Template based Graph Neural Network with Optimal Transport Distances

Cédric Vincent-Cuaz, Rémi Flamary, Marco Corneli, Titouan Vayer, Nicolas Courty

Template based Graph Neural Network with Optimal Transport Distances

Cédric Vincent-Cuaz
Univ. Côte d'Azur, INRIA, CNRS, LJAD
F-06100 Nice
cedric.vincent-cuaz@inria.fr

Rémi Flamary
IP Paris, CMAP, UMR 7641
F-91120 Palaiseau
remi.flamary@polytechnique.edu

Marco Corneli
Univ. Côte d'Azur, INRIA, CNRS, LJAD
F-06100 Nice
marco.corneli@inria.fr

Titouan Vayer
Univ. Lyon, INRIA, CNRS, ENS de Lyon
LIP UMR 5668, F-69342 Lyon
titouan.vayer@inria.fr

Nicolas Courty
Univ. Bretagne-Sud, CNRS, IRISA
F-56000 Vannes
nicolas.courty@irisa.fr

Abstract

Current Graph Neural Networks (GNN) architectures generally rely on two important components: node features embedding through message passing, and aggregation with a specialized form of pooling. The structural (or topological) information is implicitly taken into account in these two steps. We propose in this work a novel point of view, which places distances to some learnable graph templates at the core of the graph representation. This distance embedding is constructed thanks to an optimal transport distance: the Fused Gromov-Wasserstein (FGW) distance, which encodes simultaneously feature and structure dissimilarities by solving a soft graph-matching problem. We postulate that the vector of FGW distances to a set of template graphs has a strong discriminative power, which is then fed to a non-linear classifier for final predictions. Distance embedding can be seen as a new layer, and can leverage on existing message passing techniques to promote sensible feature representations. Interestingly enough, in our work the optimal set of template graphs is also learnt in an end-to-end fashion by differentiating through this layer. After describing the corresponding learning procedure, we empirically validate our claim on several synthetic and real life graph classification datasets, where our method is competitive or surpasses kernel and GNN state-of-the-art approaches. We complete our experiments by an ablation study and a sensitivity analysis to parameters.

1 Introduction

Attributed graphs are characterized by *i)* the relationships between the nodes of the graph (structural or topological information) and *ii)* some specific features or attributes endowing the nodes themselves. Learning from those data is ubiquitous in many research areas [3], *e.g.* image analysis [21, 8], brain connectivity [30], biological compounds [23] or social networks [69], to name a few. Various methodologies approach the inherent complexity of those data, such as signal processing [54],

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

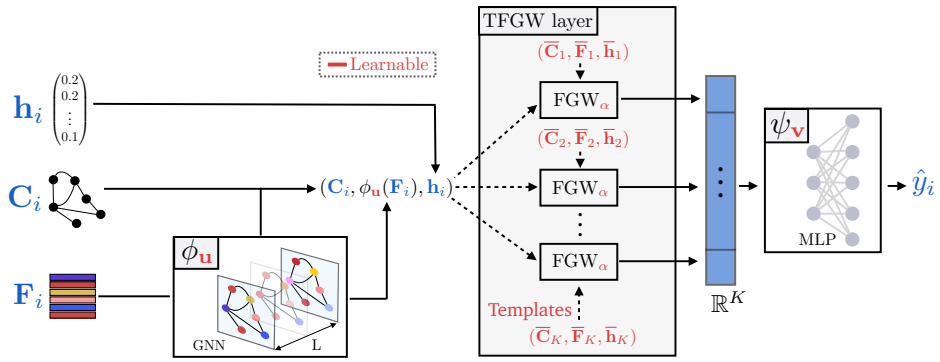

Figure 1: Illustration of the proposed model. **(left)** The input graph is represented as a triplet $(\boldsymbol{C}_i, \boldsymbol{F}_i, \boldsymbol{h}_i)$ where the matrix $\boldsymbol{C}_i$ encodes the structure, $\boldsymbol{F}_i$ the features, $\boldsymbol{h}_i$ the nodes' weights. A GNN $\phi_{\boldsymbol{u}}$ is applied to the raw features in order to extract a meaningful node representations. **(center)** The TFGW layer is applied to the filtered graph and provides a vector representation as FGW distances to templates. **(right)** a final MLP $\psi_{\boldsymbol{v}}$ is applied to this vector in order to predict the final output of the model. All objects in red are parameters that are learned from the data.

Bayesian and kernel methods on graphs [41, 29] or more recently Graph Neural Networks (GNN) [64] in the framework of the geometric deep learning [8, 7].

We are interested in this work in the classification of attributed graphs **at the instance level**. One existing approach consists in designing kernels that leverage topological properties of the observed graphs [5, 17, 19, 53]. For instance, the popular Weisfeiler-Lehman (WL) kernel [52] iteratively aggregates for each node the features of its k-hop neighborhood. Alternative approaches aim at learning vectorial representations of the graphs that can encode the graph structure (*i.e. graph representation learning* [10]). In this domain, GNN lead to state-of-the-art performances with end-to-end learnable embeddings [64]. At a given layer, these architectures typically learn the node embeddings via local permutation-invariant transformations aggregating its neighbour features [37, 25, 20, 67]. In order to obtain a representation of the whole graph suitable for classification, GNNs finally operate a pooling [26, 40] of the node embeddings, either global (*e.g* summation over nodes [67]), or hierarchical (*e.g* by iteratively clustering nodes [71, 70, 32]).

Another line of works targets the construction of meaningful distances that integrate simultaneously the structural and feature information, and that are based on optimal transport (OT) [61, 46]. Originally designed to compare probability distributions based on a geometric notion of optimality, it allows defining very general loss functions between various objects, modeled as probability distributions. In a nutshell, it proceeds by constructing a *coupling* between the distributions that minimizes a specific *cost*. Some approaches dealing with graphs rely on non-parametric models that first embed the graphs into a vectorial space and then match them via OT [43, 56, 27, 35]. Recently [11] proposed the OT-GNN model, that embeds a graph as a vector of the Wasserstein distances between the nodes' embeddings (after GNN pre-processing) and learnt point clouds, acting as templates.

Building further from OT variants, the Gromov-Wasserstein (GW) distance [39] directly handles graphs through the symmetric matrix $\boldsymbol{C}$ that encodes the distance/similarity between every pairs of nodes (*e.g.* adjacency, shortest path) and a *weight vector* $\boldsymbol{h}$ on the nodes encoding the nodes' relative importance. GW has proven to be useful for tasks such as graph matching and partitioning [66, 15] or unsupervised graph dictionary learning [65, 63, 62]. GW has been also extended to directed graphs [14] and to attributed graphs via the Fused Gromov-Wasserstein (FGW) distance [59, 58], that realizes a trade-off between an OT distance with a cost on node features and the GW distance between the similarity matrices. Despite its recent successes on complex unsupervised tasks such as graph clustering [65, 63, 62], FGW has never been explored as part of an end-to-end model for graph classification. In this work, we fill this gap by introducing a novel "layer" that embeds an attributed graph into a vector, whose coordinates are FGW distances to few (learned) graph templates. While FGW can be performed directly on raw data (*i.e.* the input structured graph without any pre-processing), we also consider the case where features representations are learnt from a GNN, similarly to OT-GNN [11], and thus also realizing a particular type of aggregation.

Contributions. We introduce a new GNN layer, named **TFGW** for **Template-based FGW** and illustrated in the center of Figure 1. From an input graph, it computes a vector of FGW distances to learnable graph templates. This layer that can be seen as an alternative to global pooling layers and can be integrated into any neural network architecture. We discuss its properties and the associated invariances. We detail the optimization strategy that enables learning simultaneously GNN pre-processing layers and graph templates relevant for a downstream task in an end-to-end fashion. We empirically demonstrate the relevance of our model in terms of performances compared to several state-of-the-art architectures. Remarkably, we show that a simple GNN model leveraging on our new layer can surpass state-of-the-art performances by a relatively large margin. Finally, we also provide some illustrative interpretations of our method and a sensitivity analysis of our model parameters.

2 Fused Gromov-Wasserstein template based layer

In order to describe our novel template-based GNN layer we first introduce more formally the FGW distance and its properties. In the following we denote by $\Sigma_n := \{\boldsymbol{h} \in \mathbb{R}_+^n | \sum_i h_i = 1\}$ the probability simplex with n-bins, and by $\mathbb{S}_n(\mathbb{A})$ the set of symmetric matrices of size n taking values in $\mathbb{A} \subset \mathbb{R}$.

2.1 Fused Gromov-Wasserstein distance

An undirected attributed graph $\mathcal{G}$ with n nodes can be modeled in the OT context as a tuple $(\boldsymbol{C}, \boldsymbol{F}, \boldsymbol{h})$, where $\boldsymbol{C} \in \mathbb{S}_n(\mathbb{R})$ is a matrix encoding relationships between nodes, $\boldsymbol{F} = (\boldsymbol{f}_1, ..., \boldsymbol{f}_n)^\top \in \mathbb{R}^{n \times d}$ is a node feature matrix and $\boldsymbol{h} \in \Sigma_n$ is a vector of weights modeling the relative importance of the nodes within the graph (Figure 1, left). Without any prior knowledge, uniform weights can be chosen ($\boldsymbol{h} = \mathbf{1}_n/n$). The matrix $\boldsymbol{C}$ can be the graph adjacency matrix, the shortest-path matrix or any other description of the node relationships (i.e. the topology) of the graph [47, 58, 15]. Let us now consider two such graphs $(\boldsymbol{C}, \boldsymbol{F}, \boldsymbol{h})$ and $(\overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \overline{\boldsymbol{h}})$, of respective sizes n and $\overline{n}$ (with possibly $n \neq \overline{n}$). The Fused Gromov-Wasserstein (FGW) distance is defined for $\alpha \in [0, 1]$ as [58, 59]:

$$\mathrm{FGW}_\alpha(\boldsymbol{C}, \boldsymbol{F}, \boldsymbol{h}, \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \overline{\boldsymbol{h}}) = \min_{\boldsymbol{T} \in \mathcal{U}(\boldsymbol{h}, \overline{\boldsymbol{h}})} \sum_{ijkl} \left(\alpha(C_{ij} - \overline{C}_{kl})^2 + (1-\alpha)\|\boldsymbol{f}_i - \overline{\boldsymbol{f}}_k\|_2^2\right) T_{ik} T_{jl} \quad (1)$$

where $\mathcal{U}(\boldsymbol{h}, \overline{\boldsymbol{h}}) := \{\boldsymbol{T} \in \mathbb{R}_+^{n \times \overline{n}} | \boldsymbol{T}\mathbf{1}_{\overline{n}} = \boldsymbol{h}, \boldsymbol{T}^\top \mathbf{1}_n = \overline{\boldsymbol{h}}\}$ is the set of admissible coupling between $\boldsymbol{h}$ and $\overline{\boldsymbol{h}}$. FGW aims at finding an optimal coupling $\boldsymbol{T}^\star$ by minimizing a trade-off cost, via α, between a Wasserstein (W) cost on the features and a Gromov-Wasserstein (GW) cost on the similarity matrices, both sharing the same coupling. The optimal coupling $\boldsymbol{T}^\star$ acts as a soft matching of the nodes, which tends to associate pairs of nodes that have similar pairwise relations in $\boldsymbol{C}$ and $\overline{\boldsymbol{C}}$ (GW cost), and similar features in $\boldsymbol{F}$ and $\overline{\boldsymbol{F}}$ (W cost).

Interestingly, FGW defines a metric on the space of attributed graphs. In particular, if $\boldsymbol{C}$ and $\overline{\boldsymbol{C}}$ are shortest-path matrices, the FGW distance vanishes if and only if the two attributed graphs are the same up to a permutation [59, Theorem 3.2]. Such invariance involves that two graphs *strongly* isomorphic according to Weisfeiler-Lehman base tests [33, 56] will have a zero FGW distance for any α and, more importantly, $\mathrm{FGW}_\alpha = 0$ implies that the graphs are strongly isomorphic[1]. When $\boldsymbol{C}, \overline{\boldsymbol{C}}$ are any symmetric matrices, we can mention that GW ($\alpha = 1$) also defines a pseudo-distance [55, Theorem 5.8] with respect to the notion of *weak* isomorphism [55, 15].

