbb{R}^{m\times m}$ is the (symmetric) adjacency matrix. Further, we denote $\mathcal{G}_m$ the set of attributed graphs of m nodes and $\mathcal{G} = \bigcup_{m=1}^{M} \mathcal{G}_m$, the set of attributed graphs of size up to M. In the following, $\mathbf{1}_m \in \mathbb{R}^m$ is the all one vector and we denote $\sigma_m = \{\mathbf{P} \in \{0,1\}^{m\times m} \mid \mathbf{P}\mathbf{1}_m = \mathbf{1}_m, \mathbf{P}^T\mathbf{1}_m = \mathbf{1}_m\}$ the set of permutation matrices. Two graphs $g_1 = (\mathbf{F}_1, \mathbf{A}_1), g_2 = (\mathbf{F}_2, \mathbf{A}_2) \in \mathcal{G}_m$ are said to be isomorphic whenever there exists $\mathbf{P} \in \sigma_m$ such that $(\mathbf{F}_1, \mathbf{A}_1) = (\mathbf{P}\mathbf{F}_2, \mathbf{P}\mathbf{A}_2\mathbf{P}^T)$, in which case we denote $g_1 \sim g_2$. In this work, we consider all graphs to be unordered, meaning that all operations should be invariant by Graph Isomorphism (GI).

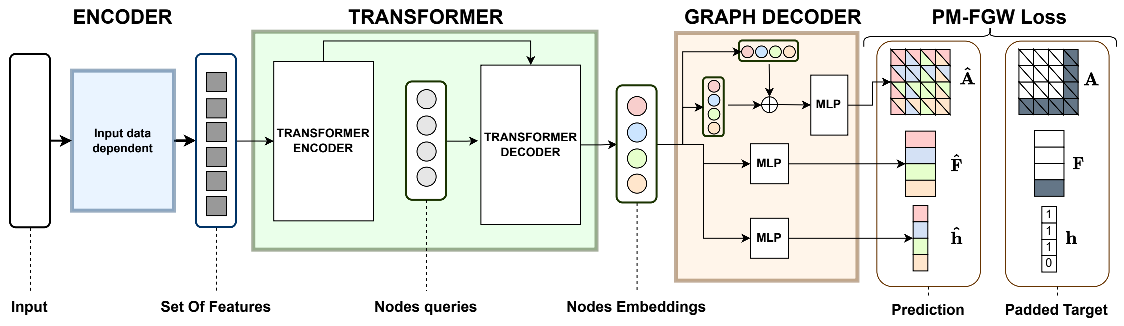

Figure 2: Illustration of the architecture for a target graph of size 3 and M = 4

2. An Optimal Transport loss for SGP

2.1. Graph matching and optimal transport.

Designing a loss function to compare graphs is a challenging task. Even for two graphs of the same size $\hat{g} = (\hat{\mathbf{F}}, \hat{\mathbf{A}})$, $g = (\mathbf{F}, \mathbf{A})$, one cannot simply compute a point-wise comparison as it would not satisfy GI invariance. One of the solutions is to solve a Graph Matching (GM) problem i.e. find a one-to-one matching between the nodes of the graphs before computing the pairwise errors between matched nodes/edges. Formally, the graph matching discrepancy is defined as $\mathrm{GM}(\hat{g}, g) = \min_{\mathbf{P} \in \sigma_m} E(\hat{g}, g, \mathbf{P})$ where the matching cost $E(\hat{g}, g, \mathbf{P})$ is

$$\sum_{i,j=1}^{m} \mathbf{P}_{i,j} \ell_F(\hat{\mathbf{f}}_i, \mathbf{f}_j) + \sum_{i,j,k,l=1}^{m} \mathbf{P}_{i,j} \mathbf{P}_{k,l} \ell_A(\hat{A}_{i,k}, A_{j,l}) \quad (1)$$

The minimization problem however is a Quadratic Assignment Problem (QAP) which is known to be one of the most difficult problems in the NP-Hard class [8]. To mitigate this computational complexity, it has been suggested to replace the space of permutation matrices with a convex relaxation [9]. The Birkhoff polytope (doubly stochastic matrices) $\pi_m = \{\mathbf{T} \in [0,1]^{m \times m} \mid \mathbf{T}\mathbb{1}_m = \mathbb{1}_m, \mathbf{T}^T \mathbb{1}_m = \mathbb{1}_m\}$ is the tightest of those relaxations as it is exactly the convex hull of σ_m which makes it a suitable choice [10]. Interestingly, the resulting metric $\mathrm{FGW}(\hat{g}, g) = \min_{\mathbf{T} \in \pi_m} E(\hat{g}, g, \mathbf{T})$ is known in the Optimal Transport (OT) [11] field as a special case of the (Fused) Gromov-Wasserstein (FGW) distance proposed by Mémoli [12]. FGW is differentiable, GI invariant and efficient solvers are available which make it an ideal candidate for SGP [13]. Unfortunately, this formulation is limited to graphs of the same size, and existing variations are not suited for SGP as discussed in appendix A.1.

2.2. A size-agnostic representation for graphs

Our first step toward building an End-To-End SGP framework is to introduce a space $\hat{\mathcal{Y}}$ that can be used to represent any graph of size up to M. We define $\hat{\mathcal{Y}}$ as:

$$\{(\mathbf{h}, \mathbf{F}, \mathbf{A}) \mid \mathbf{h} \in [0,1]^M, \mathbf{F} \in \mathbb{R}^{M \times d}, \mathbf{A} \in [0,1]^{M \times M}\}. \quad (2)$$

We refer to the elements of $\hat{\mathcal{Y}}$ as 'continuous' graphs, in opposition with 'discrete' graphs of $\mathcal{G}$. Here h_i (resp. $A_{i,j}$) should be interpreted as the probability of the existence of node i (resp. edge $[i,j]$). Any graph $g = (\mathbf{F}_m, \mathbf{A}_m) \in \mathcal{G}_m$ can be embedded into $\hat{\mathcal{Y}}$ with a Padding operator $\mathcal{P}$ that adds $m' = M - m$ dummy nodes to g

$$\mathcal{P}(g) = \left(\begin{pmatrix} \mathbf{1}_m \\ \mathbf{0}_{m'} \end{pmatrix}, \begin{pmatrix} \mathbf{F}_m \\ \mathbf{0}_{m'} \end{pmatrix}, \begin{pmatrix} \mathbf{A}_m & \mathbf{0}_{m'} \\ \mathbf{0}_{m'}^T & \mathbf{0}_{m',m'} \end{pmatrix} \right). \quad (3)$$

We denote $\mathcal{Y} = \mathcal{P}(\mathcal{G}) \subset \hat{\mathcal{Y}}$ the space of padded graphs. For any padded graph in $\mathcal{Y}$, the padding operator can be inverted to recover a discrete graph $\mathcal{P}^{-1} : \mathcal{Y} \mapsto \mathcal{G}$. Besides, any continuous graph $\hat{y} \in \hat{\mathcal{Y}}$ can be projected back to padded graphs $\mathcal{Y}$ by a threshold operator $\mathcal{T} : \hat{\mathcal{Y}} \mapsto \mathcal{Y}$. Note that $\hat{\mathcal{Y}}$ is **convex** and of **fixed dimension** which makes it ideal for parametrization with a neural network. Hence, the core idea of our work is to use a neural network to make a prediction $\hat{y} \in \hat{\mathcal{Y}}$ and to compare it to some target $g \in \mathcal{G}$ through some loss $\ell(\hat{y}, \mathcal{P}(g))$. This calls for the design of an asymmetric loss $\ell : \hat{\mathcal{Y}} \times \mathcal{Y} \mapsto \mathbb{R}_+$.

2.3. An Asymmetric loss for SGP

The Partially Masked Fused Gromov-Wasserstein (PM-FGW) is a loss between a padded target graph $\mathcal{P}(g) = (\mathbf{h}, \mathbf{F}, \mathbf{A}) \in \mathcal{Y}$ with real size $m = \|\mathbf{h}\|_1 \leq M$ and a continuous prediction $\hat{y} = (\hat{\mathbf{h}}, \hat{\mathbf{F}}, \hat{\mathbf{A}}) \in \hat{\mathcal{Y}}$, we define $\mathrm{PMFGW}(\hat{y}, \mathcal{P}(g))$ as:

$$\begin{aligned} \min_{\mathbf{T} \in \pi_M} \ & \frac{\alpha_{\mathrm{h}}}{M} \sum_{i,j} T_{i,j} \ell_h(\hat{h}_i, h_j) \\ & + \frac{\alpha_{\mathrm{f}}}{m} \sum_{i,j} T_{i,j} \ell_f(\hat{\mathbf{f}}_i, \mathbf{f}_j) h_j \\ & + \frac{\alpha_{\mathrm{A}}}{m^2} \sum_{i,j,k,l} T_{i,j} T_{k,l} \ell_A(\hat{A}_{i,k}, A_{j,l}) h_j h_l \end{aligned} \quad (4)$$

Table 1: Graph level, edge level, and node level metrics reported on test for the different models and datasets. * denotes methods that use the actual size of the graph at inference time, hence the performance reported is a non-realistic upper-bound.

Dataset	Model	Graph Level			Edge Level		Node Level Acc.	
		Edit Dist. ↓	GI Acc. ↑	PMFGW ↓	Prec. ↑	Rec. ↑	Node ↑	Size ↑
Coloring	FGWBary-ILE*	7.60	0.90	0.93	72.17	83.81	79.15	N.A.
	Relationformer	5.47	18.14	0.32	80.39	86.34	92.68	99.32
	Any2Graph	**0.33**	**84.44**	**0.03**	**98.63**	**98.87**	**99.99**	**99.85**
Toulouse	FGWBary-ILE*	9.00	0.00	1.21	72.52	56.30	1.62	N.A.
	Relationformer	**0.13**	93.28	**0.02**	99.25	99.24	99.25	98.30
	Any2Graph	**0.13**	**93.62**	**0.02**	**99.34**	**99.26**	**99.39**	**98.81**
USCities	Relationformer	2.09	55.00	0.13	**92.96**	87.98	95.18	**79.80**
	Any2Graph	**1.86**	**58.10**	**0.12**	92.91	**90.85**	**95.70**	78.95
QM9	FGWBary-ILE*	2.84	28.95	0.28	82.96	79.76	92.99	N.A.
	Relationformer	3.80	9.95	0.22	86.07	73.31	99.34	**96.0**
	Any2Graph	**2.13**	**29.85**	**0.14**	**90.19**	**88.08**	**99.77**	95.45
GDB13	Relationformer	8.83	0.01	0.29	84.14	55.89	97.57	**98.65**
	Any2Graph	**3.63**	**16.25**	**0.11**	**90.83**	**84.86**	**99.80**	98.15

The loss takes simultaneously into account each property of the graph. More precisely, the first term ensures that the padding of a node is well predicted. In particular, this requires the model to predict correctly the number of nodes in the target graph. The second term ensures that the features of all non-padding nodes ($h_i = 1$) are well predicted. Similarly, the last term ensures that the relationships between pairs of non-padding nodes ($h_i = h_j = 1$) are well predicted. The normalization in front of the sums ensures that each term is a weighted average of its internal losses as $\sum T_{i,j} = M$, $\sum T_{i,j}h_j = m$ and $\sum T_{i,j}T_{k,l}h_j h_l = m^2$. Finally $\boldsymbol{\alpha} = [\alpha_{\text{h}}, \alpha_{\text{f}}, \alpha_{\text{A}}] \in \Delta_3$ is a triplet of hyperparameters on the simplex balancing the relative scale of the different terms. For ℓ_A and ℓ_h we use the cross-entropy between the predicted value after a sigmoid and the actual binary value in the target. This is equivalent to a logistic regression loss after the OT plan has matched the nodes. For ℓ_f we use the squared ℓ_2 or the cross-entropy loss when the node features are continuous or discrete, respectively. A key feature of this loss is its flexibility. Not only any ground losses can be considered but it is also straightforward to introduce richer spectral representations of the graph [14]. For instance, we discuss the benefits of leveraging a diffused version of the nodes features $\mathbf{F}$ in appendix D.2. Finally, PMFGW translates all the good properties of FGW to the new size-agnostic representation:

Proposition 1 (Complexity)**.** *Each step of the inner optimization can be performed in* $\mathcal{O}(M^3)$.

Proposition 2 (GI Invariance)**.** *If* $\hat{y} \sim \hat{y}'$ *and* $g \sim g'$ *then* PMFGW$(\hat{y}, \mathcal{P}(g))$ = PMFGW$(\hat{y}', \mathcal{P}(g'))$.

Proposition 3 (Positivity)**.** PMFGW$(\hat{y}, \mathcal{P}(g)) \geq 0$ *with equality if and only if* $\hat{y} \sim \mathcal{P}(g)$.

We discuss PMFGW in greater detail in appendix A. The proofs of the propositions are provided in Appendix C.

2.4. Any2Graph framework

We now wrap up the tools introduced above to build Any2Graph, a general framework for SGP.