Solving for FGW. The optimization problem 1 is a non-convex quadratic program [59, equation 6], whose non-convexity comes from the GW cost. A possible optimization procedure to solve this problem is a Conditional Gradient (CG) algorithm, which is known to converge to a local optimum [31]. The computational complexity of each iteration is $O(n^2\overline{n} + \overline{n}^2 n)$ [47]. Thus, if two graphs of considerably different sizes are considered, the complexity is quadratic with respect to the largest size. Existing attempts to reduce this computational cost either exploit entropic regularization of OT [47, 50] or graph partitioning [66, 13].

[1]Two graphs $(\boldsymbol{C}, \boldsymbol{F}, \boldsymbol{h})$ and $(\overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \overline{\boldsymbol{h}})$ are strongly isomorphic if $n = \overline{n}$ and there exists a permutation matrix $\mathbf{P} \in \{0, 1\}^{n \times n}$ such that $\overline{\boldsymbol{C}} = \mathbf{P}\boldsymbol{C}\mathbf{P}^\top$, $\overline{\boldsymbol{F}} = \mathbf{P}\boldsymbol{F}$ and $\overline{\boldsymbol{h}} = \mathbf{P}\boldsymbol{h}$

2.2 Template-based (T)FGW Graph Neural Networks

Building upon the FGW distance and its properties, we propose a simple layer for a GNN that takes a graph $(\boldsymbol{C}, \boldsymbol{F}, \boldsymbol{h})$ as input and computes its FGW distances to a list of K *template graphs* $\overline{\mathcal{G}} := \{(\overline{\boldsymbol{C}}_k, \overline{\boldsymbol{F}}_k, \overline{\boldsymbol{h}}_k)\}_{k \in [\![K]\!]}$ as follows :

$$\mathrm{TFGW}_{\overline{\mathcal{G}},\alpha}(\boldsymbol{C}, \boldsymbol{F}, \boldsymbol{h}) := \left[\mathrm{FGW}_\alpha(\boldsymbol{C}, \boldsymbol{F}, \boldsymbol{h}, \overline{\boldsymbol{C}}_k, \overline{\boldsymbol{F}}_k, \overline{\boldsymbol{h}}_k)\right]_{k=1}^K \tag{2}$$

We postulate that this graph representation can be discriminant between the observed graphs due to FGW. This claim relies on the theory of [2] allowing one to learn provably strongly discriminant classifiers based on the distances from the observed graphs and templates that are sampled from the dataset (see e.g. [48] adopting the Wasserstein distance). However such an approach often requires a large amount of templates which might be prohibitive if the distance is costly to compute. Instead, we propose to **learn** the graph templates $\overline{\mathcal{G}}$ in a supervised manner. In the same way, we also learn the trade-off parameter α on the data. As such, the TFGW layer can automatically adapt to the data whose discriminating information can be discovered either in the features or in the structure of the graphs, or in a combination of the two. Moreover, the template structures can leverage on any type of input representation $\boldsymbol{C}_i$ since they are learnt directly from the data. Indeed, in the numerical experiments we implemented the model using either adjacency matrices (ADJ) that provide more interpretable templates (component $C_{i,j} \in [0, 1]$ can be seen as a probability of link between nodes) or shortest path matrices (SP) that are more complex to interpret but encode global relations between the nodes.

The TFGW layer can be used directly as a first layer to build a graph representation feeding a fully connected network (MLP) for *e.g.* graphs classication. In order to enhance the discriminating power of the model, we propose to put a GNN (denoted by $\phi_{\boldsymbol{u}}$ and parametrized by $\boldsymbol{u}$) on top of the TFGW layer. We assume in the remainder that this GNN model $\phi_{\boldsymbol{u}}$ is injective in order to preserve isomorphism relations between graphs (see [67] for more details). With a slight abuse of notation, we write $\phi_{\boldsymbol{u}}(\mathbf{F})$ to denote the feature matrix of an observed graph after being processed by the GNN.

Learning with TFGW-GNN. We focus on a classification task where we observe a dataset $\mathcal{D}$ of I graphs $\{\mathcal{G}_i = (\boldsymbol{C}_i, \boldsymbol{F}_i, \boldsymbol{h}_i)\}_{i \in [\![I]\!]}$ with variable number of nodes $\{n_i\}_{i \in [\![I]\!]}$ and where each graph is assigned to a label $y_i \in \mathcal{Y}$, with $\mathcal{Y}$ a finite set. The full model is illustrated in Figure 1. We first process the features of the nodes of the input graphs via the GNN $\phi_{\boldsymbol{u}}$, then use the TFGW layer to represent the graphs as vectors in $\mathbb{R}^K$. Finally we use the final MLP model $\psi_{\boldsymbol{v}} : \mathbb{R}^K \to \mathcal{Y}$ parameterized by $\boldsymbol{v}$, to predict the label for any input graph. The whole model is learned in a end-to-end fashion by minimizing the cross-entropy loss on the whole dataset leading to the following optimization problem :

$$\min_{\boldsymbol{u}, \boldsymbol{v}, \{(\overline{\boldsymbol{C}}_k, \overline{\boldsymbol{F}}_k, \overline{\boldsymbol{h}}_k)\}, \alpha} \quad \frac{1}{I} \sum_{i=1}^{I} \mathcal{L}\left(y_i, \psi_{\boldsymbol{v}}\left(\mathrm{TFGW}_{\overline{\mathcal{G}},\alpha}(\boldsymbol{C}_i, \phi_{\boldsymbol{u}}(\boldsymbol{F}_i), \boldsymbol{h}_i)\right)\right). \tag{3}$$

Notable parameters of (3) are the template graphs in the embeddings $\{(\overline{\boldsymbol{C}}_k, \overline{\boldsymbol{F}}_k, \overline{\boldsymbol{h}}_k)\}$ and more precisely their pairwise node relationship $\overline{\boldsymbol{C}}_k$, node features $\overline{\boldsymbol{F}}_k$ and the distribution on the nodes $\overline{\boldsymbol{h}}_k$ on the simplex. The last parameter reweighs individual nodes in each template and performs nodes selection when some weights are exactly 0 [63, 62]. Finally, the global parameter α is also learnt from the whole dataset. Although it is possible to learn a different α per template, we observed that this extra level of flexibility is prone to overfitting, and we will not consider it in the experimental section.

Optimization and differentiation of TFGW. We propose to solve the optimization problem in (3) using stochastic gradient descent. The FGW distances are computed by adapting the conditional gradient solver implemented in the POT toolbox [18]. The solver was designed to allow backward propagation of the gradients *w.r.t.* all the parameters of the distance and was adapted to also compute the gradient *w.r.t.* the parameter α. The gradients are obtained using the Envelop Theorem [1] allowing to keep $\boldsymbol{T}^\star$ constant. We used Pytorch [45] to implement the model. The template structure $\overline{\boldsymbol{C}}_k$, node weights $\overline{\boldsymbol{h}}_k$ and α are updated with a projected gradient respectively on the set of symmetric matrices $\mathbb{S}_{\overline{n}_k}(\mathbb{R}_+)$ ($\mathbb{S}_{\overline{n}_k}([0, 1])$ when $\boldsymbol{C}_i$ are adjacency matrices), the simplex $\Sigma_{\overline{n}_k}$ and $[0, 1]$. The projection onto the probability simplex of the node weights leads to sparse solutions [16], therefore

the size of each $(\overline{\boldsymbol{C}}_k, \overline{\boldsymbol{F}}_k, \overline{\boldsymbol{h}}_k)$ can decrease along iterations hence reducing the *effective* number of their parameters to optimize. This way the numerical solver can leverage on the fact that many computations are unnecessary as soon as the weights are set to zero. Note that the FGW solver from POT uses an OT solver implemented in C++ on CPU which means that it comes with some overhead (memory transfer between GPU and CPU) when training the model on GPU. Still the multiple FGW distances computation has been implemented in parallel on CPU with a computational time that remains reasonable in practice (see experimental section 3.3). While a GPU solver can be found when using entropy regularized FGW, it introduces a new parameter related to the regularization strength which is more cumbersome to set, and that we did not consider it in the experiments.

Properties of the TFGW layer. We now discuss a property of the proposed layer resulting from the properties of FGW (see Section 2.1). We have the following result:

Lemma 1 *The* TFGW *embeddings are invariant to strong isomorphism.*

This lemma directly stems from the fact that FGW is invariant to strong isomorphism of one of its inputs. This proposition implies that two graphs with any aforementioned representation which only differ by a permutation of the nodes will share the same TFGW embedding. Moreover such a property holds for any mapping ϕ_u which is injective, such as a Multi-Layer Perceptron (MLP) [22] or any GNN with a sum aggregation scheme as described in [67].

Moreover, the optimal coupling $\boldsymbol{T}^\star$ resulting from TFGW between $(\boldsymbol{C}_i, \phi_{\boldsymbol{u}}(\boldsymbol{F}_i), \boldsymbol{h}_i)$ and the template $(\overline{\boldsymbol{C}}_k, \overline{\boldsymbol{F}}_k, \overline{\boldsymbol{h}}_k)$, will encode correspondances between the nodes of the graph and the nodes of the template that will be propagated during the backward operation. The size of the inputs, the size of the templates and their respective weight $\overline{\boldsymbol{h}}_k$ will play a crucial role regarding this operation. Also note that, since the templates are estimated here to optimize a supervised task, they will promote discriminant distance embedding instead of graph reconstruction quality as proposed in other FGW unsupervised learning methods [63, 62].

3 Numerical experiments

This section aims at illustrating the performances of our approach for graph classification in synthetic and real-world datasets. First, we showcase the relevance of our TFGW layer on existing synthetic datasets known to require expressiveness beyond the WL-test (Section 3.1). Then we benchmark our model with state-of-the-art approaches on well-known real-world datasets (Section 3.2). We finally discuss our results through a sensitivity analysis of our models (Section 3.3).

3.1 Synthetic datasets beyond WL test

Identification of graphs beyond the WL test is one important challenge faced by the GNN community. In order to test the ability of TFGW to handle such fundamentally difficult problems we consider two synthetic datasets: 4-CYCLES [34, 44] contains graphs with (possibly) disconnected cycles where the label y_i is the presence of a cycle of length 4; SKIP-CIRCLES [12] contains circular graphs with skip links and the labels (10 classes) are the lengths of the skip links among $\{2, 3, 4, 5, 6, 9, 11, 12, 13, 16\}$.

Table 1: Average accuracy on synthetic datasets (10 simulations).

model	4-CYCLES	SKIP-CIRCLES
TFGW	**0.99(0.03)**	**1.00(0.00)**
TFGW-fix	0.63(0.11)	**1.00(0.00)**
GIN	0.50(0.00)	0.10(0.00)
DropGIN	**1.00(0.01)**	0.82(0.28)

We compare the performances of the TFGW layer for embedding such graphs with GIN [67] designed to be at least as expressive as the WL test, and DropGIN [44] which proposed a successful dropout technique to overcome some drawbacks of GIN. We replicate the benchmark of [44] by considering for both GIN and DropGIN, 4 GIN layers for 4-CYCLES, and 9 GIN layers for SKIP-CIRCLES as the skip links can form cycles of up to 17 hops. Since the graphs do not have features we use directly the TFGW on the raw graph representation with $\alpha = 1$ hence computing only the GW distance. The GNN methods above artificially adds a feature equal to 1 on all nodes as they have the same degree. For these experiments we use adjacency matrices for $\boldsymbol{C}_i$ and we investigate two flavours of TFGW: 1) in TFGW-fix we fix the templates by sampling *one template per class* from the training dataset (this can be seen as a simpler FGW feature extraction); 2)

Table 2: Test set classification accuracies from 10-fold CV. The first (resp. second) best performing method is highlighted in bold (resp. underlined).

category	model	MUTAG	PTC	ENZYMES	PROTEIN	NCI1	IMDB-B	IMDB-M	COLLAB
Ours	TFGW ADJ (L=2)	**96.4(3.3)**	**72.4(5.7)**	73.8(4.6)	**82.9(2.7)**	**88.1(2.5)**	**78.3(3.7)**	**56.8(3.1)**	**84.3(2.6)**
	TFGW SP (L=2)	94.8(3.5)	70.8(6.3)	**75.1(5.0)**	82.0(3.0)	86.1(2.7)	74.1(5.4)	54.9(3.9)	80.9(3.1)
OT emb.	OT-GNN (L=2)	91.6(4.6)	68.0(7.5)	66.9(3.8)	76.6(4.0)	82.9(2.1)	67.5(3.5)	52.1(3.0)	80.7(2.9)
	OT-GNN (L=4)	92.1(3.7)	65.4(9.6)	67.3(4.3)	78.0(5.1)	83.6(2.5)	69.1(4.4)	51.9(2.8)	81.1(2.5)
	WEGL	91.0(3.4)	66.0(2.4)	60.0(2.8)	73.7(1.9)	75.5(1.4)	66.4(2.1)	50.3(1.0)	79.6(0.5)
GNN	PATCHYSAN	91.6(4.6)	58.9(3.7)	55.9(4.5)	75.1(3.3)	76.9(2.3)	62.9(3.9)	45.9(2.5)	73.1(2.7)
	GIN	90.1(4.4)	63.1(3.9)	62.2(3.6)	76.2(2.8)	82.2(0.8)	64.3(3.1)	50.9(1.7)	79.3(1.7)
	DropGIN	89.8(6.2)	62.3(6.8)	65.8(2.7)	76.9(4.3)	81.9(2.5)	66.3(4.5)	51.6(3.2)	80.1(2.8)
	PPGN	90.4(5.6)	65.6(6.0)	66.9(4.3)	77.1(4.0)	82.7(1.8)	67.2(4.1)	51.3(2.8)	81.0(2.1)
	DIFFPOOL	86.1(2.0)	45.0(5.2)	61.0(3.1)	71.7(1.4)	80.9(0.7)	61.1(2.0)	45.8(1.4)	80.8(1.6)
Kernels	FGW - ADJ	82.6(7.2)	55.3(8.0)	72.2(4.0)	72.4(4.7)	74.4(2.1)	70.8(3.6)	48.9(3.9)	80.6(1.5)
	FGW - SP	84.4(7.3)	55.5(7.0)	70.5(6.2)	74.3(3.3)	72.8(1.5)	65.0(4.7)	47.8(3.8)	77.8(2.4)
	WL	87.4(5.4)	56.0(3.9)	69.5(3.2)	74.4(2.6)	85.6(1.2)	67.5(4.0)	48.5(4.2)	78.5(1.7)
	WWL	86.3(7.9)	52.6(6.8)	71.4(5.1)	73.1(1.4)	85.7(0.8)	71.6(3.8)	52.6(3.0)	81.4(2.1)
	Gain with TFGW	**4.3**	**4.4**	**2.9**	**4.9**	**2.4**	**5.3**	**4.2**	**2.9**

for TFGW we learn the templates from the training data (as many as the number of classes) as proposed in the previous sections. Results are averaged over 10 runs and reported in Table 1. TFGW based methods perform very well on both datasets with impressive results on SKIP-CIRCLE when GNN have limited performances. This is due to the fact that different samples from one class of SKIP-CIRCLE are generated by permuting nodes of the same graph and FGW distances are invariant to these permutations. 4-CYCLES has a more complex structure with intra-class heterogeneity and requires more than two templates to perform as good as DropGIN. To illustrate this we have computed the accuracy on this dataset as a function of the number of templates K in Figure 2. We can see that a perfect classification is reached up to $K = 4$ for TFGW, while TFGW-fix still struggles to generalize at $K = 20$. This illustrates that learning the templates is essential to keep K (and numerical complexity) small while ensuring good performances.

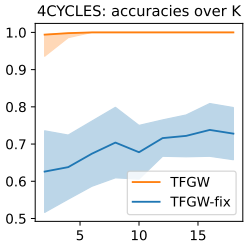

Figure 2: Test accuracy distributions by number of templates either fixed or learned.

3.2 Graph classification benchmmark

We now evaluate and compare the performances of our TFGW GNN with a number of state-of-the-art graph classifiers, from kernel methods to GNN. The numerical experiments are conducted on real life graph datasets to provide a fair benchmark of all methods on several heterogeneous graph structures.

Datasets. We use 8 well-known graph classification datasets [24]: 5 bioinformatics datasets among which 3 have discrete node features (MUTAG, PTC, NCI1 [28, 52]) and 2 have continuous node features (ENZYMES, PROTEINS[6]) and 3 social network datasets (COLLAB, IMDB-B, IDBM-M [69]). In order to analyse them with all methods, we augment unattributed graphs from social networks with node degree features. Detailed description and statistics on these datasets are reported in the supplementary material.

Baselines. We benchmark our approaches to the following state-of-the-art baselines for graphs classification, split into 3 categories: i) *kernel based approaches*, including FGW [59] operating on adjacency and shortest-path matrices, the WL subtree kernel [52, WL] and the Wasserstein WL kernel [56, WWL]. For these methods that do not require a stopping criterion dependent on a validation set, we report results using for parameter validation a 10-fold nested cross-validation [59, 29] repeated 10 times. ii) *OT based representation learning* models, including WEGL [27] and OT-GNN [11]. iii) *GNN models*, with global or more sophisticated pooling operations, including PATCHY-SAN [42], DIFFPOOL [70], PPGN [36], GIN [67] and its augmented version through structure perturbations DropGIN [44]. For all these methods, we adopt the hyper-parameters suggested in the respective papers, but with a slightly different model selection scheme, as detailed in the next paragraph.

Benchmark settings. Recent GNN literature [37, 67, 36, 44] successfully addressed many limitations in terms of model expressiveness compared to the WL tests. Within that scope, they suggested

to benchmark their models using a 10-fold cross-validation (CV) where the best average accuracy on the validation folds was reported. We suggest here to quantify the generalization capacities of GNN based models by performing a 10-fold cross validation with a holdout test set never seen during training. For each split, we track the accuracy on the validation fold every 5 epochs, then the model whose parameters maximize that accuracy is retained. Finally, the model used to predict on the holdout test set is the one with maximal validation accuracy averaged across all folds. This setting is more realistic than a simple 10-fold CV and allows a better understanding of the generalization performances [4]. This point explains why some existing approaches have here different performances than those reported in their original paper.

For all the TFGW based approaches we empirically study the impact of the input structure representation by considering adjacency (ADJ) and shortest-path (SP) matrices $\boldsymbol{C}_i$. For all template based models, we set the size of the templates to the median size of the observed graphs.

We validate the number of templates K in $\{\beta|\mathcal{Y}|\}_\beta$, with $\beta \in \{2, 4, 6, 8\}$ and $|\mathcal{Y}|$ the number of classes. Only for ENZYMES with 6 classes of 100 graphs each, we validate $\beta \in \{1, 2, 3, 4\}$. All parameters of our TFGW layers highlighted in red in Figure 1 are learned while $\phi_{\boldsymbol{u}}$ is a GIN architecture [67] composed of $L = 2$ layers aggregated using the Jumping Knowledge scheme [68] known to prevent overfitting in global pooling frameworks. For OT-GNN we validate the number of GIN layers in $L \in \{2, 4\}$. Finally for fairness, we validate the number of hidden units within the GNN layers and the application of dropout on the final MLP for predictions, similarly to GIN and DropGIN.

Results analysis. The results of the comparisons in terms of accuracy are reported in Table 2. Our TFGW approach consistently *outperforms with significant margins* the state-of-the-art approaches from all categories. Even if most of the benchmarked models can perfectly fit the train sets by learning implicitly the graphs structure [67, 44, 36], enforcing such knowledge explicitly as our TFGW layer does (through FGW distances) leads to considerably stronger generalization performances. On 7 out of 8 datasets, TFGW leads to better performances while operating on adjacency matrices (TFGW ADJ) than on shortest-path ones (TFGW SP). Interestingly, this ranking with respect to those input representations does not necessarily match the one of the FGW kernel which extracts knowledge from the graph structures $\boldsymbol{C}_i$ through FGW, as our TFGW layer. These different dependencies to the provided inputs may be due to the GNN pre-processing of node features which suggests the study of its ablation. Finally to complete this analysis, we report in Table 3 the number of parameters of best selected models across various methods, for the dataset PTC. We refer the reader interested in an extended benchmark to the supplementary material. Our TFGW leads to better classification performances while having comparable number of parameters than these competitors. We also reported for these models, their averaged prediction time per graph. These measures were taken on CPUs (Intel Core i9-9900K CPU, 3.60 GHz) in order to fairly compare the numerical complexity of these methods, as OT solver used in TFGW and OT-GNN are currently limited to these devices (see the detailed discussion in Section 2.1). Although the theoretical complexity of our approach is *at most* cubic in the number of nodes, we still get in practice a fairly good speed for classifying graphs, in comparison to the other competitive methods.

Table 3: Number of parameters and averaged prediction time per graph.

model	PTC	
	parameters	runtimes (ms)
(ours) TFGW	25.1k	12.1
OT-GNN	30.8k	7.6
GIN	29.9k	0.19
DropGIN	44.1k	14.3

3.3 Ablation study, sensitivity analysis and discussions

In this section we inspect the role of some of the model parameters (α in FGW, weights estimation in the templates, depth of the GNN $\phi_{\boldsymbol{u}}$) in terms of the classification performance. To this end, we first conduct on all datasets an ablation study on the graph template weights $\overline{\boldsymbol{h}}_k$ and the number of GIN layers in $\phi_{\boldsymbol{u}}$. Then, we take a closer look at the estimated trade-off parameters α and provide a sensitivity analysis *w.r.t.* the number of templates and the number of GIN layers.

Ablation Study. Following the same procedure as in Section 3.2, we benchmark the following settings for our TFGW models: for adjacency (ADJ) and shortest path (SP) representations $\boldsymbol{C}_i$, we learn the distance layers either directly, on the raw data (*i.e.* $L = 0, \phi_{\mathbf{u}} = \mathrm{id}$), or after embedding the data with $(L = 1)$ or $(L = 2)$ GIN layers. For $L = 0$ we either fix the graph template weights $\overline{\boldsymbol{h}}_k$

Table 4: Classification results from 10-fold cross-validation of our TFGW models in various scenarios: for $L \in \{0, 1, 2\}$ GIN layers, we either fix templates weights $\overline{\boldsymbol{h}}_k$ to uniform distributions or learn them. The first and second best performing method are respectively highlighted in bold and underlined.

model	inputs	$\overline{\boldsymbol{h}}_k$	MUTAG	PTC	ENZYMES	PROTEIN	NCI1	IMDB-B	IMDB-M	COLLAB
TFGW (L=0)	ADJ	uniform	92.1(4.5)	63.6(5.0)	67.4(7.3)	78.0(2.0)	80.3(1.5)	69.9(2.5)	49.7(4.1)	78.7(3.1)
	ADJ	learnt	94.2(3.0)	64.9(4.1)	72.1(5.5)	78.8(2.2)	82.1(2.5)	71.3(4.3)	52.3(2.5)	80.9(2.7)
	SP	uniform	94.8(3.7)	66.5(6.7)	72.7(6.9)	77.5(2.4)	79.6(3.7)	68.1(4.4)	48.3(3.6)	78.4(3.4)
	SP	learnt	95.9(4.1)	67.9(5.8)	**75.1(5.6)**	79.5(2.9)	83.9(2.0)	72.6(3.1)	53.1(2.5)	79.8(2.5)
TFGW(L=1)	ADJ	learnt	94.8(3.1)	68.7(5.8)	72.7(5.1)	81.5(2.8)	85.4(2.8)	76.3(4.3)	55.9(2.4)	82.6(1.8)
	SP	learnt	95.4(3.5)	70.9(5.5)	74.9(4.8)	82.1(3.4)	85.7(3.1)	73.8(4.8)	54.2(3.3)	81.1(2.5)
TFGW (L=2)	ADJ	learnt	**96.4(3.3)**	**72.4(5.7)**	73.8(4.6)	**82.9(2.7)**	**88.1(2.5)**	**78.3(3.7)**	**56.8(3.1)**	**84.3(2.6)**
	SP	learnt	94.8(3.5)	70.8(6.3)	**75.1(5.0)**	82.0(3.0)	86.1(2.7)	74.1(5.4)	54.9(3.9)	80.9(3.1)

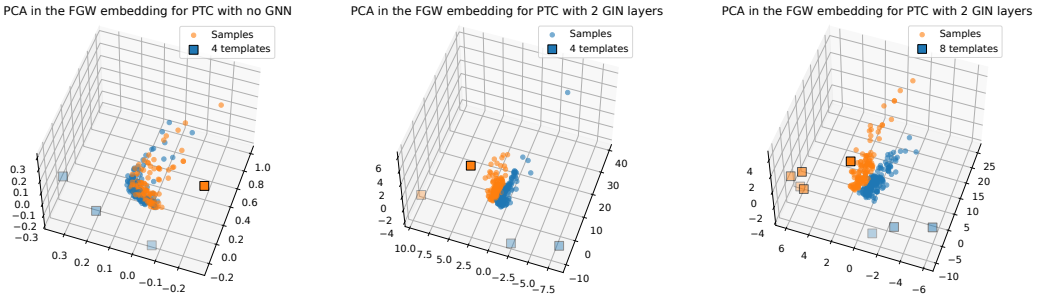

Figure 3: PCA projections of the template based embeddings for different models and number of templates.(need to debug colors)

uniformly or learn them. The results (test accuracy) are reported in Table 4. Learning the weights systematically improves the generalization capabilities of our models of at least 1% or 2% for both ADJ and SP graph representations. For a given number of graph templates, the weights learning allows to better fit the specificities of the classes (e.g. varying proportion of nodes in different parts of the graphs). Moreover, as weights can become sparse in the simplex during training they also allow the model to have templates whose number of nodes adapts to the classification objective, while bringing computational benefits as discussed in Section 2.2. Those observations explain why we learnt them by default in the benchmark of the previous subsection.