Formally, the goal of SGP is to learn a function $f : \mathcal{X} \to \mathcal{G}$ using the training samples $\{(x_i, g_i)\}_{i=1}^n \in (\mathcal{X} \times \mathcal{G})^n$. In Any2Graph, we relax the output space and learn a function $\hat{f} : \mathcal{X} \to \hat{\mathcal{Y}}$ that predicts a continuous graph $\hat{y} := f(x)$ as defined in the previous section. Assuming $\hat{f}$ is a parametric model (in this work, a deep neural network) completely determined by a parameter θ, the Any2Graph objective writes as the following empirical risk minimization problem:

$$\min_{\theta} \quad \frac{1}{n}\sum_{i=1}^{n} \text{PMFGW}(\hat{f}_\theta(x_i), \mathcal{P}(g_i)) \tag{5}$$

At inference time, we recover a discrete prediction by a straightforward decoding $f(x) = \mathcal{P}^{-1} \circ \mathcal{T}(\hat{y})$. Where $\mathcal{T}$ is the thresholding operator with threshold $1/2$ on the edges and nodes and $\mathcal{P}^{-1}$ is the inverse padding defined in the previous section. In other words, the full decoding pipeline $\mathcal{P}^{-1} \circ \mathcal{T}$ removes the nodes i (resp. edges (i, j)) whose predicted probability is smaller than $1/2$ i.e. $\hat{h}_i < 1/2$ (resp. $\hat{A}_{i,j} < 1/2$)). Unlike surrogate regression methods, this decoding step does not induce any computational overhead.

The architecture we propose for $\hat{f}_\theta$ is heavily inspired by that of Relationformer [7] and illustrated in figure 2. A full description is provided in appendix B.1. The main novelty is that we investigate more general encoders to enable graph prediction from data other than images. We discuss how to build encoders adapted to text, images, graphs and vectors in appendix B.2 .

3. Numerical experiments

3.1. Experimental setting

We consider 5 datasets that cover a wide spectrum of different input modalities, graph types and sizes. The 4 real-world datasets *Toulouse*, *USCities*, *QM9* and *GDB13* are completed with *Coloring*, a novel synthetic dataset described in appendix F. We compare Any2Graph to its direct competitor Relationformer [7] and a surrogate regression approach (FGW-Bary) based on FGW barycenters [6]. For a fair comparison, we use the same architecture for Relationformer and Any2Graph and augment both with feature diffusion as discussed in appendix D.2. Given the heterogeneity of the datasets considered, we report task-agnostic metrics such as average edit distance [15] or GI Accuracy (the fraction of graphs that are perfectly predicted). The detailed experimental setting is provided in appendix E.

3.2. Prediction Performances

Table 1 shows the performances of the different methods on the five datasets. First, we observe that Any2Graph achieves state-of-the-art performances for all datasets and graph-level metrics. Relationformer is similar to PMFGW, except its loss relies on a bipartite matching taking only the node features into account. The consequence is that Relationformer performs very close to Any2Graph on the Sat2Graph tasks (*Toulouse* and *USCities*) but much poorly on any task where nodes are not uniquely identified by their features. FGW-Bary relies on a computationally heavy barycenter which in its current implementation cannot scale to the datasets featuring the larger graphs (*GDB13* and *USCities*). The SOTA kernel it leverages is very efficient for *QM9*, despite this, it is still outperformed by Any2Graph given that we use feature diffusion.

3.3. Computational Performances

Table 2 shows the ”speed” of each method both in training and inference. Speed is expressed in terms of the number of graph processes per second. Because of the barycenter computation, FGWBary is several orders of magnitude slower than Any2Graph. Note that Relationformer is faster at training time because it does not require solving a QAP. Overall our proposed approach strikes the best of both worlds by achieving SOTA prediction performances at all levels of the graph, at a very low computational inference cost. All values are computed on NVIDIA V100/Intel Xeon E5-2660.

3.4. Analysing the proposed approach

First, we explore the effect of M (the maximum number of nodes) whose default value is that of the largest graph in the train set. We train our model on *Coloring* for M between 10 (default value) and 25 and report the (test) edit distance. To quantify the effective number of nodes used by the model, we also record the number of active nodes i.e. that are masked less than 99% of the time. Results are reported in Figure 3 (left). Interestingly, we observe that performances are robust w.r.t. the choice of M which can be explained by the number of active nodes reaching a plateau. This suggests that the model automatically learns how many nodes it needs to reach the required expressiveness. Similarly, we check that our model is robust to the choice of parameter $\boldsymbol{\alpha}$ which balances the terms of the loss. To this end, we train our model on Coloring for different values $\boldsymbol{\alpha}$ on the simplex and report the (test) edit distance on Figure 3 (right). We observe that performance is optimal for uniform $\boldsymbol{\alpha}$ and robust to other choices as long as there is not too much weight on the structure loss term (corner $\boldsymbol{\alpha}_A = 1$). We further discuss this property in appendix D.1.

Table 2: Speed of the different methods on *QM9* (in graph per second). Training of FGWBary has a closed-form expression computed at once on CPU.

METHOD	TRAIN. ↑	PRED. ↑
FGWBARY	N.A.	1
RELATIONFORMER	2K	10K
ANY2GRAPH	1K	10K

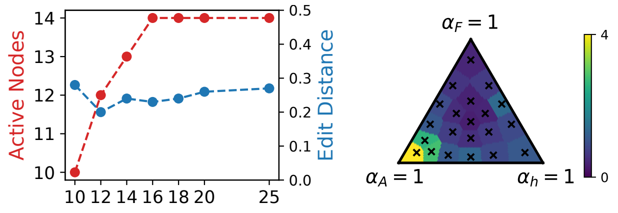

Figure 3: Effect of the choice of M (left) and $\boldsymbol{\alpha}$ (right) on the performances (test edit distance) for *Coloring*.

4. Conclusion and limitations

We present Any2Graph, a novel end-to-end deep learning approach to SGP based on an original asymmetric Partially-Masked Fused Gromov-Wasserstein loss. To the best of our knowledge, it is the first complete and versatile framework to achieve SOTA performance on various graph prediction tasks and input modalities, both in terms of accuracy and computational cost, which makes it a good candidate to be used as a novel baseline in SGP.

The main limitation of Any2Graph is its scalability to large graphs. We envision two approaches to address this issue. First, we plan to explore more general diffusion schemes on the adjacency matrix to capture higher-order interactions that may occur in large graphs. Secondly, we wish to accelerate the OT plan computation with entropic regularization [16] to fully parallelize the solver on a GPU.

References

[1] X. Bresson and T. Laurent, "A two-step graph convolutional decoder for molecule generation," *arXiv preprint arXiv:1906.03412*, 2019.

[2] D. Belli and T. Kipf, "Image-conditioned graph generation for road network extraction," *arXiv preprint arXiv:1910.14388*, 2019.

[3] A. Babu, A. Shrivastava, A. Aghajanyan, A. Aly, A. Fan, and M. Ghazvininejad, "Non-autoregressive semantic parsing for compositional task-oriented dialog," *arXiv preprint arXiv:2104.04923*, 2021.

[4] M. Suhail, A. Mittal, B. Siddiquie, *et al.*, "Energy-based learning for scene graph generation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 13 936–13 945.

[5] C. Brouard, H. Shen, K. Dührkop, F. d'Alché-Buc, S. Böcker, and J. Rousu, "Fast metabolite identification with input output kernel regression," *Bioinformatics*, vol. 32, no. 12, pp. i28–i36, 2016.

[6] L. Brogat-Motte, R. Flamary, C. Brouard, J. Rousu, and F. d'Alché-Buc, "Learning to predict graphs with fused gromov-wasserstein barycenters," in *International Conference on Machine Learning*, PMLR, 2022, pp. 2321–2335.

[7] S. Shit, R. Koner, B. Wittmann, *et al.*, "Relationformer: A unified framework for image-to-graph generation," in *European Conference on Computer Vision*, Springer, 2022, pp. 422–439.

[8] E. M. Loiola, N. M. M. De Abreu, P. O. Boaventura-Netto, P. Hahn, and T. Querido, "A survey for the quadratic assignment problem," *European journal of operational research*, vol. 176, no. 2, pp. 657–690, 2007.

[9] Y. Aflalo, A. Bronstein, and R. Kimmel, "On convex relaxation of graph isomorphism," *Proceedings of the National Academy of Sciences*, vol. 112, no. 10, pp. 2942–2947, 2015.

[10] S. Gold and A. Rangarajan, "A graduated assignment algorithm for graph matching," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 18, no. 4, pp. 377–388, 1996.

[11] C. Villani, *Optimal transport : old and new*. Berlin: Springer, 2009.

[12] F. Mémoli, "Gromov-wasserstein distances and the metric approach to object matching.," *Foundations of Computational Mathematics*, vol. 11, no. 4, pp. 417–487, 2011.

[13] T. Vayer, L. Chapel, R. Flamary, R. Tavenard, and N. Courty, "Fused gromov-wasserstein distance for structured objects," *Algorithms*, vol. 13 (9), p. 212, 2020.

[14] A. Barbe, M. Sebban, P. Gonçalves, P. Borgnat, and R. Gribonval, "Graph diffusion wasserstein distances," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2020, pp. 577–592.

[15] X. Gao, B. Xiao, D. Tao, and X. Li, "A survey of graph edit distance," *Pattern Analysis and applications*, vol. 13, pp. 113–129, 2010.

[16] G. Rioux, Z. Goldfeld, and K. Kato, "Entropic gromov-wasserstein distances: Stability and algorithms," *arXiv preprint arXiv:2306.00182*, 2023.

[17] A. Thual, H. Tran, T. Zemskova, *et al.*, "Aligning individual brains with fused unbalanced gromov-wasserstein," in *Neural Information Processing Systems (NeurIPS)*, 2022.

[18] L. Chapel, M. Z. Alaya, and G. Gasso, "Partial optimal transport with applications on positive-unlabeled learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 2903–2913, 2020.

[19] T. Vayer, L. Chapel, R. Flamary, R. Tavenard, and N. Courty, "Optimal transport for structured data with application on graphs," in *International Conference on Machine Learning (ICML)*, 2019.

[20] M. Thorpe, S. Park, S. Kolouri, G. K. Rohde, and D. Slepčev, "A transportation l^ p l p distance for signal analysis," *Journal of mathematical imaging and vision*, vol. 59, pp. 187–210, 2017.

[21] J. T. Vogelstein, J. M. Conroy, V. Lyzinski, *et al.*, "Fast approximate quadratic programming for large (brain) graph matching," *arXiv preprint arXiv:1112.5507*, 2011.

[22] G. Peyré, M. Cuturi, and J. Solomon, "Gromov-wasserstein averaging of kernel and distance matrices," in *International conference on machine learning*, PMLR, 2016, pp. 2664–2672.

[23] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep sets," *Advances in neural information processing systems*, vol. 30, 2017.

[24] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[25] G. Birkhoff, "Three observations on linear algebra," *Univ. Nac. Tacuman, Rev. Ser. A*, vol. 5, pp. 147–151, 1946.

[26] M. Simonovsky and N. Komodakis, "Graphvae: Towards generation of small graphs using variational autoencoders," in *Artificial Neural Networks and Machine Learning–ICANN*, Springer, 2018, pp. 412–422.

[27] H. De Plaen, P.-F. De Plaen, J. A. Suykens, M. Proesmans, T. Tuytelaars, and L. Van Gool, "Unbalanced optimal transport: A unified framework for object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 3198–3207.

[28] Z. Wu, B. Ramsundar, E. N. Feinberg, *et al.*, "Moleculenet: A benchmark for molecular machine learning," *Chemical science*, vol. 9, no. 2, pp. 513–530, 2018.

[29] L. C. Blum and J.-L. Reymond, "970 million druglike small molecules for virtual screening in the chemical universe database gdb-13," *Journal of the American Chemical Society*, vol. 131, no. 25, pp. 8732–8733, 2009.

[30] U. V. Ucak, I. Ashyrmamatov, and J. Lee, "Reconstruction of lossless molecular representations from fingerprints," *Journal of Cheminformatics*, vol. 15, no. 1, pp. 1–11, 2023.

[31] G. Landrum *et al.*, "Rdkit: A software suite for cheminformatics, computational chemistry, and predictive modeling," *Greg Landrum*, vol. 8, no. 31.10, p. 5281, 2013.

[32] Y. Yang, M. Pilanci, and M. J. Wainwright, "Randomized sketches for kernels: Fast and optimal nonparametric regression," *The Annals of Statistics*, vol. 45, no. 3, pp. 991–1023, Jun. 2017. DOI: `10.1214/16-AOS1472`. [Online]. Available: `https://doi.org/10.1214/16-AOS1472`.

[33] T. El Ahmad, P. Laforgue, and F. d'Alché-Buc, "Fast Kernel Methods for Generic Lipschitz Losses via \p\-Sparsified Sketches," *Transactions on Machine Learning Research*, 2023.