Next, we see in Table 4 that using GNN layers as a pre-processing for our TFGW layer enhances generalization powers of our models, whose best performances are obtained for $L = 2$. Interestingly, for $L = 0$, TFGW with SP matrices outperforms TFGW with ADJ matrices, meaning that the shortest path distance brings more discriminant information on raw data. But when $L \geq 1$ (*i.e.* when a GNN pre-processes the node features), TFGW with ADJ matrices improves the accuracy. An explanation could be that the GNN $\phi_{\boldsymbol{u}}$ can somehow replicate (and outperform) a SP metric between nodes. This emphasizes that the strength of our approach clearly exhibited in Table 2 lies in the inductive bias of our FGW distance embedding. This last point is further reinforced by additional experiments reported in the supplementary material, where GIN layers (used by default for our TFGW model) are replaced by GAT layers [60].

Importance of the structure/feature aspect of FGW. To the best of our knowledge we are the first to actually learn the trade-off parameter α of FGW in a supervised way. For this matter, we verify that our models did not converge to degenerated solutions where either the structure ($\alpha = 0$) or the features ($\alpha = 1$) are omitted. To this end we report in Figure 4 the distributions of the estimated α for some models learnt on datasets PTC and IMDB-B, where features are respectively existing in the dataset or created using node degrees. We can see that for both kinds of input graph representations, α parameters are strictly between 0 and 1. One can notice the variances of those distributions illustrating the non-uniqueness

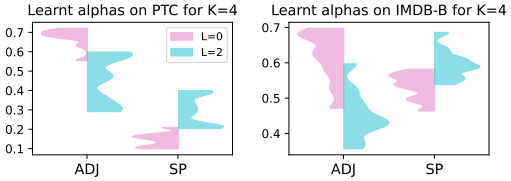

Figure 4: Distributions of estimated α.

of this trade-off parameter coming from the non-convexity of our optimization problem (a given value of α can potentially be compensated by the scaling of the GNN output). Unfortunately, the analysis of the real relative importance between structures and features can not be achieved only by looking at those values as the node embeddings and templates are different across models and data splits.

Visualizing the TFGW embedding. In order to interpret the TFGW embedding, we illustrate in Figure 3 the PCA projection of our distance embeddings learned on PTC with $L = 0$ and $L = 2$ and the number of templates K varying in $\{4, 8\}$. For this experiment, we have chosen the PCA because it allows to have a more interpretable low dimensional projection that preserves the geometry compared to local neighbourhood based embeddings such as TSNE [57] or UMAP [38]. As depicted in the figure, the learned templates are extreme points in the embedding space of the PCA. This result is particularly interesting because existing unsupervised FGW representation learning methods tend toward estimating templates that belong to the data manifold, or to form a "convex enveloppe" of the data to ensure good reconstruction [63, 65]. On the contrary, the templates learned through our approach seem to be located on a plane in the PCA space while the samples evolve orthogonally to this plane (when the FGW distance increases). In a classification context, this means that the learned templates will not actually represent realistic graphs from the data but might encode "exaggerated" or "extreme" features in order to maximize the margin between classes in the embedding. To reinforce this intuition, we added plots of the estimated templates in the supplementary. Finally, in the figure, the samples are coloured *w.r.t.* their class and the templates are coloured by their predicted class. Interestingly, the classes are already well separated with 4 templates but the separation is clearly non-linear whereas using GNN pre-processing and a larger number of templates leads to a linear separation of the two classes.

Sensitivity to the number of templates and GNN layers. To illustrate the sensitivity of our TFGW layer to the number of templates K and the number of GNN layers L in $\phi_{\mathbf{u}}$, we learned our models on the PTC dataset with L varying in $\{0, 1, 2, 3, 4\}$. We follow the same procedure than in the benchmark of Section 3.2 regarding the validation of K and the learning process, while fixing the number of hidden units in the GNN layers to 16. The test accuracy distributions for all settings are reported in Figure 5. Two phases are clearly distinguishable. The first one for $L \leq 2$, where for each L we see that the performance across the number of templates steadily increases, and the second for $L > 2$ where this performance progressively decreases as a function of L. Moreover, in the first phase performances are considerably dependent on K to compensate for a simple node representation, while this dependency is mitigated in the second which exhibits a slight overfitting. Note that these deeper models still lead to competitive results in comparison with benchmarked approaches in Table 2, with best averaged accuracies of 70.6 $(L = 3)$ and 67.4 $(L = 4)$. On one hand, these observations led us to set our number of layers to $L = 2$ for all benchmarked datasets which lead to strong generalization power. On the other hand, deeper models might be a way to benefit from our FGW embeddings with very few templates which can be interesting from a computational perspective on larger graph datasets.

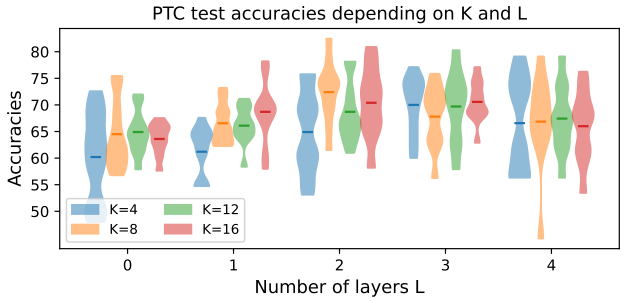

Figure 5: Test accuracy distributions by number of templates and number of GNN layers.

4 Conclusion

We have introduced a new GNN layer whose goal is to represent a graph by its distances to template graphs, according to the optimal transport metric FGW. The proposed layer can be used directly on raw graph data as the first layer of a GNN or can also benefit from more involved node embedding using classical GNN layers. In a graph classification context, we combined this TFGW layer with a simple MLP model. We demonstrated on several benchmark datasets that this approach compared favorably with state-of-the-art GNN and kernel based classifiers. A sensitivity analysis and an ablation

study were presented to justify the choice of several parameters explaining the good generalization performances.

We believe that the new way to represent complex structured data provided by TFGW will open the door to novel and hopefully more interpretable GNN architectures. From a practical perspective, future works will be dedicated to combine TFGW with fast GPU solvers for network flow [51]. This would greatly accelerate our approach and more generally OT based deep learning methods. We also believe that the FGW distance and its existing extensions can be used with other learning strategies including semi-relaxed FGW [62] for sub-graph detection.

5 Acknowledgements

This work is partially funded through the projects OTTOPIA ANR20-CHIA-0030 and 3IA Côte d'Azur Investments ANR-19-P3IA-0002 of the French National Research Agency (ANR). This research was supported by 3rd Programme d'Investissements d'Avenir ANR-18-EUR-0006-02. This action benefited from the support of the Chair "Challenging Technology for Responsible Energy" led by l'X – Ecole polytechnique and the Fondation de l'Ecole polytechnique. This work is supported by the ACADEMICS grant of the IDEXLYON, project of the Université de Lyon, PIA operated by ANR-16-IDEX-0005. This project was supported in part by the AllegroAssai ANR project ANR-19-CHIA-0009. The authors are grateful to the OPAL infrastructure from Université Côte d'Azur for providing resources and support.

References

[1] S. Afriat. Theory of maxima and the method of lagrange. *SIAM Journal on Applied Mathematics*, 20(3):343–357, 1971.

[2] M.-F. Balcan, A. Blum, and N. Srebro. A theory of learning with similarity functions. *Machine Learning*, 72(1):89–112, 2008.

[3] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu. Relational inductive biases, deep learning, and graph networks, 2018.

[4] Y. Bengio and Y. Grandvalet. No unbiased estimator of the variance of k-fold cross-validation. *Advances in Neural Information Processing Systems*, 16, 2003.

[5] K. M. Borgwardt and H.-P. Kriegel. Shortest-path kernels on graphs. In *Fifth IEEE international conference on data mining (ICDM'05)*, pages 8–pp. IEEE, 2005.

[6] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56, 2005.

[7] M. M. Bronstein, J. Bruna, T. Cohen, and P. Velivcković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *ArXiv*, abs/2104.13478, 2021.

[8] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[9] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[10] I. Chami, S. Abu-El-Haija, B. Perozzi, C. Ré, and K. Murphy. Machine learning on graphs: A model and comprehensive taxonomy. *Journal of Machine Learning Research*, 23(89):1–64, 2022.

[11] B. Chen, G. Bécigneul, O.-E. Ganea, R. Barzilay, and T. Jaakkola. Optimal transport graph neural networks. *arXiv preprint arXiv:2006.04804*, 2020.

[12] Z. Chen, S. Villar, L. Chen, and J. Bruna. On the equivalence between graph isomorphism testing and function approximation with gnns. *Advances in neural information processing systems*, 32, 2019.

[13] S. Chowdhury, D. Miller, and T. Needham. Quantized gromov-wasserstein. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 811–827. Springer, 2021.

[14] S. Chowdhury and F. Mémoli. The Gromov-Wasserstein distance between networks and stable network invariants. *arXiv:1808.04337 [cs, math]*, Sept. 2019. arXiv: 1808.04337.

[15] S. Chowdhury and T. Needham. Generalized spectral clustering via gromov-wasserstein learning. In *International Conference on Artificial Intelligence and Statistics*, pages 712–720. PMLR, 2021.

[16] L. Condat. Fast projection onto the simplex and the l1 ball. *Mathematical Programming*, 158(1):575–585, 2016.

[17] A. Feragen, N. Kasenburg, J. Petersen, M. de Bruijne, and K. Borgwardt. Scalable kernels for graphs with continuous attributes. *Advances in neural information processing systems*, 26, 2013.

[18] R. Flamary, N. Courty, A. Gramfort, M. Z. Alaya, A. Boisbunon, S. Chambon, L. Chapel, A. Corenflos, K. Fatras, N. Fournier, L. Gautheron, N. T. Gayraud, H. Janati, A. Rakotomamonjy, I. Redko, A. Rolet, A. Schutz, V. Seguy, D. J. Sutherland, R. Tavenard, A. Tong, and T. Vayer. Pot: Python optimal transport. *Journal of Machine Learning Research*, 22(78):1–8, 2021.

[19] T. Gärtner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Learning theory and kernel machines*, pages 129–143. Springer, 2003.

[20] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[21] Z. Harchaoui and F. Bach. Image classification with segmentation graph kernels. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007.

[22] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[23] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[24] K. Kersting, N. M. Kriege, C. Morris, P. Mutzel, and M. Neumann. Benchmark data sets for graph kernels, 2016.

[25] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[26] B. Knyazev, G. W. Taylor, and M. Amer. Understanding attention and generalization in graph neural networks. *Advances in neural information processing systems*, 32, 2019.

[27] S. Kolouri, N. Naderializadeh, G. K. Rohde, and H. Hoffmann. Wasserstein embedding for graph learning. In *International Conference on Learning Representations*, 2021.

[28] N. Kriege and P. Mutzel. Subgraph matching kernels for attributed graphs. *arXiv preprint arXiv:1206.6483*, 2012.

[29] N. M. Kriege, F. D. Johansson, and C. Morris. A survey on graph kernels. *Applied Network Science*, 5(1):1–42, 2020.

[30] S. I. Ktena, S. Parisot, E. Ferrante, M. Rajchl, M. Lee, B. Glocker, and D. Rueckert. Distance metric learning using graph convolutional networks: Application to functional brain networks. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 469–477. Springer, 2017.

[31] S. Lacoste-Julien. Convergence rate of frank-wolfe for non-convex objectives. *arXiv preprint arXiv:1607.00345*, 2016.

[32] J. Lee, I. Lee, and J. Kang. Self-attention graph pooling. In *International conference on machine learning*, pages 3734–3743. PMLR, 2019.

[33] A. Leman. The reduction of a graph to canonical form and the algebra which appears therein. 2018.

[34] A. Loukas. What graph neural networks cannot learn: depth vs width. In *International Conference on Learning Representations*, 2020.

[35] H. P. Maretic, M. El Gheche, G. Chierchia, and P. Frossard. Got: an optimal transport framework for graph comparison. In *Advances in Neural Information Processing Systems*, pages 13876–13887, 2019.

[36] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman. Provably powerful graph networks. *Advances in neural information processing systems*, 32, 2019.

[37] H. Maron, H. Ben-Hamu, N. Shamir, and Y. Lipman. Invariant and equivariant graph networks. In *International Conference on Learning Representations*, 2019.

[38] L. McInnes, J. Healy, and J. Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[39] F. Mémoli. Gromov–wasserstein distances and the metric approach to object matching. *Foundations of computational mathematics*, 11(4):417–487, 2011.

[40] D. Mesquita, A. Souza, and S. Kaski. Rethinking pooling in graph neural networks. *Advances in Neural Information Processing Systems*, 33:2220–2231, 2020.

[41] Y. C. Ng, N. Colombo, and R. Silva. Bayesian semi-supervised learning with graph gaussian processes. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[42] M. Niepert, M. Ahmed, and K. Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pages 2014–2023. PMLR, 2016.

[43] G. Nikolentzos, P. Meladianos, and M. Vazirgiannis. Matching node embeddings for graph similarity. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 2429–2435, 2017.

[44] P. A. Papp, K. Martinkus, L. Faber, and R. Wattenhofer. DropGNN: Random dropouts increase the expressiveness of graph neural networks. In *Advances in Neural Information Processing Systems*, 2021.

[45] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[46] G. Peyré and M. Cuturi. Computational optimal transport. *Foundations and Trends in Machine Learning*, 11:355–607, 2019.

[47] G. Peyré, M. Cuturi, and J. Solomon. Gromov-wasserstein averaging of kernel and distance matrices. In *International Conference on Machine Learning*, pages 2664–2672, 2016.

[48] A. Rakotomamonjy, A. Traoré, M. Berar, R. Flamary, and N. Courty. Distance measure machines. *arXiv preprint arXiv:1803.00250*, 2018.

[49] F. Santambrogio. Optimal transport for applied mathematicians. *Birkäuser, NY*, 55(58-63):94, 2015.

[50] M. Scetbon, G. Peyré, and M. Cuturi. Linear-time gromov wasserstein distances using low rank couplings and costs, 2021.

[51] A. Shekhovtsov and V. Hlaváč. A distributed mincut/maxflow algorithm combining path augmentation and push-relabel. *International journal of computer vision*, 104(3):315–342, 2013.

[52] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

[53] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt. Efficient graphlet kernels for large graph comparison. In *Artificial intelligence and statistics*, pages 488–495. PMLR, 2009.

[54] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30:83–98, 2013.

[55] K.-T. Sturm. The space of spaces: curvature bounds and gradient flows on the space of metric measure spaces. *arXiv preprint arXiv:1208.0434*, 2012.

[56] M. Togninalli, E. Ghisu, F. Llinares-López, B. Rieck, and K. Borgwardt. Wasserstein weisfeiler–lehman graph kernels. In *Advances in Neural Information Processing Systems*, pages 6436–6446. Curran Associates, Inc., 2019.

[57] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

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

[59] T. Vayer, N. Courty, R. Tavenard, and R. Flamary. Optimal transport for structured data with application on graphs. In *International Conference on Machine Learning*, pages 6275–6284. PMLR, 2019.

[60] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[61] C. Villani. *Optimal transport: old and new*, volume 338. Springer, 2009.

[62] C. Vincent-Cuaz, R. Flamary, M. Corneli, T. Vayer, and N. Courty. Semi-relaxed gromov-wasserstein divergence and applications on graphs. In *International Conference on Learning Representations*, 2022.

[63] C. Vincent-Cuaz, T. Vayer, R. Flamary, M. Corneli, and N. Courty. Online graph dictionary learning. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10564–10574. PMLR, 18–24 Jul 2021.

[64] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[65] H. Xu. Gromov-wasserstein factorization models for graph clustering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 6478–6485, 2020.

[66] H. Xu, D. Luo, and L. Carin. Scalable gromov-wasserstein learning for graph partitioning and matching. *Advances in neural information processing systems*, 32:3052–3062, 2019.

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

[68] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pages 5453–5462. PMLR, 2018.

[69] P. Yanardag and S. Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1365–1374, 2015.

[70] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31, 2018.

[71] M. Zhang, Z. Cui, M. Neumann, and Y. Chen. An end-to-end deep learning architecture for graph classification. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] Every key points are encapuslated in the contributions part at the end of the introduction.
   (b) Did you describe the limitations of your work? [Yes] Yes the limitations of our work mostly lie in its computational complexity and the lack of a current implementation designed for GPU parallelization of the FGW distance solver in section 2.1 and 3.2.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] Indeed the full set of assumptions of all theoretical results are in the paper. Few technical definitions which are common in the OT litterature have been reported to the supplementary material.
   (b) Did you include complete proofs of all theoretical results? [Yes] A simple intuition for Lemma 1 is given in the paper and more details are provided in the supplementary.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The code is provided in the supplementary material. A read me has been made to reproduce these experiments more easily.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] For each kind of experiments most settings are provided in the core of the paper (see section 3). Complementary details on the experiments are reported in the supplementary material
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Yes, each tables and figures contain such information on the distributions when relevant.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] Our code mostly depends on the POT and Pytorch librairies which are quoted in section 2.2
   (b) Did you mention the license of the assets? [Yes] Those assets have been following the format prescribed in their respective websites/papers.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Some implementations of the methods reproduced in the benchmark section 3.2 depend on assets which were not used initially for our own method. Therefore we quoted them in the supplementary
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] Used datasets are public ones, we quoted their designers as sign of consent.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] Used datasets are well-known public ones that do not contain such content.

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

6 Supplementary

6.1 Notations

An undirected attributed graph $\mathcal{G}$ with n nodes can be modeled in the OT context as a tuple $(\boldsymbol{C}, \boldsymbol{F}, \boldsymbol{h})$, where $\boldsymbol{C} \in \mathbb{S}_n(\mathbb{R})$ is a matrix encoding relationships between nodes, $\boldsymbol{F} = (\boldsymbol{f}_1, ..., \boldsymbol{f}_n)^\top \in \mathbb{R}^{n \times d}$ is a node feature matrix and $\boldsymbol{h} \in \Sigma_n$ is a vector of weights modeling the relative importance of the nodes within the graph (Figure 1 of the main paper). *We always assume in the following that values in $\boldsymbol{C}$ and $\boldsymbol{F}$ are finite*. Let us now consider two such graphs $\mathcal{G} = (\boldsymbol{C}, \boldsymbol{F}, \boldsymbol{h})$ and $\overline{\mathcal{G}} = (\overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \overline{\boldsymbol{h}})$, of respective sizes n and $\overline{n}$ (with possibly $n \neq \overline{n}$). The Fused Gromov-Wasserstein (FGW) distance is defined for $\alpha \in [0, 1]$ as [58, 59]:

$$\mathrm{FGW}_\alpha(\boldsymbol{C}, \boldsymbol{F}, \boldsymbol{h}, \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \overline{\boldsymbol{h}}) = \min_{\boldsymbol{T} \in \mathcal{U}(\boldsymbol{h}, \overline{\boldsymbol{h}})} \mathcal{E}_\alpha^{FGW}(\boldsymbol{C}, \boldsymbol{F}, \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \boldsymbol{T}) \tag{4}$$

with $\mathcal{U}(\boldsymbol{h}, \overline{\boldsymbol{h}}) := \{\boldsymbol{T} \in \mathbb{R}_+^{n \times \overline{n}} | \boldsymbol{T}\mathbf{1}_{\overline{n}} = \boldsymbol{h}, \boldsymbol{T}^\top \mathbf{1}_n = \overline{\boldsymbol{h}}\}$, the set of admissible coupling between $\boldsymbol{h}$ and $\overline{\boldsymbol{h}}$. For any $\boldsymbol{T} \in \mathcal{U}(\boldsymbol{h}, \overline{\boldsymbol{h}})$, the FGW cost $\mathcal{E}_\alpha^{FGW}$ can be decomposed as

$$\mathcal{E}_\alpha^{FGW}(\boldsymbol{C}, \boldsymbol{F}, \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \boldsymbol{T}) = \alpha \mathcal{E}^{GW}(\boldsymbol{C}, \overline{\boldsymbol{C}}, \boldsymbol{T}) + (1 - \alpha)\mathcal{E}^{W}(\boldsymbol{F}, \overline{\boldsymbol{F}}, \boldsymbol{T}) \tag{5}$$

which respectively refers to a Gromov-Wasserstein matching cost $\mathcal{E}^{GW}$ between graph structures $\boldsymbol{C}$ and $\overline{\boldsymbol{C}}$ reading as

$$\mathcal{E}^{GW}(\boldsymbol{C}, \overline{\boldsymbol{C}}, \boldsymbol{T}) = \sum_{ijkl} (C_{ij} - \overline{C}_{kl})^2 T_{ik} T_{jl} \tag{6}$$

and a Wasserstein matching cost $\mathcal{E}^{W}$ between nodes features $\boldsymbol{F}$ and $\overline{\boldsymbol{F}}$,

$$\mathcal{E}^{W}(\boldsymbol{F}, \overline{\boldsymbol{F}}, \boldsymbol{T}) = \sum_{ik} \|\boldsymbol{f}_i - \overline{\boldsymbol{f}}_k\|_2^2 T_{ik} \tag{7}$$

6.2 Theoretical results

Preliminaries. Given two graphs $\mathcal{G}$ and $\overline{\mathcal{G}}$, we first provide a reformulation of each matching costs $\mathcal{E}^{GW}$ and $\mathcal{E}^{W}$ through matrix operations which will facilitate the readability of our proof.

By first expanding the GW matching cost given in 6 and using the marginal constraints over $\boldsymbol{T} \in \mathcal{U}(\boldsymbol{h}, \overline{\boldsymbol{h}})$, $\mathcal{E}^{GW}$ can be expressed as

$$\begin{aligned}
\mathcal{E}^{GW}(\boldsymbol{C}, \overline{\boldsymbol{C}}, \boldsymbol{T}) &= \sum_{ij} C_{ij}^2 h_i h_j + \sum_{kl} \overline{C}_{kl}^2 \overline{h}_k \overline{h}_l - 2 \sum_{ijkl} C_{ij} \overline{C}_{kl} T_{ik} T_{jl} \\
&= \langle \boldsymbol{C}^2, \boldsymbol{h}\boldsymbol{h}^\top \rangle + \langle \overline{\boldsymbol{C}}^2, \overline{\boldsymbol{h}}\overline{\boldsymbol{h}}^\top \rangle - 2\langle \boldsymbol{T}^\top \boldsymbol{C} \boldsymbol{T}, \overline{\boldsymbol{C}} \rangle \\
&= \langle \boldsymbol{T}^\top \boldsymbol{C}^2 \boldsymbol{T}, \mathbf{1}_{\overline{n} \times \overline{n}} \rangle + \langle \boldsymbol{T} \overline{\boldsymbol{C}}^2 \boldsymbol{T}^\top, \mathbf{1}_{n \times n} \rangle - 2\langle \boldsymbol{T}^\top \boldsymbol{C} \boldsymbol{T}, \overline{\boldsymbol{C}} \rangle
\end{aligned} \tag{8}$$

where power operations are applied element-wise and $\mathbf{1}^{p \times q}$ is the matrix of ones of size $p \times q$ for any integers p and q.