[34] R. Wang, Z. Guo, W. Pan, *et al.*, “Pygmtools: A python graph matching toolkit,” *Journal of Machine Learning Research*, vol. 25, no. 33, pp. 1–7, 2024. [Online]. Available: https://jmlr.org/papers/v25/23-0572.html.

[35] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[36] R. Xiong, Y. Yang, D. He, *et al.*, “On layer normalization in the transformer architecture,” in *International Conference on Machine Learning*, PMLR, 2020, pp. 10 524–10 533.

[37] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.

[38] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[39] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu, “Spatial tessellations: Concepts and applications of voronoi diagrams,” 2009.

[40] R. M. Karp, *Reducibility among combinatorial problems*. Springer, 2010.

A. Additional details about PMFGW

This section is organised as follows. First we discuss existing discrepancy to compare graphs of different sizes and why we believe that they are not suited for SGP A.1. Then, we highlight the relationship between PMFGW and other existing metrics A.2. Next, we describe the solver we use to compute the PMFGW optimal transport plan A.3. Finally, we introduce a toy example illustrating the loss landscape of PMFGW on a toy example A.4

A.1. Comparing graphs of arbitrary size

The GM and FGW described in section 2.1 cannot directly be used to compare graphs of different sizes but several lines of work have been proposed to address this limitation.

The first approach is to fully leverage Optimal Transport by introducing weights on the graph nodes. The Fused-Gromov-Wasserstein distance can then be used to compare graphs of different sizes as long as they have the same total mass [13]. However, this approach raises specific issues. In scenarios where masses are uniform, nodes in larger graphs receive lower mass which might not be suitable for practical applications. Conversely, employing non-uniform masses complicates interpretation, as decoding a discrete object from a weighted one becomes less straightforward. Those issues can be mitigated by leveraging Unbalanced Optimal Transport (UOT) [17], which relaxes marginal constraints, allowing for different total masses in the graphs. Unfortunately, UOT introduces several additional regularization parameters that are difficult to tune, especially in scenarios like SGP, where model predictions exhibit wide variability during training.

Another close line of work is Partial Matching (PM) [18], which consists in matching a small graph g to a subgraph of the larger graph $\hat{g}$. In practice, this can be done by adding dummy nodes to g through some padding operator $\mathcal{P}$ after which one can directly compute PM$(\hat{g}, g) =$ GM$(\hat{g}, \mathcal{P}(g))$ [10]. However, PM is not suited to train a model as the learned model would only predict a graph that **includes** the target graph, with no indication of which subgraph is actually the correct prediction. We discuss the relationship between PMFGW and Partial Matching in the next section.

A.2. Relation to existing metrics

PMFGW is an asymmetric extension of FGW [19] suited for comparing a continuous predicted graph with a padded target. The extension is achieved by adding (1) a novel term to quantify the prediction of node padding, and (2) the partial masking of the components of the second and third terms to reflect padding. It should be noted that in contrast to what is usually done in OT, the node masking vectors ($\mathbf{h}$ and $\hat{\mathbf{h}}$) are not used as a marginal distribution but directly integrated into the loss. In that sense, the additional node masking term is very similar to the one of OTL$_p$ [20] that proposed to use uniform marginal weight and move the part that measures the similarity between the distribution weights in an additional linear term. However, OTL$_p$ is restricted to linear OT problems and does not use the marginal distributions as a masking for other terms as in PMFGW.

PMFGW also relates to Partial GM/GW [18] as both metrics compare graphs by padding the smallest one with zero-cost dummy nodes. The critical difference lies in the new vector $\hat{\mathbf{h}}$ which predicts which sub-graphs are activated, i.e., should be matched to the target. To be precise, PMFGW and Partial Fused Gromov Wasserstein (PFGW) are equal if and only if l_h is set to 0. We prove this statement and provide a formal definition of PFGW in C.3. Note that setting $l_h = 0$ would obviously be undesirable since the vector $\hat{\mathbf{h}}$ would disappears from the loss and the model would not be trained to predict it correctly. In particular, this would prevent the model from learning to predict the size of the target graph.

A.3. PMFGW solver

Computing PMFGW requires solving the inner optimization problem presented in Equation (4) whose objective rewrites

$$\langle \mathbf{T}, \mathbf{U} \rangle + \langle \mathbf{T}, \mathbf{L} \otimes \mathbf{T} \rangle$$

where $\mathbf{U}$ is a fixed matrix, $\mathbf{L}$ a fixed tensor and $\otimes$ the tensor matrix product. A standard way of solving this problem [21] is to use a conditional gradient (CG) algorithm which iteratively solves a linearization of the problem. Each step of the algorithm requires solving a linear OT/Matching problem of cost $\langle \mathbf{T}, \mathbf{C}^{(k)} \rangle$ where the linear cost $\mathbf{C}^{(k)} = \mathbf{U} + \mathbf{L} \otimes \mathbf{T}^{(k)}$ is updated at each iteration. The linear problem can be solved with a Hungarian solver with cost $\mathcal{O}(M^3)$ while the overall complexity of computing the tensor product $\mathbf{L} \otimes \mathbf{T}^{(k)}$ is theoretically $\mathcal{O}(M^4)$. Fortunately, this bottleneck can be avoided thanks to a $\mathcal{O}(M^3)$ factorization (Proposition 1) that extend a result from Peyré et al. [22].

A.4. Illustrating PMFGW On A Toy Example

On the one hand, we consider a target graph of size 2, $\mathbf{g} = (\mathbf{F}_2, \mathbf{A}_2)$ where

$$\mathbf{F}_2 = \begin{pmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \end{pmatrix} ; \mathbf{A}_2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

for some node features $\mathbf{f}_1$ and $\mathbf{f}_2$. For $M = 3$ the padded target is $\mathcal{P}(\mathbf{g}) = (\mathbf{h}, \mathbf{F}, \mathbf{A})$ where

$$\mathbf{h} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} ; \mathbf{F} = \begin{pmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ - \end{pmatrix} ; \mathbf{A} = \begin{pmatrix} 0 & 1 & - \\ 1 & 0 & - \\ - & - & - \end{pmatrix}.$$

On the other hand, we consider a predicted graph $\hat{\mathbf{y}}_{a,h} = (\hat{\mathbf{h}}, \hat{\mathbf{F}}, \hat{\mathbf{A}})$ that has the form

$$\hat{\mathbf{h}} = \begin{pmatrix} 1 \\ h \\ 1-h \end{pmatrix} ; \hat{\mathbf{F}} = \begin{pmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \mathbf{f}_2 \end{pmatrix} ; \hat{\mathbf{A}} = \begin{pmatrix} 0 & a & 1-a \\ a & 0 & 0 \\ 1-a & 0 & 0 \end{pmatrix}$$

for some $a, h \in [0, 1]$. The loss between the prediction and the (padded) target is

$$\mathcal{L}_{\text{train}}(a, h) = \text{PMFGW}(\hat{\mathbf{y}}_{a,h}, \mathcal{P}_3(\mathbf{g}))$$

We are interested in the landscape of this loss. First of all, it appears that $\hat{\mathbf{y}}_{1,1}$ and $\hat{\mathbf{y}}_{0,0}$ and $\mathcal{P}(\mathbf{g})$ are isomorphic, thus we get two global minima $\mathcal{L}_{\text{train}}(1, 1) = \mathcal{L}_{\text{train}}(0, 0) = 0$. Going into greater detail, it can be shown that for $\ell_h(a, b) = \ell_A(a, b) = (a - b)^2$ we have the following expression

$$\mathcal{L}_{\text{train}}(a, h) = \min\left((1-a)^2 + \frac{2}{3}(1-h)^2; a^2 + \frac{2}{3}h^2\right)$$

and the optimal transport plan is the permutation $(1, 2, 3)$ when $(1-a)^2 + \frac{2}{3}(1-h)^2 < a^2 + \frac{2}{3}h^2$ and $(1, 3, 2)$ otherwise. In this toy example, the optimal transport plan is always a permutation.

At inference time, we could similarly be interested in the edit distance between the (discrete) prediction and the target

$$\mathcal{L}_{\text{eval}}(a, h) = \text{ED}(\mathcal{P}_3^{-1}\mathcal{T}(\hat{\mathbf{y}}_{a,h}), \mathbf{g}).$$

Once again, the expression can be computed explicitly

$$\mathcal{L}_{\text{eval}}(a, h) = \mathbb{1}[a < 0.5 \text{ and } h > 0.5] + \mathbb{1}[a > 0.5 \text{ and } h < 0.5]$$

We provide in Figure 4 an illustration of the edit distance and the proposed loss that is clearly a continuous and smoothed version of the edit distance which allows for learning the NN parameters.

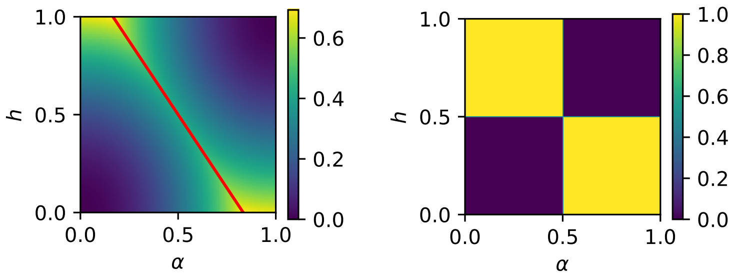

Figure 4: Heatmap of $\mathcal{L}_{\text{train}}$ (left) and $\mathcal{L}_{\text{eval}}$ (right). The red line represents the transition between the regime where the optimal plan is the permutation $(1, 2, 3)$ and that where it is $(1, 3, 2)$. In both cases, the optimal plan is a permutation.

B. Additional details about the architecture

B.1. Overview

The architecture that we use to parameterize $f : \mathcal{X} \to \hat{\mathcal{Y}}$ (left part of Figure 2) is composed of three modules , namely the **encoder** that extracts features from the input, the **transformer** that convert these features into M nodes embeddings, that are expected to capture both feature and structure information, and the **graph decoder** that predicts the properties of our output graph, i.e., $(\hat{\mathbf{h}}, \hat{\mathbf{F}}, \hat{\mathbf{A}})$. As we will discuss later, the proposed architecture draws heavily on that of Relationformer [7] since the latter has been shown to yield to state-of-the-art results on the Image2Graph task.

Encoder The encoder extracts k feature vectors in $\mathbb{R}^{d_e}$ from the input. Note that k is not fixed a priori and can depend on the input (for instance sequence length in case of text input). This is critical for encoding structures as complex as graphs and the subsequent transformer is particularly apt at treating this kind of representation. By properly designing the encoder, we can accommodate different types of input data. We describe how to handle images, text, graphs, and vectors and provide general guidelines to address other input modalities in the next section.

Transformer This module takes as input a set of feature vectors and outputs a fixed number of M node embeddings. This resembles the approach taken in machine translation, and we used an architecture based on a stack of transformer encoder-decoders, akin to [7].

Graph decoder This module decodes a graph from the set of node embeddings $\mathbf{Z} = [\mathbf{z}_1, \dots, \mathbf{z}_M]^T$ using the following equation:

$$\hat{h}_i = \sigma(\mathrm{MLP}_m(\mathbf{z}_i)), \quad \hat{F}_i = \mathrm{MLP}_f(\mathbf{z}_i), \quad \hat{A}_{i,j} = \sigma(\mathrm{MLP}_s^2(\mathrm{MLP}_s^1(\mathbf{z}_i) + \mathrm{MLP}_s^1(\mathbf{z}_j))) \tag{6}$$

where σ is the sigmoid function and MLP_m, MLP_f, MLP_s^k are multi-layer perceptrons heads corresponding to each component of the graph (mask, features, structure). The adjacency matrix is expected to be symmetric which motivate us to parameterize it as suggested by [23].

Positioning with Relationformer As discussed above, the architecture is similar to the one proposed in Relationformer [7], with two modifications: (1) we use a symmetric operation with a sum to compute the adjacency matrix while Relationformer uses a concatenation that is not symmetric; (2) we investigate more general encoders to enable graph prediction from data other than images. However, as stated in the previous section, the main originality of our framework lies in the design of the PMFGW loss. Interestingly Relationformer uses a loss that presents similarities with FGW but where the matching is done on the node features only, before computing a quadratic-linear loss similar to PMFGW. In other words, they solve a bi-level optimization problem, where the plan is computed on only part of the information, leading to potentially suboptimal results on heterophilic graphs as demonstrated in the next section.

B.2. Encoding any input to graphs

Philosophy of the Any2Graph encoder Any2Graph is compatible with different types of inputs, given that one selects the appropriate encoder. The role of the encoder is to extract a set of feature vectors from the inputs x i.e. each input is mapped to a list of k feature vectors of dimension d_e where k is not necessarily fixed. This is critically different from extracting a unique feature vector ($k = 1$). If k is set to 1, the rest of the architecture must reconstruct an entire graph from a single vector, and the architecture is akin to that of an auto-encoder. In Any2Graph, we avoid this undesirable bottleneck by opting for a richer ($k > 1$) and more flexible (k is not fixed) representation of the input. The k feature vectors are then fed to a transformer which is well suited to process sets of different sizes. Since the transformer module is permutation-invariant any meaningful ordering is lost in the process. To alleviate this issue, we add positional encoding to the feature vectors whenever the ordering carries information. Finally, note that the encoder might highly benefit from pre-training whenever applicable; but this goes beyond the scope of this paper.