Then through similar operations $\mathcal{E}^{W}$ can be expressed as

$$\begin{aligned}
\mathcal{E}_\alpha(\boldsymbol{C}, \boldsymbol{F}, \boldsymbol{h}, \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \overline{\boldsymbol{h}}, \boldsymbol{T}) &= \sum_{i} \|\boldsymbol{f}_i\|_2^2 h_i + \sum_{k} \|\overline{\boldsymbol{f}}_k\|_2^2 \overline{h}_k - 2 \sum_{ik} \langle \boldsymbol{f}_i, \overline{\boldsymbol{f}}_k \rangle T_{ik} \\
&= \langle \boldsymbol{F}^2 \mathbf{1}_d, \boldsymbol{h} \rangle + \langle \overline{\boldsymbol{F}}^2 \mathbf{1}_d, \overline{\boldsymbol{h}} \rangle - 2\langle \boldsymbol{F} \overline{\boldsymbol{F}}^\top, \boldsymbol{T} \rangle \\
&= \langle \boldsymbol{T}^\top \boldsymbol{F}^2, \mathbf{1}_{\overline{n} \times d} \rangle + \langle \boldsymbol{T} \overline{\boldsymbol{F}}^2, \mathbf{1}_{n \times d} \rangle - 2\langle \boldsymbol{F}^\top \boldsymbol{T}, \overline{\boldsymbol{F}}^\top \rangle
\end{aligned} \tag{9}$$

Lemma 1 *The TFGW embeddings are invariant to strong isomorphism.*

Proof of Lemma 1. First, as our TFGW embeddings can operate after embedding the nodes feature of any graph, let us also introduce such an application. Given any feature matrix $\boldsymbol{F} = (\boldsymbol{f}_1, ..., \boldsymbol{f}_n)^\top \subset \mathbb{R}^{n\times d}$, we denote by $\phi : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d'}$ an application such that $\phi(\boldsymbol{F}) = (\varphi(\boldsymbol{f}_1), ..., \varphi(\boldsymbol{f}_n))^\top$ with $\varphi : \mathbb{R}^d \to \mathbb{R}^{d'}$.

Let us now consider any pair of graphs $\mathcal{G}_1 = (\boldsymbol{C}_1, \boldsymbol{F}_1, \boldsymbol{h}_1)$ and $\mathcal{G}_2 = (\boldsymbol{C}_2, \boldsymbol{F}_2, \boldsymbol{h}_2)$ defined as in the subsection 6.1. Assume that $\mathcal{G}_1$ and $\mathcal{G}_2$ are *strongly isomorphic*. This is equivalent to assuming that they have the same number of nodes n and there exists a permutation matrix $\boldsymbol{P} \in \{0, 1\}^{n\times n}$ such that $\boldsymbol{C}_2 = \boldsymbol{P}\boldsymbol{C}_1\boldsymbol{P}^\top$, $\boldsymbol{F}_2 = \boldsymbol{P}\boldsymbol{F}_1$ and $\boldsymbol{h}_2 = \boldsymbol{P}\boldsymbol{h}_1$ [59, 14].

First observe that the application ϕ preserves the relation of strong isomorphism. Indeed, as ϕ operates on each node independently through φ, we have $\phi(\boldsymbol{F}_2) = \boldsymbol{P}\phi(\boldsymbol{F}_1)$ *i.e*,

$$\phi(\boldsymbol{F}_2) = (\varphi(\boldsymbol{F}_{2,1}), ..., \varphi(\boldsymbol{F}_{2,n})) = \boldsymbol{P}(\varphi(\boldsymbol{F}_{1,1}), ..., \varphi(\boldsymbol{F}_{1,n})) = \boldsymbol{P}\phi(\boldsymbol{F}_1) \tag{10}$$

Therefore the embedded graphs $(\boldsymbol{C}_1, \phi(\boldsymbol{F}_1), \boldsymbol{h}_1)$ and $(\boldsymbol{C}_2, \phi(\boldsymbol{F}_2), \boldsymbol{h}_2)$ are also strongly isomorphic and are associated by the same permutation $\boldsymbol{P}$ linking $\mathcal{G}_1$ and $\mathcal{G}_2$.

Let us consider any graph template $\overline{\mathcal{G}} = (\overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \overline{\boldsymbol{h}})$. We will prove now that the FGW cost from $(\boldsymbol{C}_1, \phi(\boldsymbol{F}_1), \boldsymbol{h}_1)$ to $\overline{\mathcal{G}}$ applied in $\boldsymbol{T}$ is the same than the FGW cost from $(\boldsymbol{C}_2, \phi(\boldsymbol{F}_2), \boldsymbol{h}_2)$ to $\overline{\mathcal{G}}$ applied in $\boldsymbol{P}\boldsymbol{T}$. To this end we will prove that analog relations hold for the Gromov-Wasserstein and the Wasserstein matching costs independently (in this generic scenario), then we will conclude thanks the equation 5 which expresses FGW as a linear combination between both aforementioned costs.

First using the reformulation of $\mathcal{E}^{GW}$ of equation 8, we have

$$\begin{aligned}
\mathcal{E}^{GW}(\boldsymbol{C}_1, \overline{\boldsymbol{C}}, \boldsymbol{T}) &= \langle \boldsymbol{T}^\top \boldsymbol{C}_1^2 \boldsymbol{T}, \boldsymbol{1}_{\overline{n}\times\overline{n}} \rangle + \langle \boldsymbol{T}\overline{\boldsymbol{C}}^2\boldsymbol{T}^\top, \boldsymbol{1}_{n\times n} \rangle - 2\langle \boldsymbol{T}^\top \boldsymbol{C}_1 \boldsymbol{T}, \overline{\boldsymbol{C}} \rangle \\
&= \langle \boldsymbol{T}^\top \boldsymbol{P}^\top \boldsymbol{C}_2^2 \boldsymbol{P}\boldsymbol{T}, \boldsymbol{1}_{\overline{n}\times\overline{n}} \rangle + \langle \boldsymbol{T}\overline{\boldsymbol{C}}^2\boldsymbol{T}^\top, \boldsymbol{P}^\top \boldsymbol{1}_{n\times n}\boldsymbol{P} \rangle - 2\langle \boldsymbol{T}^\top \boldsymbol{P}^\top \boldsymbol{C}_2 \boldsymbol{P}\boldsymbol{T}, \overline{\boldsymbol{C}} \rangle \\
&= \langle (\boldsymbol{P}\boldsymbol{T})^\top \boldsymbol{C}_2^2 \boldsymbol{P}\boldsymbol{T}, \boldsymbol{1}_{\overline{n}\times\overline{n}} \rangle + \langle \boldsymbol{P}\boldsymbol{T}\overline{\boldsymbol{C}}^2(\boldsymbol{P}\boldsymbol{T})^\top, \boldsymbol{1}_{n\times n} \rangle - 2\langle (\boldsymbol{P}\boldsymbol{T})^\top \boldsymbol{C}_2 \boldsymbol{P}\boldsymbol{T}, \overline{\boldsymbol{C}} \rangle \\
&= \mathcal{E}^{GW}(\boldsymbol{C}_2, \overline{\boldsymbol{C}}, \boldsymbol{P}\boldsymbol{T})
\end{aligned} \tag{11}$$

where we used $\boldsymbol{C}_1^2 = (\boldsymbol{P}^\top \boldsymbol{C}_2 \boldsymbol{P})^2 = \boldsymbol{P}^\top \boldsymbol{C}_2^2 \boldsymbol{P}$ and the invariance to permutations of $\boldsymbol{1}_{n\times n}$.

Then, for $\mathcal{E}^W$ similar operations using equation 9 and $\phi(\boldsymbol{F}_2)^2 = (\boldsymbol{P}\phi(\boldsymbol{F}_1))^2 = \boldsymbol{P}\phi(\boldsymbol{F}_1)^2$ lead to,

$$\begin{aligned}
\mathcal{E}^{W}(\phi(\boldsymbol{F}_1), \overline{\boldsymbol{F}}, \boldsymbol{T}) &= \langle \boldsymbol{T}^\top \phi(\boldsymbol{F}_1)^2, \boldsymbol{1}_{\overline{n}\times d} \rangle + \langle \boldsymbol{T}\overline{\boldsymbol{F}}^2, \boldsymbol{1}_{n\times d} \rangle - 2\langle \phi(\boldsymbol{F}_1)^\top \boldsymbol{T}, \overline{\boldsymbol{F}}^\top \rangle \\
&= \langle \boldsymbol{T}^\top \boldsymbol{P}^\top \phi(\boldsymbol{F}_2)^2, \boldsymbol{1}_{\overline{n}\times d} \rangle + \langle \boldsymbol{T}\overline{\boldsymbol{F}}^2, \boldsymbol{P}\boldsymbol{1}_{n\times d} \rangle - 2\langle \phi(\boldsymbol{F}_2)^\top \boldsymbol{P}\boldsymbol{T}, \overline{\boldsymbol{F}}^\top \rangle \\
&= \langle (\boldsymbol{P}\boldsymbol{T})^\top \phi(\boldsymbol{F}_2)^2, \boldsymbol{1}_{\overline{n}\times d} \rangle + \langle \boldsymbol{P}\boldsymbol{T}\overline{\boldsymbol{F}}^2, \boldsymbol{1}_{n\times d} \rangle - 2\langle \phi(\boldsymbol{F}_2)^\top \boldsymbol{P}\boldsymbol{T}, \overline{\boldsymbol{F}}^\top \rangle \\
&= \mathcal{E}^{W}(\phi(\boldsymbol{F}_2), \overline{\boldsymbol{F}}, \boldsymbol{P}\boldsymbol{T})
\end{aligned} \tag{12}$$

Therefore, the same result holds for FGW using 5 and equations 11 12 as

$$\begin{aligned}
\mathcal{E}_\alpha^{FGW}(\boldsymbol{C}_1, \phi(\boldsymbol{F}_1), \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \boldsymbol{T}) &= \alpha\mathcal{E}^{GW}(\boldsymbol{C}_1, \overline{\boldsymbol{C}}, \boldsymbol{T}) + (1-\alpha)\mathcal{E}^{W}(\phi(\boldsymbol{F}_1), \overline{\boldsymbol{F}}, \boldsymbol{T}) \\
&= \alpha\mathcal{E}^{GW}(\boldsymbol{C}_2, \overline{\boldsymbol{C}}, \boldsymbol{P}\boldsymbol{T}) + (1-\alpha)\mathcal{E}^{W}(\phi(\boldsymbol{F}_2), \overline{\boldsymbol{F}}, \boldsymbol{P}\boldsymbol{T}) \\
&= \mathcal{E}_\alpha^{FGW}(\boldsymbol{C}_2, \phi(\boldsymbol{F}_2), \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \boldsymbol{P}\boldsymbol{T})
\end{aligned} \tag{13}$$

Following an analog derivation than above, one can easily prove for $\boldsymbol{T} \in \mathcal{U}(\boldsymbol{h}_2, \overline{\boldsymbol{h}})$ that

$$\mathcal{E}_\alpha^{FGW}(\boldsymbol{C}_2, \phi(\boldsymbol{F}_2), \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \boldsymbol{T}) = \mathcal{E}_\alpha^{FGW}(\boldsymbol{C}_1, \phi(\boldsymbol{F}_1), \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \boldsymbol{P}^\top\boldsymbol{T}) \tag{14}$$

Using the relations 13 and 14, we will now prove the following equality

$$\mathrm{FGW}_\alpha(\boldsymbol{C}_1, \phi(\boldsymbol{F}_1), \boldsymbol{h}_1, \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \overline{\boldsymbol{h}}) = \mathrm{FGW}_\alpha(\boldsymbol{C}_2, \phi(\boldsymbol{F}_2), \boldsymbol{h}_2, \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \overline{\boldsymbol{h}}) \tag{15}$$