We now provide a general description of the encoders that can be used for each input modality.

Images For Image2Graph task we use Convolutional Neural Networks (CNN) as suggested in [7]. From an input image of shape $h \times w \times c$ the CNN outputs a tensor of shape $H \times W \times C$ which is seen as $H \times W$ feature vectors of dimension C. The raw output of the CNN is reshaped and passed through a linear layer to produce the final output of shape $H \times W \times d_e$. Since the ordering of the $H \times W$ features carries spatial information we add positional encoding accordingly.

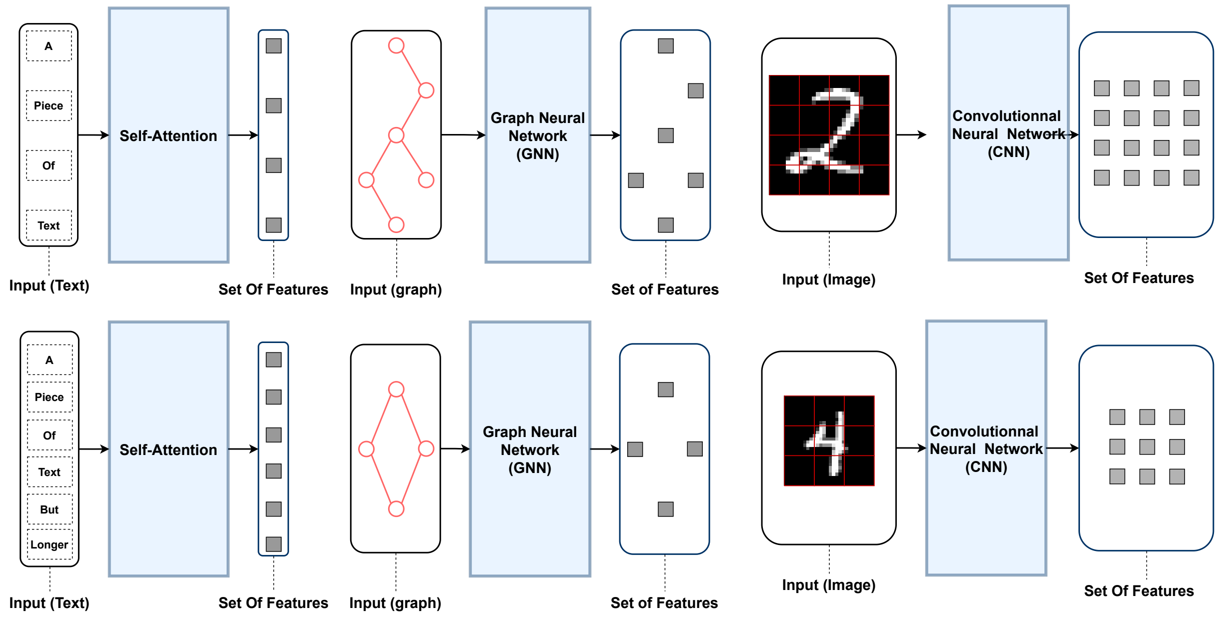

Figure 5: Illustration of encoders extracting k features vectors for different input modalities. For text/fingerprint, k is the number of input tokens. For graphs, k is the size of the input graph. For images, k depends on the resolution of the image and the CNN kernel size.

Fingerprint/text For tasks where the input is a list of tokens (e.g. Text2Graph or Fingerprint2Graph) we use the classical NLP pipeline: each token is transformed into a vector by an embedding layer and the list of vectors is then processed by a transformer encoder module. In text2graph the tokens ordering carries semantic meaning and positionnal encoding should be added. On the contrary, in Fingerprint2Graph, the fingerprint ordering carries no information and the permutation invariance of the transformer module is a welcomed property.

Graph For a graph2graph task (not featured in this paper) we would suggest using a Graph Neural Network (GNN) [24]. A GNN naturally extracts k feature vectors from an input graph, where k is the number of nodes in the input graph. No positional encoding is required.

Vector We explore a Vect2Graph task in Appendix F. The naive encoder we use is composed of k parallel MLPs devoted to the extraction of the k feature vectors. This approach is arguably simplistic and more suited encoders should be considered depending on the type of data.

C. Formal Statements And Proofs

In this section, we write

$$\text{PMFGW}(\hat{y}, \mathcal{P}(g)) = \min_{\mathbf{T}\in\pi_M} \sum_{i,j} T_{i,j}\ell_h(\hat{h}_i, h_j) + \sum_{i,j} T_{i,j}\ell_f(\hat{\mathbf{f}}_i, \mathbf{f}_j)h_j + \sum_{i,j,k,l} T_{i,j}T_{k,l}\ell_A(\hat{A}_{i,k}, A_{j,l})h_j h_l,$$

meaning that we absorb the normalization factors in the ground losses to lighten the notation.

Alternatively, we also consider the matrix formulation:

$$\text{PMFGW}(\hat{y}, \mathcal{P}(g)) = \min_{\mathbf{T}\in\pi_M} \langle \mathbf{T}, \mathbf{C}\rangle + \langle \mathbf{T}, \mathbf{L}\otimes\mathbf{T}\rangle,$$

where $C_{i,j} = \ell_h(\hat{h}_i, h_j) + \ell_f(\hat{\mathbf{f}}_i, \mathbf{f}_j)h_j$, $L_{i,j,k,l} = \ell_A(\hat{A}_{i,k}, A_{j,l})h_j h_l$ and $\otimes$ is the tensor/matrix product.

C.1. PMFGW fast computation

The following results generalize Proposition 1 of [22] so that it can be applied to the computation of PMFGW.

Proposition 4. *Assuming that the ground loss than can decomposed as $\ell(a,b) = f_1(a) + f_2(b) - h_1(a)h_2(b)$, for any transport plan $\mathbf{T} \in \mathbb{R}^{n\times m}$ and matrices $\mathbf{A}, \mathbf{W} \in \mathbb{R}^{n\times n}$ and $\mathbf{A}', \mathbf{W}' \in \mathbb{R}^{m\times m}$, then the tensor product of the form*

$$(\mathbf{L} \otimes \mathbf{T})_{i,i'} = \sum_{j,j'} T_{j,j'} \ell(A_{i,j}, A'_{i',j'}) W_{i,j} W'_{i',j'}$$

can be computed as

$$\mathbf{L} \otimes \mathbf{T} = \mathbf{U}_1 \mathbf{T} \mathbf{W}'^T + \mathbf{W}\mathbf{T}\mathbf{U}_2^T - \mathbf{V}_1 \mathbf{T} \mathbf{V}_2^T,$$

where $\mathbf{U}_1 = f_1(\mathbf{A}) \cdot \mathbf{W}$, $\mathbf{U}_2 = f_2(\mathbf{A}') \cdot \mathbf{W}'$, $\mathbf{V}_1 = h_1(\mathbf{A}) \cdot \mathbf{W}$, $\mathbf{V}_2 = h_2(\mathbf{A}') \cdot \mathbf{W}'$ and $[\cdot]$ is the point-wise multiplication.

Proof. Thanks to the decomposition assumption the tensor product can be decomposed into 3 terms:

$$(\mathbf{L} \otimes \mathbf{T})_{i,i'} = \sum_{j,j'} T_{j,j'} f_1(A_{i,j}) W_{i,j} W'_{i',j'} + \sum_{j,j'} T_{j,j'} f_2(A'_{i',j'}) W_{i,j} W'_{i',j'} - \sum_{j,j'} T_{j,j'} h_1(A_{i,j}) h_2(A'_{i',j'}) W_{i,j} W'_{i',j'}.$$
$$= \sum_j f_1(A_{i,j}) W_{i,j} \sum_{j'} T_{j,j'} W'_{i',j'} + \sum_{j'} f_2(A'_{i',j'}) W'_{i',j'} \sum_j T_{j,j'} W_{i,j} - \sum_j h_1(A_{i,j}) W_{i,j} \sum_{j'} T_{j,j'} h_2(A'_{i',j'}) W'_{i',j'}.$$

Introducing $\mathbf{U}_1, \mathbf{U}_2, \mathbf{V}_1$ and $\mathbf{U}_2$ as defined above, we write:

$$\begin{aligned}(\mathbf{L} \otimes \mathbf{T})_{i,i'} &= \sum_j (U_1)_{i,j} \sum_{j'} T_{j,j'} W'_{i',j'} + \sum_{j'} (U_2)_{i',j'} \sum_j T_{j,j'} W_{i,j} - \sum_j (V_1)_{i,j} \sum_{j'} T_{j,j'} (V_2)_{i',j'}, \\ &= \sum_j (U_1)_{i,j} (TW'^T)_{j,i'} + \sum_{j'} (U_2)_{i',j'} (WT)_{i,j'} - \sum_j (V_1)_{i,j} (T_{j,j'} V_2^T)_{j,i'},\end{aligned}$$

which concludes that $\mathbf{L} \otimes \mathbf{T} = \mathbf{U}_1 \mathbf{T} \mathbf{W}'^T + \mathbf{W}\mathbf{T}\mathbf{U}_2^T - \mathbf{V}_1 \mathbf{T} \mathbf{V}_2^T$. □

Remark 1 (Computational cost). $\mathbf{U}_1, \mathbf{U}_2, \mathbf{V}_1, \mathbf{V}_2$ can be pre-computed for a cost of $\mathcal{O}(n^2 + m^2)$, after which $\mathbf{L} \otimes \mathbf{T}$ can be computed (for any $\mathbf{T}$) at a cost of $\mathcal{O}(mn^2 + nm^2)$.

Remark 2 (Kullback-Leibler divergence decomposition). The Kullback-Leibler divergence $KL(p,q) = q \log \frac{q}{p} + (1 - q) \log \frac{(1-q)}{(1-p)}$, which we use as ground loss in our experiments satisfies the required decomposition given $f_1(p) = -\log(p)$, $f_2(q) = q\log(q) + (1-q)\log(1-q)$, $h_1(p) = \log(\frac{1-p}{p})$ and, $h_2(q) = 1 - q$

Remark 3. The tensor product that appears in PMFGW is a special case of this theorem that corresponds to $n = m = M$, $W_{i,j} = 1$ and $W'_{i',j'} = h_{i'} h_{j'}$. Thus, proposition 1 is a direct corollary.

C.2. PMFGW divergence properties

First, we provide below a more detailed version of Proposition 2

Proposition 5 (GI Invariance)**.** *For any $m \le M$, $\hat{y}, \hat{y}' \in \hat{\mathcal{Y}}$ and $g, g' \in \mathcal{G}_m$, we have that*

$$\hat{y} \sim \hat{y}', g \sim g' \implies \textit{PMFGW}(\hat{y}, \mathcal{P}(g)) = \textit{PMFGW}(\hat{y}', \mathcal{P}(g')).$$

Proof. We denote $\hat{y} = (\hat{\mathbf{h}}, \hat{\mathbf{F}}, \hat{\mathbf{A}})$ and $\mathcal{P}(g) = (\mathbf{h}, \mathbf{F}, \mathbf{A})$. Since $\hat{y}$ and $\hat{y}'$ are isomorphic, there exist a permutation $\mathbf{P} \in \sigma_M$ such that $\hat{y}' = (\mathbf{P}\hat{\mathbf{h}}, \mathbf{P}\hat{\mathbf{F}}, \mathbf{P}\hat{\mathbf{A}}\mathbf{P}^T)$. Moreover, the fact that g and g' are isomorphic implies that $\mathcal{P}(g)$ and $\mathcal{P}(g')$ are isomorphic as well, thus there exist a permutation $\mathbf{Q} \in \sigma_M$ such that $\mathcal{P}(g') = (\mathbf{Q}\mathbf{h}, \mathbf{Q}\mathbf{F}, \mathbf{Q}\mathbf{A}\mathbf{Q}^T)$. Plugging into the PMFGW objective we get

$$\begin{aligned}
\text{PMFGW}(\hat{y}', \mathcal{P}(g')) &= \min_{\mathbf{T}\in\pi_M} \sum_{i,j} T_{i,j}\ell_h((\mathbf{P}\hat{\mathbf{h}})_i, (\mathbf{Q}\mathbf{h})_j) + \sum_{i,j} T_{i,j}\ell_f((\mathbf{P}\hat{\mathbf{F}})_i, (\mathbf{Q}\mathbf{F})_j)(\mathbf{Q}\mathbf{h})_j \\
&\quad + \sum_{i,j,k,l} T_{i,j}T_{k,l}\ell_A((\mathbf{P}\hat{\mathbf{A}}\mathbf{P}^T)_{i,k}, (\mathbf{Q}\mathbf{A}\mathbf{Q}^T)_{j,l})(\mathbf{Q}\mathbf{h})_j(\mathbf{Q}\mathbf{h})_l \\
&= \min_{\mathbf{T}\in\pi_M} \sum_{i,j} (\mathbf{P}^T\mathbf{T}\mathbf{Q})_{i,j}\ell_h(\hat{\mathbf{h}}_i, \mathbf{h}_j) + \sum_{i,j} (\mathbf{P}^T\mathbf{T}\mathbf{Q})_{i,j}\ell_f(\hat{\mathbf{f}}_i, \mathbf{f}_j)h_j \\
&\quad + \sum_{i,j,k,l} (\mathbf{P}^T\mathbf{T}\mathbf{Q})_{i,j}(\mathbf{P}^T\mathbf{T}\mathbf{Q})_{k,l}\ell_A(\hat{A}_{i,k}, A_{j,l})h_jh_l.
\end{aligned}$$

Denoting $\tilde{\mathbf{T}} = \mathbf{P}^T\mathbf{T}\mathbf{Q}$ we have that

$$\begin{aligned}
\text{PMFGW}(\hat{y}', \mathcal{P}(g')) &= \min_{\tilde{\mathbf{T}}\in\pi_M} \sum_{i,j} \tilde{T}_{i,j}\ell_h(\hat{h}_i, h_j) + \sum_{i,j} \tilde{T}_{i,j}\ell_f(\hat{\mathbf{f}}_i, \mathbf{f}_j)h_j \\
&\quad + \sum_{i,j,k,l} \tilde{T}_{i,j}\tilde{T}_{k,l}\ell_A(\hat{A}_{i,k}, A_{j,l})h_jh_l \\
&= \text{PMFGW}(\hat{y}, \mathcal{P}(g)).
\end{aligned}$$

□

We now provide a more detailed version of Proposition 3

Definition 1. *We say that $\ell : \hat{\mathcal{X}} \times \mathcal{X} \mapsto \mathbb{R}$ is positive when for any $x, y \in \hat{\mathcal{X}} \times \mathcal{X}$, $\ell(x, y) \geq 0$ with equality if and only if $x = y$.*

Proposition 6 (Positivity)**.** *Let us assume that $\ell_h : [0, 1] \times \{0, 1\} \mapsto \mathbb{R}, \ell_f : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ and $\ell_A : [0, 1] \times \{0, 1\} \mapsto \mathbb{R}$ are positive. Then we have that for any $\hat{y} \in \hat{\mathcal{Y}}, g \in \mathcal{G}$:*

- *i) PMFGW$(\hat{y}, \mathcal{P}(g)) \geq 0$*
- *ii) There is equality if and only if $\hat{y} \sim \mathcal{P}(g)$*
- *iii) In that case $\mathcal{P}^{-1}\mathcal{T}(\hat{y}) \sim g$*

Proof. The direct implication of ii) is the only statement that is not trivial. First, let us show that if PMFGW$(\hat{y}, \mathcal{P}(g)) = 0$, the optimal transport $\mathbf{T}^*$ is a permutation. Recall that any transport plan is a convex combination of permutations [25] i.e. there exist $\lambda_1, ..., \lambda_K \in]0, 1]$ and $\mathbf{P}_1, ..., \mathbf{P}_K \in \sigma_M$ such that $\sum_{k=1}^K \lambda_k = 1$ and $\mathbf{T}^* = \sum_{k=1}^K \lambda_k\mathbf{P}_k$. Thus

$$0 = \langle \mathbf{T}^*, \mathbf{C}\rangle + \langle \mathbf{T}^*, \mathbf{L} \otimes \mathbf{T}^*\rangle \tag{7}$$

$$= \sum_{k=1}^K \lambda_k\langle \mathbf{P}_k, \mathbf{C}\rangle + \sum_{k=1}^K \lambda_k^2\langle \mathbf{P}_k, \mathbf{L} \otimes \mathbf{P}_k\rangle + \sum_{k\neq l}^K \lambda_k\lambda_l\langle \mathbf{P}_k, \mathbf{L} \otimes \mathbf{P}_l\rangle. \tag{8}$$

This is a sum of positive terms, thus all terms are null and in particular, for any k

$$0 = \langle \mathbf{P}_k, \mathbf{C}\rangle + \langle \mathbf{P}_k, \mathbf{L} \otimes \mathbf{P}_k\rangle. \tag{9}$$

Thus all the $\mathbf{P}_k$ are optimal transport plans. In the following, we chose one of them and denote it $\mathbf{P}$. Moving back to the developed formulation of PMFGW we get that

$$0 = \text{PMFGW}(\hat{y}, \mathcal{P}(g)) = \sum_{i,j} P_{i,j}\ell_h(\hat{h}_i, h_j) + \sum_{i,j} P_{i,j}\ell_f(\hat{\mathbf{f}}_i, \mathbf{f}_j)h_j + \sum_{i,j,k,l} P_{i,j}P_{k,l}\ell_A(\hat{A}_{i,k}, A_{j,l})h_j h_l.$$

Once again this is a sum of positive terms thus for all i, j, k, and l

$$0 = P_{i,j}\ell_h(\hat{h}_i, h_j) = P_{i,j}\ell_f(\hat{\mathbf{f}}_i, \mathbf{f}_j)h_j = P_{i,j}P_{k,l}\ell_A(\hat{A}_{i,k}, A_{j,l})h_j h_l$$

and thus

$$0 = \ell_h((\mathbf{P}^T\hat{\mathbf{h}})_j, h_j) = \ell_f((\mathbf{P}^T\hat{\mathbf{F}})_j, F_j)h_j = \ell_A((\mathbf{P}^T\hat{\mathbf{A}}\mathbf{P})_{j,l}, A_{j,l})h_j h_l.$$

And from the positivity of ℓ_h, ℓ_f and ℓ_A we get that: $\mathbf{P}^T\hat{\mathbf{h}} = \mathbf{h}$, $\mathbf{P}^T\hat{\mathbf{F}}[:m] = \mathbf{F}[:m]$ and $\mathbf{P}^T\hat{\mathbf{A}}\mathbf{P}[:m, :m] = \mathbf{A}[:m, :m]$. Since the nodes $i > m$ are not activated, by abuse of notation we simply write $\mathbf{P}^T\hat{\mathbf{F}} = \mathbf{F}$ and $\mathbf{P}^T\hat{\mathbf{A}}\mathbf{P} = \mathbf{A}$. This concludes that $\hat{y} \sim \mathcal{P}(g)$.

□

C.3. PMFGW and Partial Fused Gromov Wasserstein

Following [18], we define an OT relaxation of the Partial Matching problem.

For a large graph $\hat{g} = (\hat{\mathbf{F}}, \hat{\mathbf{A}}) \in \mathcal{G}_M$ and a smaller graph $g = (\mathbf{F}, \mathbf{A}) \in \mathcal{G}_m$, the set of transport plan transporting a subgraph of $\hat{g}$ to g can be defined as

$$\pi_{M,m} = \{\mathbf{T} \in [0,1]^{M\times m} \mid \mathbf{T}\mathbb{1}_m \leq \mathbb{1}_M, \mathbf{T}^T\mathbb{1}_M = \mathbb{1}_m, \mathbb{1}_M^T\mathbf{T}\mathbb{1}_m = m\}$$

and the associated partial Fused Gromov Wasserstein distance is

$$\text{partialFGW}(\hat{g}, g) = \min_{\mathbf{T}\in\pi_{M,m}} \sum_{i=1}^{M}\sum_{j=1}^{m} \mathbf{T}_{i,j}\ell_F(\hat{\mathbf{f}}_i, \mathbf{f}_j) + \sum_{i,k=1}^{M}\sum_{j,l=1}^{m} \mathbf{T}_{i,j}\mathbf{T}_{k,l}\ell_A(A_{i,k}, A_{j,l}).$$

In the following, we show that partialFGW$(\hat{g}, g)$ is equivalent to the padded Fused Gromov Wasserstein distance defined as

$$\text{paddedFGW}(\hat{g}, g) = \min_{\mathbf{T}\in\pi_{M}} \sum_{i=1}^{M}\sum_{j=1}^{m} \mathbf{T}_{i,j}\ell_F(\hat{\mathbf{f}}_i, \mathbf{f}_j) + \sum_{i,k=1}^{M}\sum_{j,l=1}^{m} \mathbf{T}_{i,j}\mathbf{T}_{k,l}\ell_A(A_{i,k}, A_{j,l}).$$

Lemma 4. Any transport plan $\mathbf{T} \in \pi_M$ has the form $\mathbf{T} = \begin{pmatrix}\mathbf{T}_p & \mathbf{T}_2\end{pmatrix}$ where $\mathbf{T}_p$ is a partial transport plan i.e. $\mathbf{T}_p \in \pi_{M,m}$.

Proof. Let us check that $\mathbf{T}_p$ is in π_M.

- $\mathbb{1}_M = \mathbf{T}\mathbb{1}_M = \mathbf{T}_p\mathbb{1}_m + \mathbf{T}_2\mathbb{1}_{M-m} \geq \mathbf{T}_p\mathbb{1}_m$
- $\mathbb{1}_M = \mathbf{T}^T\mathbb{1}_M = \begin{pmatrix}\mathbf{T}_p^T\mathbb{1}_M \\ \mathbf{T}_2^T\mathbb{1}_M\end{pmatrix}$. And thus $\mathbf{T}_p^T\mathbb{1}_M = \mathbb{1}_m$
- From the previous we immediately get that $\mathbb{1}_M^T\mathbf{T}_p\mathbb{1}_m = m$

□

Lemma 5. For any partial transport plan $\mathbf{T}_p \in \pi_{M,m}$ there exist $T_2 \in \mathbb{R}^{M\times(M-m)}$ such that $\mathbf{T} = \begin{pmatrix}\mathbf{T}_p & \mathbf{T}_2\end{pmatrix} \in \pi_M$.

Proof. Let us define $p = \mathbb{1}_M - \mathbf{T}_p \mathbb{1}_m$. This is the mass of the larger graph that is not matched by $\mathbf{T}_p$. Note that since $\mathbf{T}_p \in \pi_{M,m}$ we have that $p \geq 0$. Thus we can set $\mathbf{T}_2 = \frac{1}{M-m} p \mathbb{1}_{M-m}^T$ i.e. we spread the remaining mass uniformly across the padding nodes. Let us check that $\mathbf{T} = \begin{pmatrix} \mathbf{T}_p & \mathbf{T}_2 \end{pmatrix} \in \pi_M$ is indeed a valid transport plan.

- $\mathbf{T}\mathbb{1}_M = \mathbf{T}_p \mathbb{1}_m + \mathbf{T}_2 \mathbb{1}_{M-m} = \mathbf{T}_p \mathbb{1}_m + p = \mathbb{1}_m$
- $\mathbf{T}^T \mathbb{1}_M = \begin{pmatrix} \mathbf{T}_p^T \mathbb{1}_M \\ \mathbf{T}_2^T \mathbb{1}_M \end{pmatrix} = \begin{pmatrix} \mathbb{1}_m \\ \frac{1}{M-m}(p^T \mathbb{1}_M) \mathbb{1}_{M-m} \end{pmatrix} = \begin{pmatrix} \mathbb{1}_m \\ \frac{1}{M-m}(\mathbb{1}_M^T \mathbb{1}_M - \mathbb{1}_m^T \mathbf{T}_p^T \mathbb{1}_M) \mathbb{1}_{M-m} \end{pmatrix} = \begin{pmatrix} \mathbb{1}_m \\ \mathbb{1}_{M-m} \end{pmatrix} = \mathbb{1}_M$

□

Proposition 7. *paddedFGW and partialFGW are equal and any optimal plan* $\mathbf{T}^*$ *of paddedFGW has the form* $\mathbf{T}^* = (T_p^*, T_2)$ *where* T_p^* *is optimal for partialFGW.*

Proof. Follows directly from the two previous lemmas. □

Remark 6. The proposed PMFGW is equivalent to paddedFGW (and thus to partialFGW) if and only if ℓ_h is set to a constant.

Remark 7. The algorithm proposed to compute PMFGW can be applied to paddedFGW and thus to partialFGW. Hence, we have indirectly introduced an alternative to the algorithm of [18]. Further comparisons are left for future work.

D. Training Any2Graph

D.1. Learning dynamics

The PMFGW loss is composed of three terms, two of them are linear and account for the prediction of the nodes and their features, one is quadratic and accounts for the prediction of edges. The last term is arguably the harder to minimize for the model, as a consequence, we observe that the training performs best when the two first terms are minimized first which then guides the minimization of the structure term. In other words, the model must first learn to predict the nodes before addressing their relationship. Fortunately, this behavior naturally arises in Any2Graph as long as α_A, the hyperparameter controlling the importance of the quadratic term, is not too large. This is illustrated in figure 6.