First of all, the existence of optimal solutions for both FGW problems is ensured by the Weierstrass theorem [49]. We denote an optimal coupling $\boldsymbol{T}_1^\star \in \mathcal{U}(\boldsymbol{h}_1, \overline{\boldsymbol{h}})$ for $\mathrm{FGW}_\alpha(\boldsymbol{C}_1, \phi(\boldsymbol{F}_1), \boldsymbol{h}_1, \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \overline{\boldsymbol{h}})$. Assume there exists an optimal coupling $\boldsymbol{T}_2^\star$ for $\mathrm{FGW}_\alpha(\boldsymbol{C}_2, \phi(\boldsymbol{F}_2), \boldsymbol{h}_2, \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \overline{\boldsymbol{h}})$ such that

$$\mathcal{E}_\alpha^{FGW}(\boldsymbol{C}_2, \phi(\boldsymbol{F}_2), \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \boldsymbol{T}_2^\star) < \mathcal{E}_\alpha^{FGW}(\boldsymbol{C}_2, \phi(\boldsymbol{F}_2), \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \boldsymbol{P}\boldsymbol{T}_1^\star) \tag{16}$$

then using the equalities 14 for the l.h.s and 13 for the r.h.s, we have

$$\mathcal{E}_\alpha^{FGW}(\boldsymbol{C}_1, \phi(\boldsymbol{F}_1), \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \boldsymbol{P}^\top \boldsymbol{T}_2^\star) < \mathcal{E}_\alpha^{FGW}(\boldsymbol{C}_1, \phi(\boldsymbol{F}_1), \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \boldsymbol{T}_1^\star) \tag{17}$$

which contradicts the optimality of $\boldsymbol{T}_1^\star$. Therefore such $\boldsymbol{T}_2^\star$ can not exist and necessarily $\boldsymbol{P}\boldsymbol{T}_1^\star$ is an optimal coupling for $\mathrm{FGW}_\alpha(\boldsymbol{C}_2, \phi(\boldsymbol{F}_2), \boldsymbol{h}_2, \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \overline{\boldsymbol{h}})$. Finally, we can conclude using the optimality of $\boldsymbol{T}_1^\star$ and $\boldsymbol{P}\boldsymbol{T}_1^\star$ for their respective FGW matching problems and the equality 13:

$$\begin{aligned} &\mathcal{E}_\alpha^{FGW}(\boldsymbol{C}_1, \phi(\boldsymbol{F}_1), \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \boldsymbol{T}_1^\star) = \mathcal{E}_\alpha^{FGW}(\boldsymbol{C}_2, \phi(\boldsymbol{F}_2), \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \boldsymbol{P}\boldsymbol{T}_1^\star) \\ \Leftrightarrow\ &\mathrm{FGW}_\alpha(\boldsymbol{C}_1, \phi(\boldsymbol{F}_1), \boldsymbol{h}_1, \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \overline{\boldsymbol{h}}) = \mathrm{FGW}_\alpha(\boldsymbol{C}_2, \phi(\boldsymbol{F}_2), \boldsymbol{h}_2, \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \overline{\boldsymbol{h}}) \end{aligned} \tag{18}$$

□

6.3 Complements on our experimental results on synthetic datasets

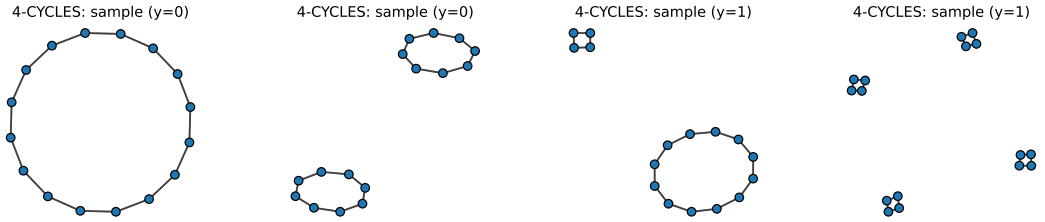

Figure 6: Few samples with different labels $y \in \{0, 1\}$ from the dataset 4-CYCLES.

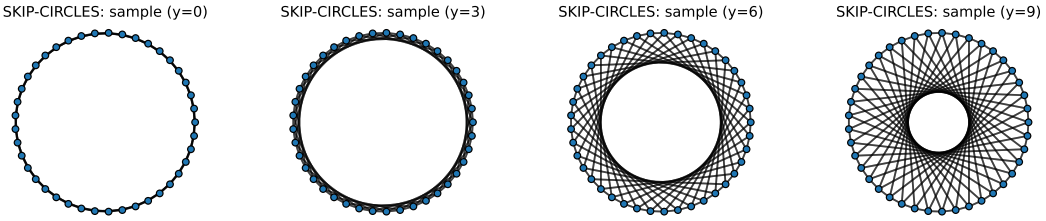

Figure 7: Unique sample from different labels $y \in \{0, 3, 6, 9\}$ corresponding respectively to $\{2, 5, 11, 16\}$ hops from the dataset SKIP-CIRCLES.

We provide here some insights and results on the synthetic datasets studied in subsection 3.1 of the main paper.

Datasets. We considered two synthetic datasets:

- 4-CYCLES [34, 44] contains graphs with (possibly) disconnected cycles where the label y_i is the presence of a cycle of length 4, as illustrated in Figure 6.
- SKIP-CIRCLES [12] contains circular graphs with skip links and the labels (10 classes) are the lengths of the skip links among $\{2, 3, 4, 5, 6, 9, 11, 12, 13, 16\}$, as illustrated in figure 7.

Details on the experiments reported in the paper. The experiments reported in the main paper, focus on the adjacency matrices for $\boldsymbol{C}_i$ for which two flavours of TFGW are investigated: 1) in TFGW-fix we fix the templates by sampling *one template per class* from the training dataset (this can be seen as a simpler FGW feature extraction); 2) for TFGW we learn the templates from the training data (as many as the number of classes). For both methods we used the FGW fixing $\alpha = 1$ (i.e the GW distance) as degrees are not discriminant for these datasets. We fixed for both methods the same MLP learned to predict labels from the TFGW embeddings. This MLP (ψ_v) contains 2 layers of 128 hidden units each, with ReLU activations. The models are learnt for 1000 epochs using

Table 5: Statistics on real datasets considered in our benchmark.

datasets	features	#graphs	#classes	mean #nodes	min #nodes	max #nodes	median #nodes
MUTAG	$\{0..6\}$	188	2	17.93	10	28	17.5
PTC-MR	$\{0,..,17\}$	344	2	14.29	2	64	13
ENZYMES	$\mathbb{R}^{18}$	600	6	32.63	2	126	32
PROTEIN	$\mathbb{R}^{29}$	1113	2	29.06	4	620	26
NCI1	$\{0,...,36\}$	4110	2	29.87	3	111	27
IMDB-B	None	1000	2	19.77	12	136	17
IMDB-M	None	1500	3	13.00	7	89	10
COLLAB	None	5000	3	74.5	32	492	52

Adam optimizer with an initial learning rate of 0.01 and taking the whole train dataset as a batch. For DropGIN [44] and GIN [67] we replicated their experiments taking the same settings than the ones described by [44]. For 4-CYCLES, they used 4 GIN layers composed of 2 layers each with 16 hidden units and batch normalization. For SKIP-CIRCLES, they used 9 similar GIN layers except that for each GIN layer the number of layer-wise hidden units is set to 32 instead of 16. Finally, as prescribed by [44] we set the number of runs to $r = 50$ and the node dropout probability $p = \frac{2}{m}$ where m is the mean number of nodes in the graphs in the dataset. These methods also use the Adam optimizer with an initial learning rate of 0.01, where the learning rate by 0.5 every 50 epochs during 1000 epochs. Switching their optimization scheme to the ones used for TFGW did not change the reported results so we kept the one from the original paper.

Additional experiments. To further emphasize the discriminative power of the TFGW embeddings, we report here additional experiments conducted on the SKIP-CIRCLES simulated datasets. For adjacency (ADJ) and shortest-path (SP) matrices as $\boldsymbol{C}_i$, instead of using one template per class ($K = 10$) as reported in the main paper for the sake of conciseness, we stress the TFGW-fix models by learning on $K \in \{2, ..., 10\}$ fixed templates sampled from the dataset. We report in Figure 8, the test accuracies averaged over 10 simulations with the same other settings than for the previously reported experiments. The averaged accuracies are illustrated in bold, while the intervals between the minimum and the maximum accuracy across runs is illustrated with a lower intensity. We can see that both methods perfectly distinguish the classes using at most 3 templates. Moreover only 2 suffice to achieve such performance using SP matrices, which is not the case for ADJ matrices. These results support our detailed analyzes in section 3 of the paper where the SP matrices are shown to better perform than ADJ ones, when no pre-processing of the node features is used.

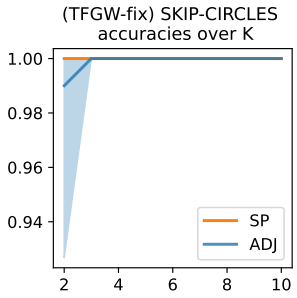

Figure 8: Test accuracies on SKIP-CIRCLES of TFGW-fix for $K \in \{2, ..., 10\}$.

6.4 Complement on the experiments on real datasets

We detail here few aspects of our experiments on real datasets reported in subsection 3.2 and 3.3 of the main paper. We first report some statistics on these datasets in Table 5.

Graph Classification benchmark. We complete here the description of the settings and the validated hyper-parameters that we used in our benchmark whose results are reported in Table 2 of the main paper.

For our method TFGW, we validate the number of templates K in $\{\beta|\mathcal{Y}|\}_\beta$, with $\beta \in \{2, 4, 6, 8\}$ and $|\mathcal{Y}|$ the number of classes. Only for ENZYMES with 6 classes of 100 graphs each, we validate $\beta \in \{1, 2, 3, 4\}$. All parameters of our TFGW layers are learned, namely the templates structure $\overline{\boldsymbol{C}}_k$, feature matrix $\overline{\boldsymbol{F}}_k$, the weights $\overline{\boldsymbol{h}}_k$, and finally a single trade-off parameter α. Moreover these templates are initialized by randomly sampling from the train dataset, a same number of graphs for each class. We also learn $\phi_{\boldsymbol{u}}$ taken as a GIN architecture [67] composed of $L = 2$ GIN layers aggregated using the Jumping Knowledge (concatenation) scheme [68]. Every GIN layer is a MLP of 2 layers with batch normalization, whose number of units is validated in $\{16, 32\}$ for bioinformatics datasets and fixed to 64 for social network datasets, as in [67]. For predictions, the same MLP ψ_v than for the experiments on synthetic datasets is used. Finally a dropout technique is applied to ψ_v

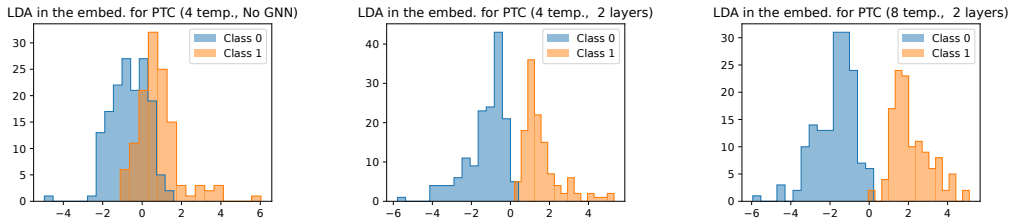

Figure 9: LDA 1D projections of the distance embeddings for different models learned on PTC.

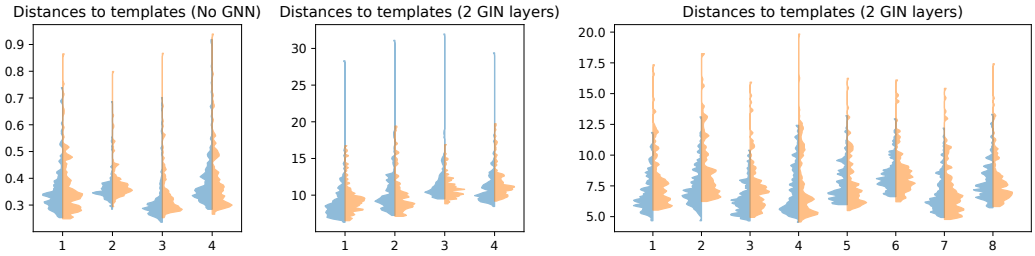

Figure 10: Distributions of the distance to the templates fo each templates for different models learned on PTC.

with a rate validated in $\{0, 0.2, 0.5\}$. We learn our models over 500 epochs using Adam optimizer with an initial learning rate of 0.01 and a batch size of 128.