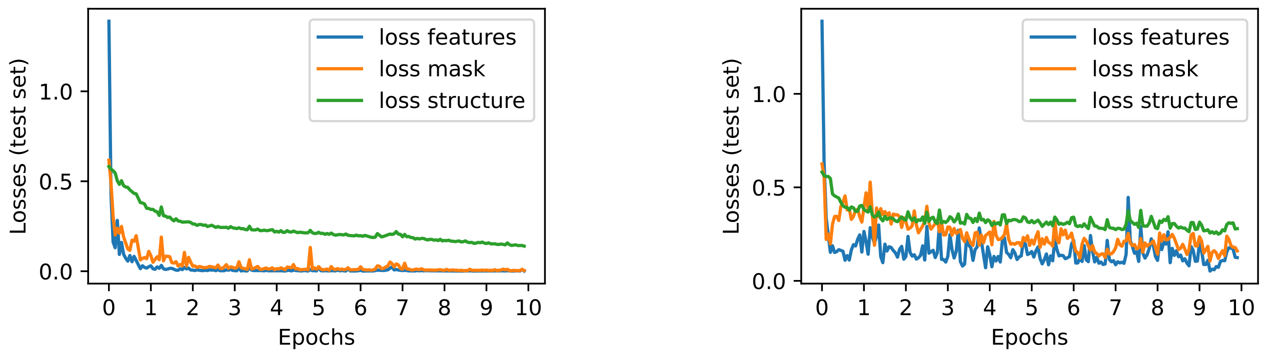

Figure 6: First epochs of training for *Coloring*. The test values of the 3 components of the loss are reported. On the left (resp. right) $\boldsymbol{\alpha}$ is set to $[1, 1, 1]$ (resp. [1,1,10]). In the first scenario, the first two terms of the loss are learned very fast and the structure is optimized next. In the second scenario, setting $\alpha_A = 10$ prevents this desirable learning dynamic.

D.2. Feature Diffusion

For the datasets where many nodes in the graphs share the same features (*QM9* and *GDB13*) the good prediction of the nodes and their features is not enough to guide the prediction of the edges and the desirable dynamic introduced above does not occur. This motivates us to perform Feature Diffusion (FD) that is replacing the node feature vector $\mathbf{F}$ by the concatenation

$$[\mathbf{F}, \mathbf{AF}]$$

Table 3: Performances of the Relationformer and Any2Graph with (+FD) and without feature diffusion.

Dataset	Model	Graph Level			Edge Level		Node Level Acc.	
		Edit Dist. ↓	GI Acc. ↑	PMFGW ↓	Prec. ↑	Rec. ↑	Node ↑	Size ↑
QM9	Relationformer	9.15	0.05	0.48	21.42	4.77	99.28	91.80
	Relationformer + FD	3.80	9.95	0.22	86.07	73.31	99.34	**96.0**
	Any2Graph	3.44	7.50	0.21	86.21	77.27	99.26	93.65
	Any2Graph + FD	**2.13**	**29.85**	**0.14**	**90.19**	**88.08**	**99.77**	95.45
GDB13	Relationformer	11.40	0.00	0.43	81.96	31.49	97.77	97.45
	Relationformer + FD	8.83	0.01	0.29	84.14	55.89	97.57	**98.65**
	Any2Graph	7.45	0.05	0.22	87.20	60.41	99.41	96.15
	Any2Graph + FD	**3.63**	**16.25**	**0.11**	**90.83**	**84.86**	**99.80**	98.15

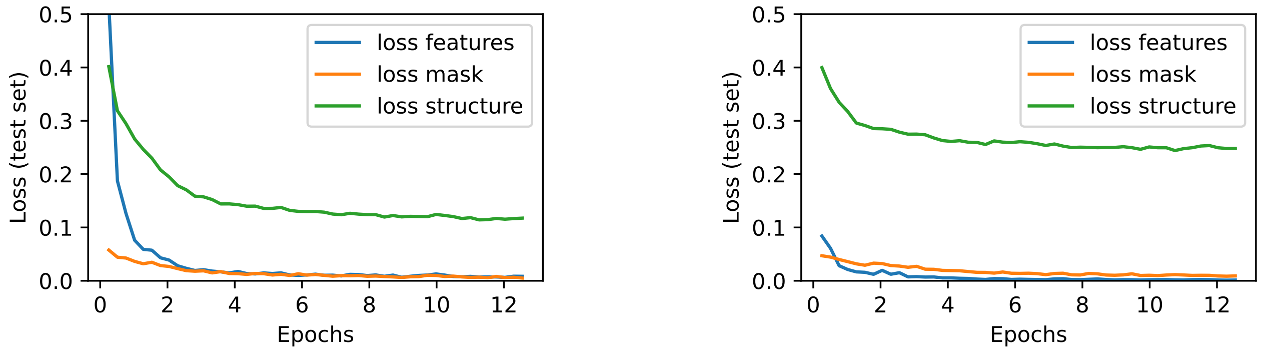

Figure 7: First epochs of training for *GDB13*. The test values of the 3 components of the loss are reported. On the left, we perform FD before training, on the right, we leave node features unchanged. We observe that the feature loss decreases slightly slower with FD (the features are more complex) but the minimization of the structure term is largely accelerated.

before training. The diffused node features carry a portion of the structural information. This makes the node feature term slightly harder to minimize but in turn, the subsequent prediction of the structure is much easier and we recover the previous dynamic. This is illustrated in figure 7. We observe that both Any2Graph and Relationformer benefits from such procedure, as reported in table 3

D.3. Optimal Transport Relaxation

We adopted the OT point of view when designing Any2Graph. In practice, this means that we do not project the OT plan back to the set of permutations with a Hungarian matcher before plugging it in the loss as in [26]. Testing the effect of adding this extra step we observed a 5% to 10% increase of the edit distance across datasets (table 4) along with a more unstable training curve (figure 8). This confirms that a continuous transport plan provides a slightly more stable gradient than a discrete permutation, which aligns with the findings of [27] on the similar topic of object detection.

Dataset	*Coloring*	*Toulouse*	*USCities*	*QM9*	*GDB13*
ED without Hungarian	**0.20**	**0.13**	**1.86**	2.13	**3.63**
ED with Hungarian	0.23	0.15	2.03	**2.08**	3.86

Table 4: Effect of adding Hungarian Matching on the performances evaluated with the test edit distance. We observe, that Hungarian Matching slightly decreases the performances on all datasets but *QM9*.

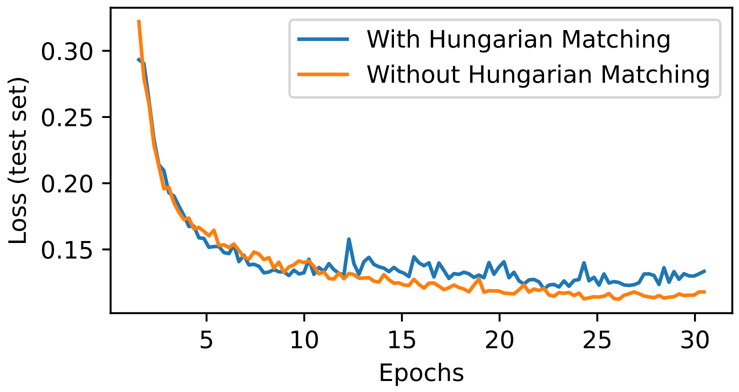

Figure 8: First epochs of training for *GDB13* with and without projection of the optimal transport plan to the set of permutations with Hungarian matching. Hungarian matching slightly decreases the performances and induces more oscillations of the loss, which could be explained by a less stable gradient.

E. Experimental setting

E.1. Datasets

In this paper, we consider five datasets for which we provide a variety of statistics in Table 5. *Coloring* is a new synthetic dataset which we describe in detail in Appendix F. *Toulouse* (resp. *USCities*) is a Sat2Graph dataset [2] where the inputs are images of size 64×64 (resp. 128×128). *QM9* [28] and *GDB13* [29] are datasets of small molecules which we use to address the Fingerprint2Graph task. Here, we compute a fingerprint representation of the molecule and attempt to reconstruct the original molecule from this loss representation. Following [30] we use the Morgan Radius-2 fingerprint [31] which represents a molecule by a bit vector of size 2048, where each bit represents the presence/absence of a given substructure. Finally, we feed our model with the list of non-zeros bits, i.e. the list of substructures (tokens) present in the molecule. The list of substructures has a min/average/max length of $2/21/27$ for *QM9* and $7/29/36$ for *GDB13*.

Table 5: Table summarizing the properties of the datasets considered.

Dataset	Size (train/test/valid)	Nodes (min/mean/max)	Edges (min/mean/max)	Input Modality	Node features
Coloring	100k/10k/10k	4/7.0/10	3/10.9/22	RGB Images	4 Classes (colors)
Toulouse	80k/10k/10k	3/6.1/9	2/5.0/14	Grey Images	2D positions
USCities	130k/10k/10k	2/7.5/17	1/5.8/20	Grey Images	2D positions
QM9	120k/10k/10k	1/8.8/9	0/9.4/13	List of tokens	4 classes (atoms)
GDB13	1300k/70k/70k	5/12.7/13	5/15.15/18	List of tokens	5 classes (atoms)

E.2. Competitors

We compare Any2Graph, to our direct end-to-end competitor Relationformer [7] that has shown to be the state-of-the-art method for Image2Graph. For a fair comparison, we use the same architecture (presented in Figure 2) for both approaches so that the only difference is the loss. We apply feature diffusion both to Any2Graph and Relationformer. Moreover, we also compare with a surrogate regression approach (FGW-Bary-ILE) based on FGW barycenters [6] whose prediction function writes as:

$$f(x) = \arg\min_{y \in \hat{\mathcal{Y}}_m} \sum_{k=1}^{K} \alpha_k(x; W) \mathrm{FGW}_2^2(y, \bar{y}_k), \tag{10}$$

where each $\bar{y}_k$ denotes the k^{th} template graph and α_k the k^{th} weight function. The templates are the training samples and weight function α_k are learned by sketched kernel ridge regression [32], [33] with gaussian kernel. For fair comparison, we also integrated feature diffusion to FGW-Bary-ILE.

E.3. Metrics

In the following, we provide a detailed description of the metrics reported in table 1.

Graph Level First we report the PMFGW loss between continuous prediction $\hat{y}$ and padded target $\mathcal{P}(g)$. For this computation, we set $\boldsymbol{\alpha}$ to the values displayed in table 6.

$$\text{PMFGW}(\hat{y}, \mathcal{P}(g))$$

Then we report the graph edit distance [15] between predicted graph $\mathcal{P}^{-1}\mathcal{T}\hat{y}$ and target g which we compute using Pygmtools [34]. All edit costs (nodes and edges) are set to 1. Note that for *Toulouse* and *USCities*, node labels are 2D positions and we consider two nodes features to be equal (edit cost of 0) whenever the L2 distance is smaller than 5% than the image width.

$$\textsc{Edit}(\mathcal{P}^{-1}\mathcal{T}\hat{y}, g)$$

Finally, we report the Graph Isomorphism Accuracy GI $\textsc{Acc}$ that is

$$\text{GI } \textsc{Acc}(\hat{y}, g) = \mathbb{1}[\textsc{Edit}(\mathcal{P}^{-1}\mathcal{T}\hat{y}, g) = 0]$$

Node level Recall that for a prediction $\hat{y} = (\hat{\mathbf{h}}, \hat{\mathbf{F}}, \hat{\mathbf{A}})$ the size of the predicted graph is $\hat{m} = ||\hat{h} > 0.5||_1$. Denoting m the size of the target graph we report the size accuracy:

$$\text{SIZE } \textsc{Acc}(\hat{y}, g) = \mathbb{1}[\hat{m} = m].$$

The remaining node and edge-level metrics need the graphs to have the same number of nodes. To this end, we select the m nodes with the highest probability $\hat{h}_i$, resulting in a graph $\hat{g} = (\tilde{\mathbf{F}}, \tilde{\mathbf{A}})$ with ground truth size. This is equivalent to assuming that the size of the graph is well predicted. Then we use Pygmtools to compute a one-to-one matching σ between the nodes of $\hat{g}$ and g that can be used to align graphs (we use the matching that minimizes the edit distance). In the following, we assume that g and $\hat{g}$ have been aligned. We can now define the node accuracy NODE $\textsc{Acc}$ as

$$\text{NODE } \textsc{Acc}(\hat{g}, g) = \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}[\tilde{F}_i = F_i],$$

which is the average number of node features that are well predicted.

Edge level Since the target adjacency matrices are typically sparse, the edge prediction accuracy is a poorly informative metric. To mitigate this issue we report both Edge Precision and Edge Recall :

$$\text{EDGE } \textsc{Prec.}(\hat{g}, g) = \frac{\sum_{i,j=1}^{m} \mathbb{1}[\tilde{A}_{i,j} = 1, A_{i,j} = 1]}{\sum_{i,j=1}^{m} \mathbb{1}[\tilde{A}_{i,j} = 1]}$$

$$\text{EDGE } \textsc{Prec.}(\hat{g}, g) = \frac{\sum_{i,j=1}^{m} \mathbb{1}[\tilde{A}_{i,j} = 1, A_{i,j} = 1]}{\sum_{i,j=1}^{m} \mathbb{1}[A_{i,j} = 1]}$$

All those metrics are then averaged other the test set.

E.4. Training And Architecture Hyperparameters

Encoder We follow the guidelines established in B.2 for the choice of the encoder. In particular, all encoders for *Coloring*, *Toulouse* and *USCities* are CNNs. The encoder for *Coloring* is a variation of Resnet18 [35], where we remove the first max-pooling layer and the last two blocks to accommodate for the low resolution of our input image. We proceed similarly for *Toulouse* except that we only remove the last block. For *USCities* we keep the full Resnet18. For the Fingerprint2Graph datasets, we use a transformer encoder. In practice, this transformer encoder and that of the encoder-decoder module are merged to avoid redundancy. All encoders end with a linear layer projecting the feature vectors to the hidden dimension d_e.

Transformer We use the Pre-LN variant [36] of the transformer encoder-decoder model as described in [37]. To reduce the number of hyperparameters, encoder and decoder modules both consist of stacks of N_τ layers, with N_h heads and the hidden dimensions of all MLP is set to $4 \times d_e$.

Decoder All MLPs in the decoder module have one hidden layer.

Optimizer We train all neural networks with the Adam optimizer [38], learning rate η, 8000 warm-up steps and all other hyperparameters set to default values. We also use gradient clipping with a max norm set to 0.1.

All hyperparameters are given in Table 6.

Table 6: Hyperparameters used to train our models. We also report the total training time on a NVIDIA V100.

Dataset	PMFGW			Architecture				Optimization			
	α_H	α_F	α_A	d_e	N_τ	N_h	M	η	Batchsize	Steps	Time
Coloring	1	1	1	256	3	8	12	3E-4	128	75K	4H
Toulouse	1	5	1	256	4	8	12	1E-4	128	100K	8H
USCities	2	5	0.5	256	4	8	20	1E-4	128	150K	14H
QM9	1	1	1	128	3	4	12	3E-4	128	150K	6H
GDB13	1	1	1	512	5	8	15	3E-4	256	150K	24H

F. *Coloring*: a new synthetic dataset for benchmarking SGP

We introduce *Coloring*, a new synthetic dataset well suited for benchmarking SGP methods. The main advantages of *Coloring* are:

- The output graph is uniquely defined from the input image.
- The complexity of the task can be finely controlled by picking the distribution of the graph sizes, the number of node labels (colors) and the resolution of the input image.
- One can generate as many pairs (inputs, output) as needed to explore different regimes, from abundant to scarce data.

To generate a new instance of *Coloring*, we apply the following steps:

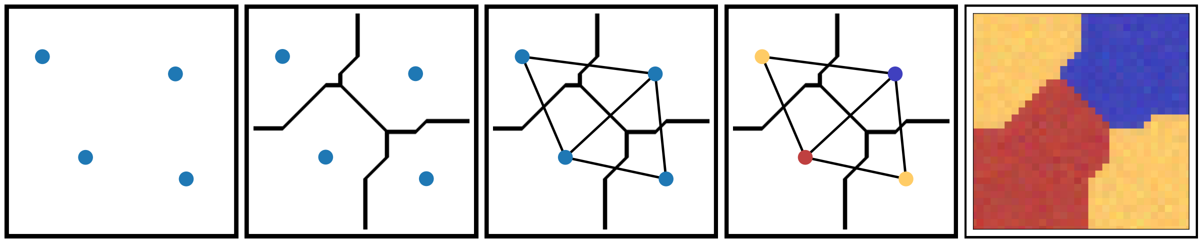

Figure 9: Illustration of the five steps to follow to generate a new instance of *Coloring*.

- 0) Sample the number of nodes (graph size) m. In this paper, we sample uniformly on some interval $[M_{\text{min}}, M_{\text{max}}]$.
- 1) Sample m centroids on $[0,1] \times [0,1]$. In this paper, we sample the centroids as uniform i.i.d. variables.
- 2) Partition $[0,1] \times [0,1]$ (the image) in a Voronoi diagram fashion [39]. In this paper, we use the L_1 distance. and an image of resolution $H \times H$.

- 3) Create the associated graph i.e. each node is a region of the image and two nodes are linked by an edge whenever the two associated regions are adjacent.
- 4) Color the graph with $K > 4$ colors. In this paper, we use $K = 4$. A coloring is said to be valid whenever no adjacent nodes have the same color. Note that graph coloring is known to be NP-complete [40].
- 5) Color the original image accordingly.

As highlighted above, *Coloring* is a flexible dataset. Beyond the default dataset simply referred as *Coloring* we also explore 2 variations in our experiments. *ColoringBig* is a more challenging dataset that features larger graphs. *ColoringVect* is a variation of *Coloring* where the input image is flattened and treated as a vector allowing us to explore a synthetic Vect2Graph task. The properties of these datasets, along with the performances of Any2Graph are reported in table 7. We hope that *Coloring* will be used to benchmark future SGP methods.

Table 7: Summary of the properties of the 3 variations of *Coloring* considered in this paper. We also report the test edit distance achieved by the different models. For FGWBary we report the best performing variant that is ILE for *ColoringVect* and NN for*Coloring*. None scales to *ColoringBig*.

Dataset	M_{min}	M_{max}	H	Number Of Samples	Any2Graph	Relationformer	FGWBary-NN
ColoringVect	4	6	16	100k	0.46	1.40	2.09
Coloring	6	10	32	100k	0.20	5.47	6.73
ColoringBig	10	15	64	200k	1.01	8.91	n.a.

G. Additional Qualitative Results

G.1. Out of distribution performances

We tested if, once trained on Toulouse dataset, the predictive model is able to cope with out-of-distribution data. Figure 10 shows that this is the case on these toy images, that are not related to satellite images or road maps. We leave for future work the investigation of this property.

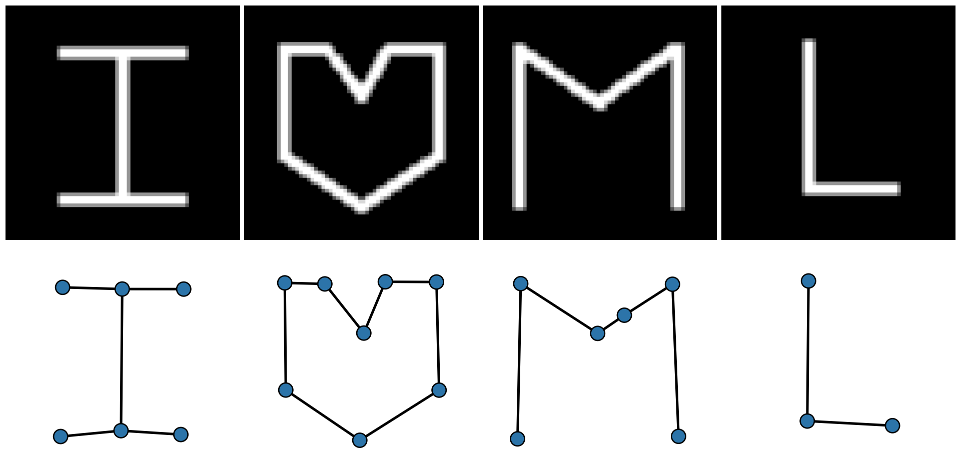

Figure 10: Any2Graph trained on Toulouse performing on out-of-distribution inputs. Input images are displayed on top row and prediction in the bottom row.

G.2. Qualitative results on COLORING

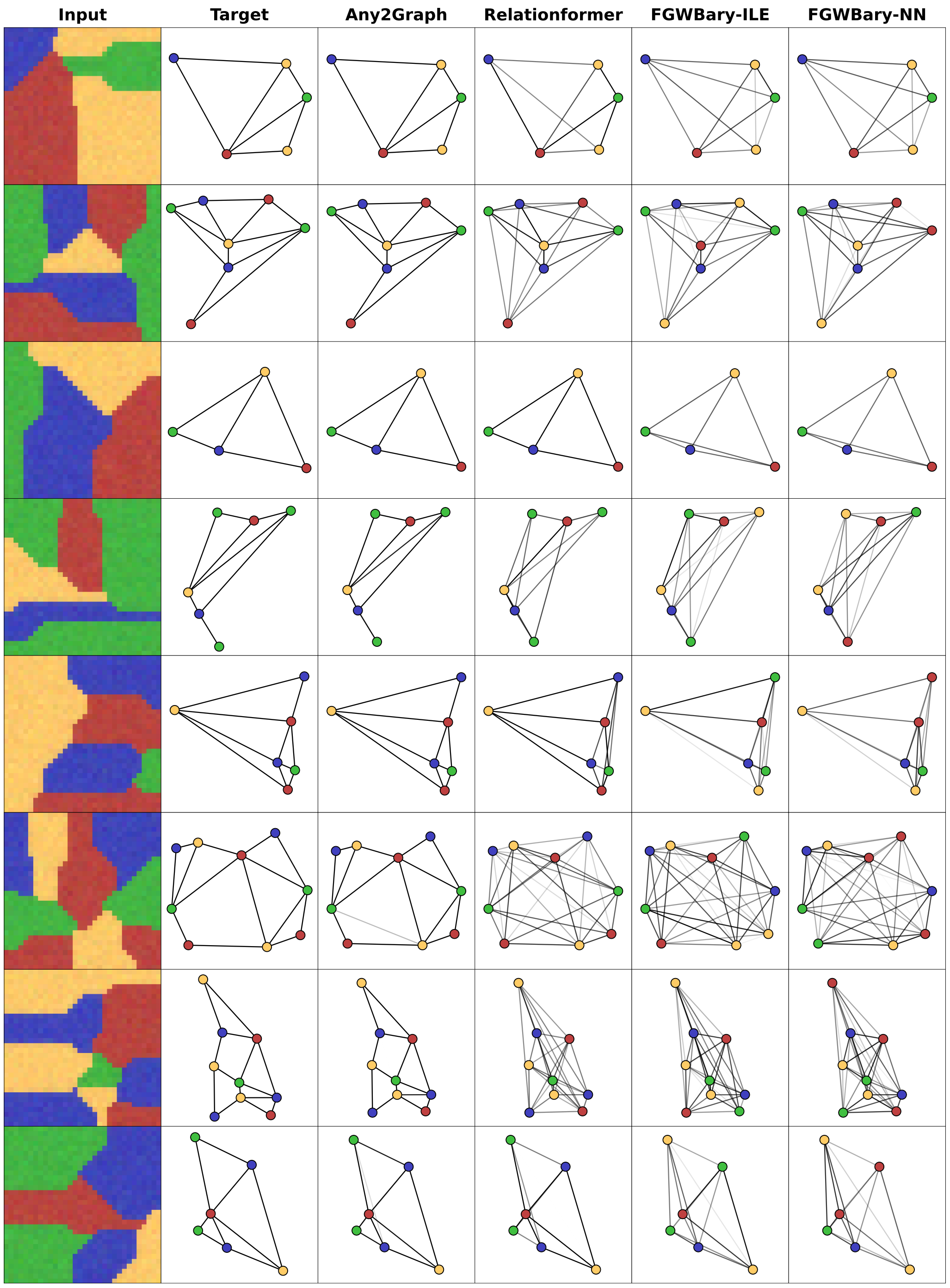

Figure 11: Graph prediction on the *Coloring* dataset.