For OT-GNN [11] which is the approach the most similar to TFGW, the exact same settings and validated hyper-parameters are considered. For WEGL [27], we validated the number of layers $L \in \{1, 2, ..., 6\}$ for their non-parametric embeddings, then learnt Random Forest classifiers shown to achieve the best results on average across datasets, whose hyper-parameters are validated as in the original paper. For GIN [67] and DropGIN [44] which share the same architecture than TFGW except for the pooling strategy (sum aggregation instead of FGW distances), we also validated the same hyper-parameters than for TFGW. For PPGN [36], we validated the same 3 architectures than authors, as the learning rates taken in $\{5.10^{-5}, 10^{-4}, 5.10^{-4}, 10^{-3}\}$ and its decay every 20 epochs taken in $\{0.5, 1\}$, while fixing a small batch size of 8 as motivated by authors' implementations. For Patchy-SAN [42], we validated their receptive field parameter in $k \in \{5, 10, 10^E\}$ considering the same other settings than the authors. For DIFFPOOL [70], following the discussion of the authors, we validated for their default version the number of pooling layer in $\{1, 2\}$, the clustering ratios in $\{10\%, 25\%\}$. Finally for the kernel methods, we cross validated the SVM parameters $C \in \{10^{-7}, 10^{-6}, ..., 10^{7}\}$ and $\gamma \in \{2^{-10}, 2^{-9}, ..., 2^{10}\}$ using the scikit-learn implementation [9]. Then for the FGW kernel [59], we validated 15 values of the trade-off parameter α via a logspace search in $(0, 0.5)$ and symmetrically $(0.5, 1)$. For WL [52] and WWL [56], the number of WL step is validated in $\{1, ..., 10\}$ while taking the discrete and continuous versions of the WL refinement suggested in [56].

Additional visualization of the TFGW embeddings. In this paragraph we complete the analysis of our TFGW embeddings detailed in the section 3.3 of the main paper. Especially, we illustrate the LDA in Figure 9 and the distributions 10 of our distance embeddings learned on PTC with $L = 0$ and $L = 2$, and the number of templates K varying in $\{4, 8\}$. Note that these embeddings are the same than studied thanks to a PCA in the main paper. First, the figure 9 supports our conclusions regarding the separability of our embeddings, which becomes more linear when increasing the number of GIN layers from $L = 0$ to $L = 2$ and the number of templates from $K = 4$ to $K = 8$. Then, the distribution of the distances in figure 10 exhibits that the discrimination between samples of different classes is achieved through the modes of these distributions. One can also notice that the range of distances increases considerably from $L = 0$ to $L = 2$. This coincides with the fact that the learned templates are extreme points in the embedding, as illustrated in the main paper, which might encode "exaggerated" features in order to maximize the margin between classes in the embedding. An instance of such learned templates are illustrated in figure 11. By comparing these templates (on

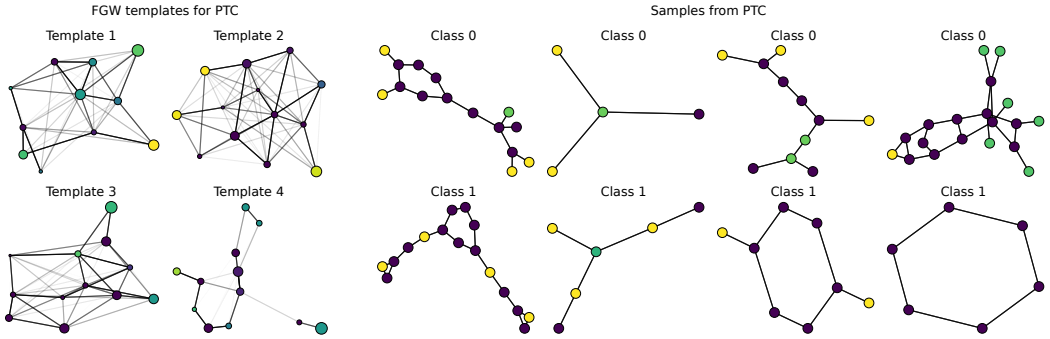

Figure 11: Illustration of the templates learned on PTC with $K = 4$ and $L = 0$ (on the left side), and some samples from this dataset (on the right sight). The graph structures are represented using the entries of $\overline{\boldsymbol{C}}_k$ (resp. $\boldsymbol{C}_i$) as repulsive strength and the corresponding edges are colored in shades of grey (black being the maximum). The node colors are computed based on their features $\overline{\boldsymbol{F}}_k$ (resp. $\boldsymbol{F}_i$). The nodes size are made proportional to the weights $\overline{\boldsymbol{h}}_k$ (resp. $\boldsymbol{h}_i$).

the left) with samples from the dataset (on the right), we can clearly see that the learned templates do not represent realistic graphs from the data. Such behavior was to be expected in our end-to-end framework where the prediction task is achieved by a MLP with non-linear activations. However we believe that our promising results achieved thanks to our TFGW modeling can open the door to novel and hopefully more interpretable end-to-end architectures.

Sensitivity of GIN to the number of layers. We aim here to benchmark our sensitivity analysis to the number of templates and the number of GIN layers of TFGW, reported in the subsection 3.3 of the main paper. To this end, we learned GIN models with a number of GIN layers varying in $L \in \{1, 2, ..., 6\}$, using analog settings and validation than detailed in our graph classification benchmark. The test accuracies of the validated models for each L are reported in Figure 12. First, the model with $L = 4$ (default for the method [67]) leads to best performances on average. Then, no clear pattern of overfitting is observed for $L > 4$, as indeed $L = 5$ leads to worst performances but $L = 6$ leads to the second best model in this benchmark. Such behavior may come from the Jumping Knowledge scheme (with concatenation) [68] as argued by the authors.

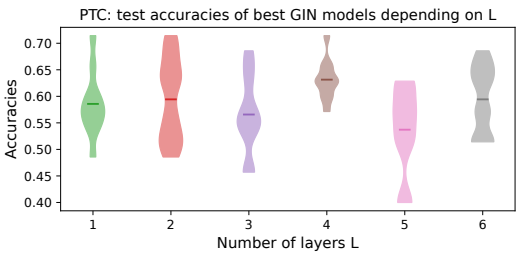

Figure 12: Test accuracies on PTC of GIN for $L \in \{1, ..., 6\}$.

Complements on runtimes. We complete in this paragraph the benchmark on runtimes provided in the subsection 3.2 of the main paper. We report for the best models selected for the benchmark in Table 2 of the main paper, their averaged prediction time per graph while using CPU and/or GPU. These were taken on CPUs (Intel Core i9-9900K CPU, 3.60 GHz) or a GPU Quatro RTX 4000. As illustrated in Table 6.4, TFGW has approximately the same or lower averaged prediction time per graph on CPU, as recent DropGIN and PPGN architectures. However, GIN, DropGIN and PPGN, get a 3-30 times speedup on GPU when TFGW is slower on GPU for PTC and 2x faster for PROTEIN (acceleration of matrix product for large graphs but overhead time for transfering between CPU and GPU for small graphs). This shows that TFGW is not yet accelerated on GPU but remains reasonable in practice, so one can still benefit from it in the GNN and MLP models. Note that the main bottleneck of the method is the OT network flow CPU solver that is called in the Conditional Gradient solver of FGW in the current implementation (see POT implementation [18]). Recent works focused on accelerating network flow algorithms on GPU which will probably be integrated soon in CUDA and will allow a similar speedup for TFGW, coupled for instance with GIN, in the future.

Table 6: Benchmark of averaged prediction time per graph (in ms) on CPU or GPU vs accuracy drop w.r.t TFGW models of compared methods for best models reported in Table 2 of the main paper.

	PTC			PROTEINS		
	CPU runtimes (ms)	GPU runtimes (ms)	Accuracy drop w.r.t TFGW (%)	CPU runtimes (ms)	GPU runtimes (ms)	Accuracy drop w.r.t TFGW (%)
(ours) TFGW	12.1	20.7	-	45.9	21.8	-
OT-GNN	7.6	8.8	4.4	27.1	11.1	4.9
GIN	0.19	0.06	9.4	2.5	0.09	6.7
DropGIN	14.3	2.1	10.1	79.8	4.7	6.0
PPGN	32.1	26.1	6.8	91.7	11.6	5.8

How does TFGW behave with other GNN architectures than GIN? Our intuition regarding this matter is that using our TFGW layer to obtain the graph representations consistently leads to better performances than simple sum pooling over graph nodes, regardless of the GNN architectures. Moreover, performances of both approaches should be correlated. To support these affirmations, we investigate here the use of Graph Attention networks [60, GAT] for graph classification, either using a simple sum pooling or using the TFGW layer. First, as the GAT architecture was investigated by its authors on node classifications and not graph classifications, we study the behavior of GAT layers within a comparable framework than the one proposed by GIN, on 3 bioinformatic datasets. We use the same Jumping Knowledge scheme than GIN, i.e we concatenate features produced at each GAT layer, then we sum them across all nodes to get the graph representation fed to the finale classifier $\psi_{\boldsymbol{v}}$. To fit to the benchmark reported in Table 2 of the main paper, we validate the features dimension in $\{16, 32\}$ (using a single attention head in GAT layers) and the dropout ratios applied to $\psi_{\boldsymbol{v}}$ in $\{0, 0.2, 0.5\}$, while considering $L \in \{1, 2, 3, 4\}$ GAT layers. The results in terms of accuracy are reported in Table 7. GAT provides competitive performances on MUTAG and PROTEIN datasets compared to GIN, while GIN largely outperforms GAT on the PTC dataset. Also, it seems harder to find a consensus across datasets on L for GAT-based architectures compared to GIN's ones. Finally, we observed that using multi-head attention was prone to overfitting and led to lower performances on these graph classification tasks, so such suggestions from GAT's authors on node classification tasks seem to not hold for these graph-level tasks.

Table 7: Test set classification accuracies from 10-fold CV. The first (resp. second) best performing method is highlighted in bold (resp. underlined).

	MUTAG	PTC	PROTEIN
GAT (L=1)	89.4(1.0)	53.1(3.4)	**77.8(1.7)**
GAT (L=2)	<u>91.1</u>(<u>2.5</u>)	52.0(4.0)	76.3(3.1)
GAT (L=3)	88.9(1.7)	<u>53.4</u>(<u>3.8</u>)	75.9(2.3)
GAT (L=4)	**91.2(2.8)**	50.9(5.8)	<u>77.6</u>(<u>2.7</u>)
GIN	90.1(4.4)	**63.1(3.9)**	76.2(2.8)

Finally, we investigate the merge of our TFGW layer with GAT layers, as $\phi_{\boldsymbol{u}}$ to produce node embeddings. We follow an analog validation than for TFGW coupled with GIN, setting $L \in \{1, 2\}$. The results are reported in the following table. We can see that TFGW with GAT leads to competitive performances compared to TFGW with GIN, at least of MUTAG and PROTEIN. GAT clearly struggles on the PTC dataset, however with our TFGW layer instead of a sum pooling, we observe a boost of performances from 53.4% to 68.7%. Even if TFGW coupled with GIN on PTC is still considerably better than TFGW with GAT. Therefore, the choice of GNN architectures to produce node embeddings to feed to the TFGW layer matters, but the gain from using TFGW seems to be independent of the GNN architecture.

Table 8: Test set classification accuracies from 10-fold CV of the TFGW models using GAT or GIN as $\phi_{\boldsymbol{u}}$. The first (resp. second) best in bold (resp. underlined).

TFGW- $\phi_{\boldsymbol{u}}$	L	input	MUTAG	PTC	PROTEIN
GAT	2	ADJ	95.4(3.5)	68.7(5.8)	**83.4(2.8)**
		SP	<u>96.2</u>(<u>3.0</u>)	67.9(5.8)	82.6(2.9)
	1	ADJ	94.8(3.1)	66.9(5.4)	82.1(3.3)
		SP	**96.4(3.3)**	68.3(6.0)	82.3(3.1)
GIN	2	ADJ	**96.4(3.3)**	**72.4(5.7)**	<u>82.9</u>(<u>2.7</u>)
		SP	94.8(3.5)	70.8(6.3)	82.0(3.0)
	1	ADJ	94.8(3.1)	68.7(5.8)	81.5(2.8)
		SP	95.4(3.5)	<u>70.9</u>(<u>5.5</u>)	82.1(3.4)