G.3. Qualitative results on QM9

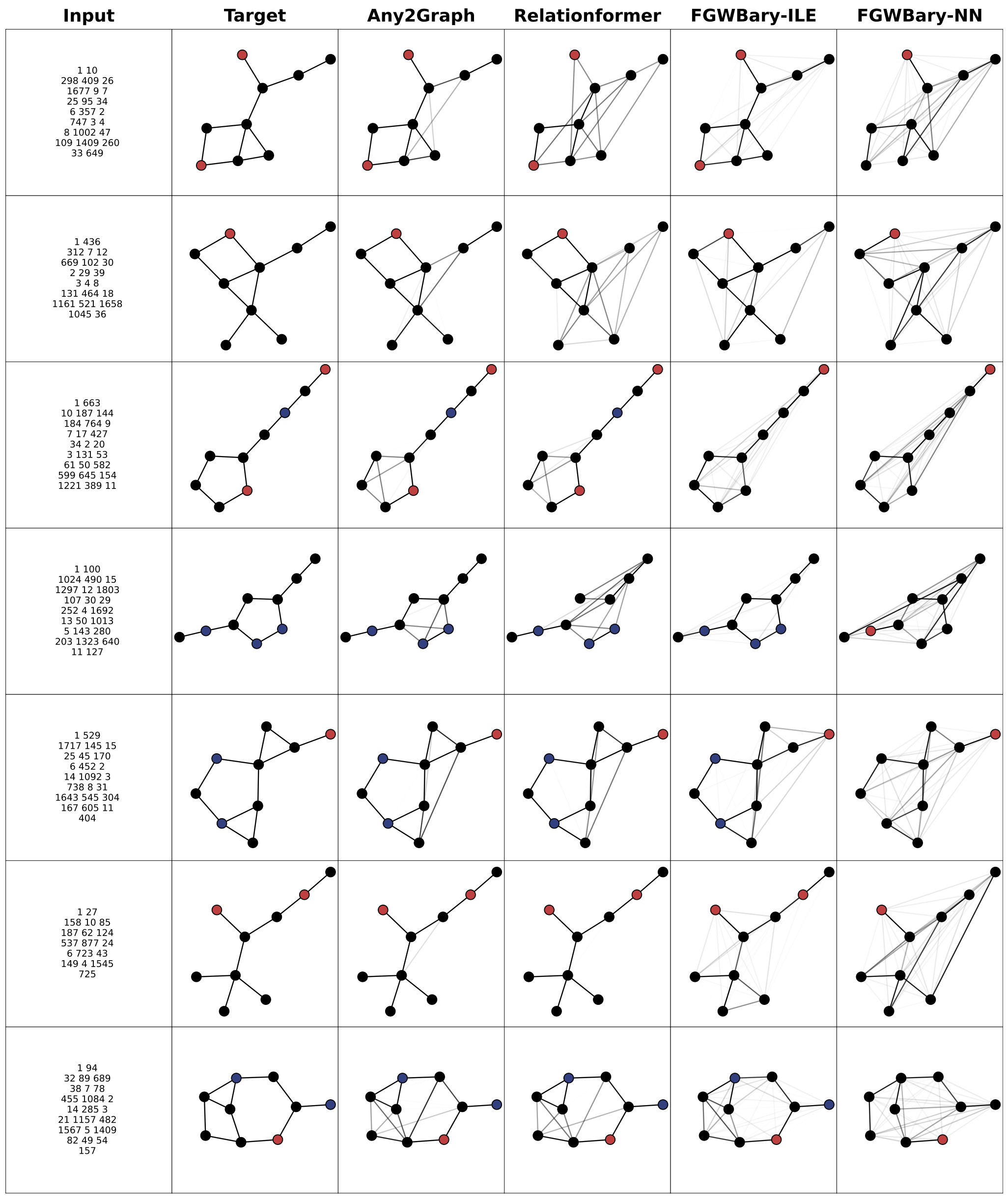

Figure 12: Graph prediction on the *QM9* dataset.

G.4. Qualitative results on GDB13

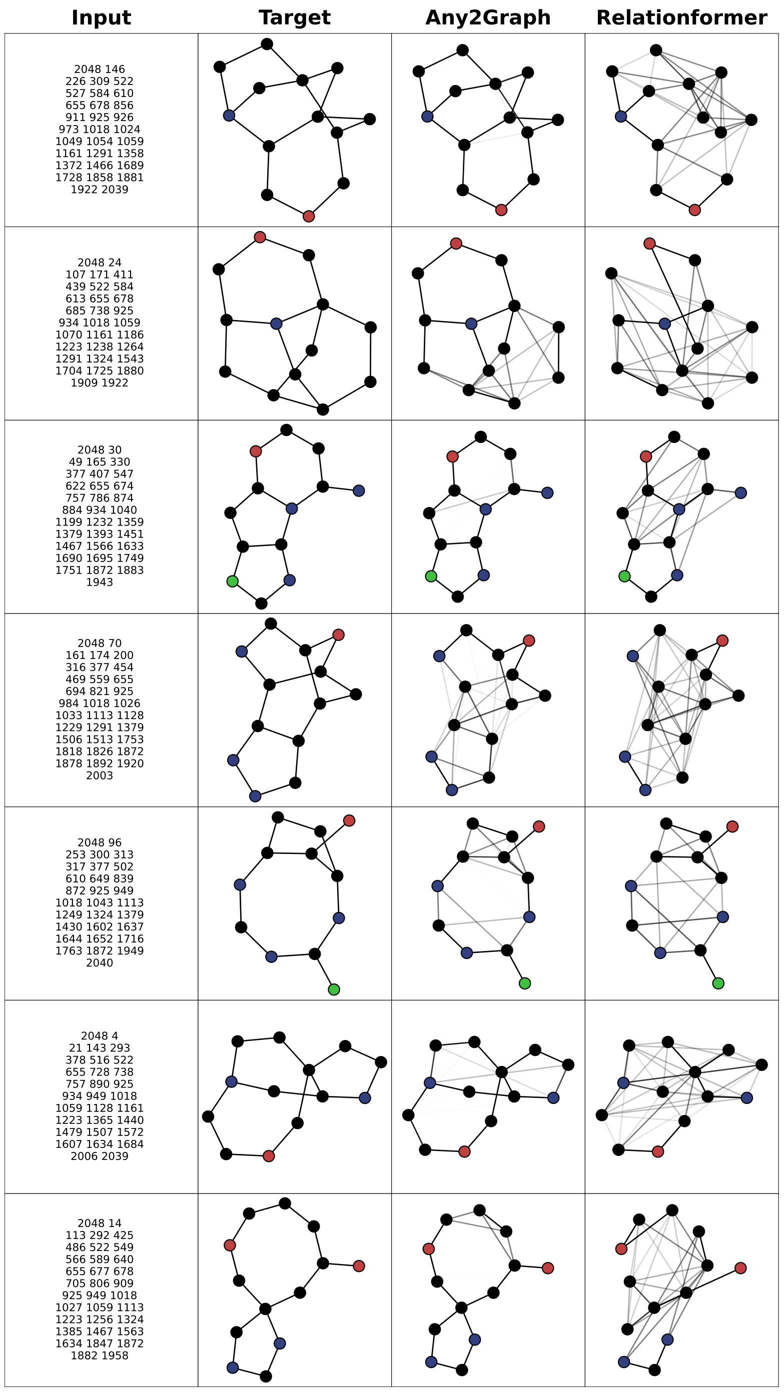

Figure 13: Graph prediction on the *GDB13* dataset.

G.5. Qualitative results on TOULOUSE

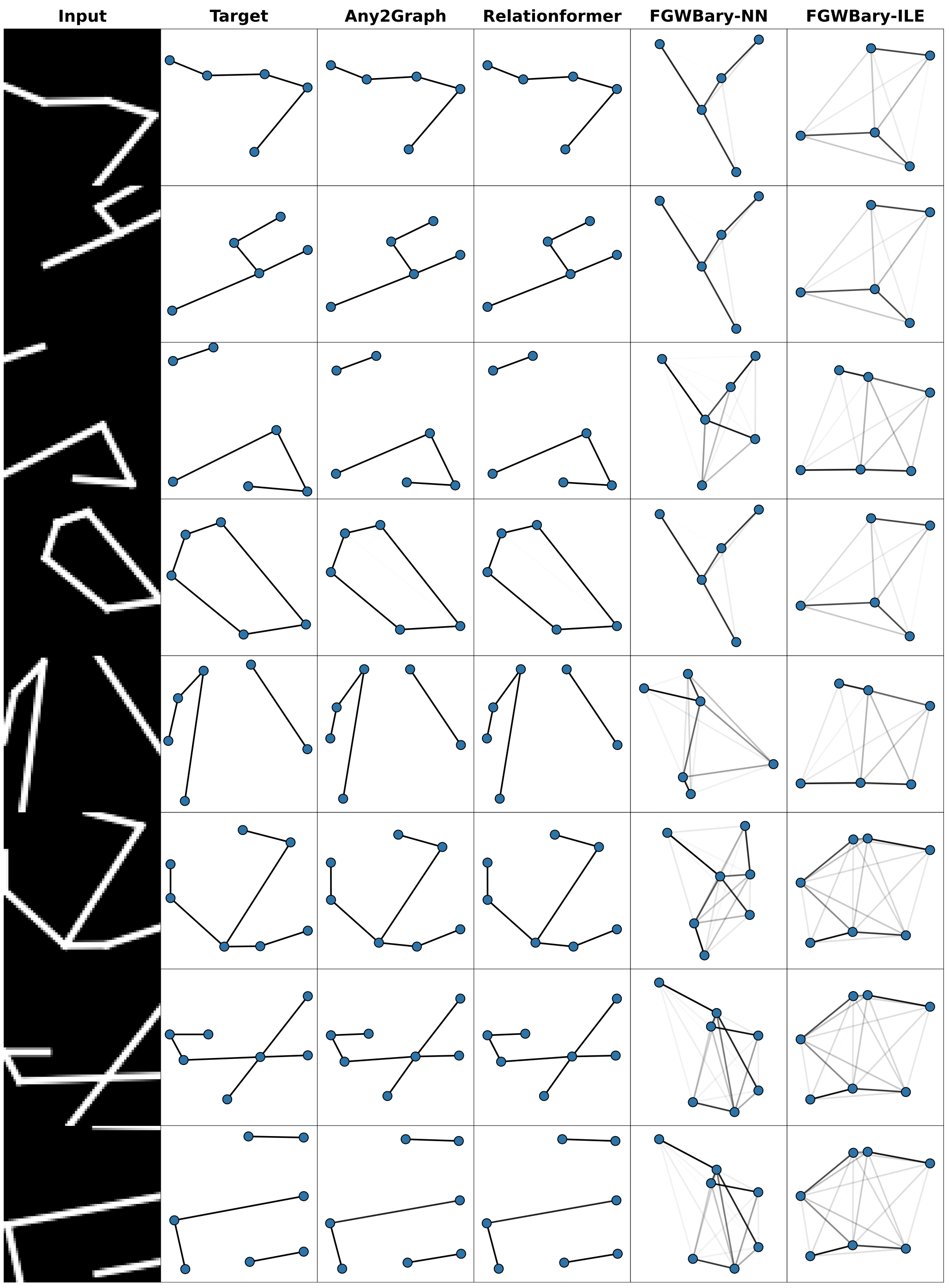

Figure 14: Graph prediction on the *Toulouse* dataset.

G.6. Qualitative results on USCities

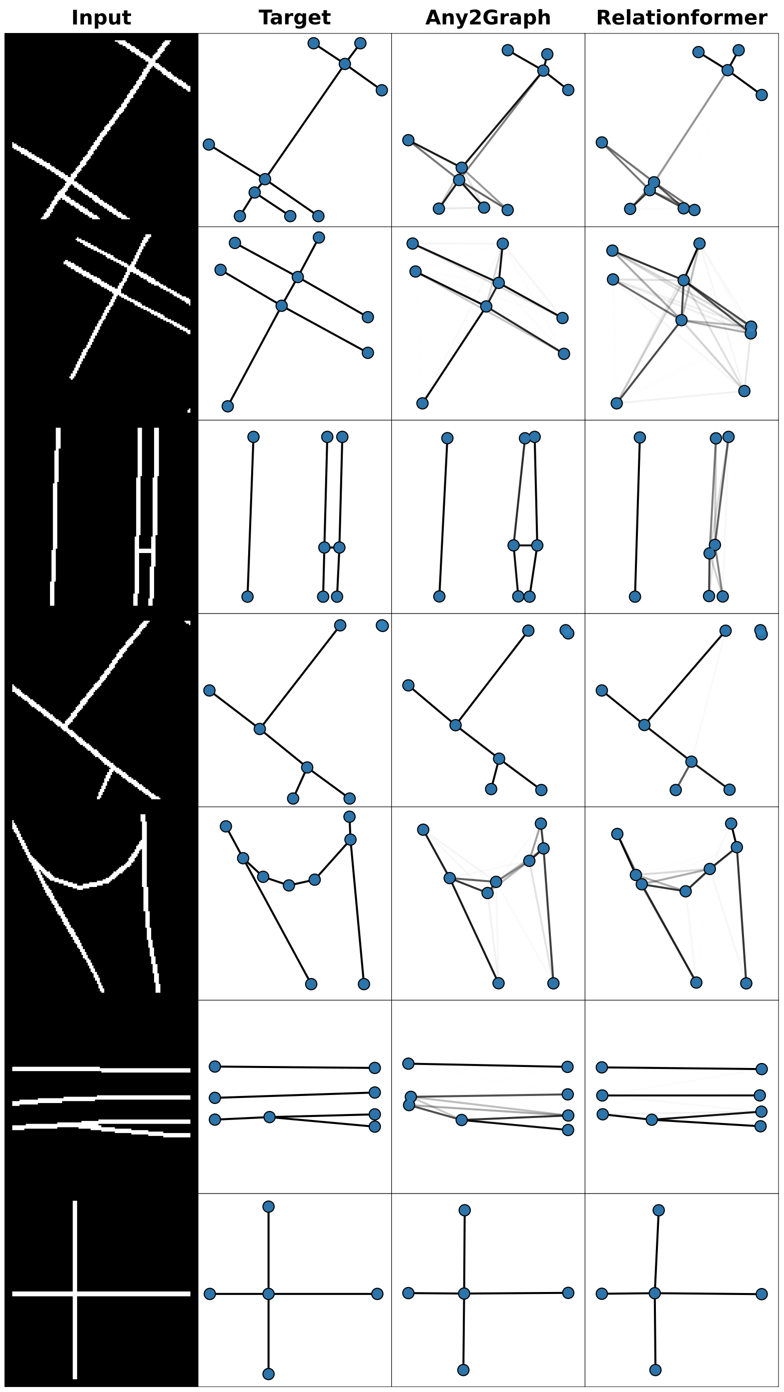

Figure 15: Graph prediction on the *USCities* dataset.