Template based Graph Neural Network with Optimal Transport Distances
Cédric Vincent-Cuaz, Rémi Flamary, Marco Corneli, Titouan Vayer, Nicolas Courty

To cite this version:
Cédric Vincent-Cuaz, Rémi Flamary, Marco Corneli, Titouan Vayer, Nicolas Courty. Template based Graph Neural Network with Optimal Transport Distances. NeurIPS 2022 – 36th Conference on Neural Information Processing Systems, Nov 2022, New Orleans, United States. hal-03839517

HAL Id: hal-03839517
https://hal.science/hal-03839517
Submitted on 4 Nov 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
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 \(\ddag\), e.g. image analysis \(\ddag\), brain connectivity \(\ddag\), biological compounds \(\ddag\) or social networks \(\ddag\), to name a few. Various methodologies approach the inherent complexity of those data, such as signal processing \(\ddag\).
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) [61,56]. 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 $C$ that encodes the distance/similarity between every pairs of nodes (e.g. adjacency, shortest path) and a \textit{weight vector $h$} 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 (\textit{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 $\Sigma_n := \{ h \in \mathbb{R}_+^n \mid \sum_i h_i = 1 \}$ 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 $G$ with $n$ nodes can be modeled in the OT context as a tuple $(C, F, h)$, where $C \in \mathbb{S}_n(\mathbb{R})$ is a matrix encoding relationships between nodes, $F = (f_1, \ldots, f_n)^\top \in \mathbb{R}^{n \times d}$ is a node feature matrix and $h \in \Sigma_n$ 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 ($h = 1_n/n$). The matrix $C$ 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 $(C, F, h)$ and $(\overline{C}, \overline{F}, \overline{h})$, 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]:

$$\text{FGW}_\alpha(C, F, h, \overline{C}, \overline{F}, \overline{h}) = \min_{T \in \mathcal{U}(h, \overline{h})} \sum_{i,j,k,l} (\alpha(C_{ij} - \overline{C}_{kl})^2 + (1 - \alpha)\|f_i - \overline{f}_k\|_2^2) T_{ik}T_{jl} \quad (1)$$

where $\mathcal{U}(h, \overline{h}) := \{ T \in \mathbb{R}_+^{n \times \overline{n}} | T1\overline{n} = h, T\top1_n = \overline{h} \}$ is the set of admissible coupling between $h$ and $\overline{h}$. FGW aims at finding an optimal coupling $T^\star$ by minimizing a trade-off cost, via $\alpha$, 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 $T^\star$ acts as a soft matching of the nodes, which tends to associate pairs of nodes that have similar pairwise relations in $C$ and $\overline{C}$ (GW cost), and similar features in $F$ and $\overline{F}$ (W cost).

Interestingly, FGW defines a metric on the space of attributed graphs. In particular, if $C$ and $\overline{C}$ 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 [34, 56] will have a zero FGW distance for any $\alpha$ and, more importantly, $\text{FGW}_\alpha = 0$ implies that the graphs are strongly isomorphic. When $C, \overline{C}$ 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, 13].

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 $O(n^2\overline{n} + \overline{n}^2n)$ [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].

\footnote{Two graphs $(C, F, h)$ and $(\overline{C}, \overline{F}, \overline{h})$ are strongly isomorphic if $n = \overline{n}$ and there exists a permutation matrix $P \in \{0, 1\}^{n \times n}$ such that $\overline{C} = PCP\top$, $\overline{F} = PF$ and $\overline{h} = Ph$.}
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 \((C, F, h)\) as input and computes its FGW distances to a list of \(K\) template graphs \(\mathcal{G} := \{(C_k, F_k, \bar{h}_k)\}_{k \in [K]}\) as follows:

\[
\text{TFGW}_{\mathcal{G}, \alpha}(C, F, h) := [\text{FGW}_\alpha (C, F, h, C_k, F_k, \bar{h}_k)]_{k=1}^K
\]

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 \(\mathcal{G}\) in a supervised manner. In the same way, we also learn the trade-off parameter \(\alpha\) 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 \(C_i\) 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 parameters (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 classification. In order to enhance the discriminating power of the model, we propose to put a GNN (denoted by \(\phi_u\) and parametrized by \(u\)) on top of the TFGW layer. We assume in the remainder that this GNN model \(\phi_u\) is injective in order to preserve isomorphism relations between graphs (see \([67]\) for more details). With a slight abuse of notation, we write \(\phi_u(F)\) 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 \(\{\mathcal{G}_i = (C_i, F_i, h_i)\}_{i \in [I]}\) 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 \(\phi_u\), then use the TFGW layer to represent the graphs as vectors in \(\mathbb{R}^K\). Finally we use the final MLP model \(\psi_v : \mathbb{R}^K \rightarrow \mathcal{Y}\) parameterized by \(v\), 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:

\[
\min_{u, v, \{(C_k, F_k, \bar{h}_k)\}, \alpha} \frac{1}{I} \sum_{i=1}^I \mathcal{L} \left( y_i, \psi_v \left( \text{TFGW}_{\mathcal{G}, \alpha}(C_i, \phi_u(F_i), h_i) \right) \right).
\]

**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 \(\alpha\). The gradients are obtained using the Envelop Theorem \([1]\) allowing to keep \(T^*\) constant. We used Pytorch \([45]\) to implement the model. The template structure \(C_k\), node weights \(\bar{h}_k\) and \(\alpha\) are updated with a projected gradient respectively on the set of symmetric matrices \(S_{n_k}([R_+])\) \((\mathbb{S}_{n_k}([0, 1]))\) when \(C_i\) are adjacency matrices), the simplex \(\Sigma_{n_k}\) and \([0, 1]\). The projection onto the probability simplex of the node weights leads to sparse solutions \([16]\), therefore
the size of each \((C_k, F_k, h_k)\) 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 \(\phi_u\) 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 \(T^\star\) resulting from TFGW between \((C_i, \phi_u(F_i), h_i)\) and the template \((C_k, F_k, h_k)\), will encode correspondences 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 \(h_k\) 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 \(\{2, 3, 4, 5, 6, 9, 11, 12, 13, 16\}\).

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 \(C_i\) 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)
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.

### 3.2 Graph classification benchmark

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, NC11 [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] and the Wasserstein WL kernel [58 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] 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 [52], DIFFPOOL [70], PPGN [35], 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 \( C_i \). 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 \( \{ \beta | Y \} \), with \( \beta \in \{ 2, 4, 6, 8 \} \) and \( |Y| \) 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 \( \phi_u \) 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] [56], 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 \( C_i \) 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.

### 3.3 Ablation study, sensitivity analysis and discussions

In this section we inspect the role of some of the model parameters (\( \alpha \) in FGW, weights estimation in the templates, depth of the GNN \( \phi_u \)) in terms of the classification performance. To this end, we first conduct on all datasets an ablation study on the graph template weights \( \mathbf{F}_k \) and the number of GIN layers in \( \phi_u \). Then, we take a closer look at the estimated trade-off parameters \( \alpha \) 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 \( C_i \), we learn the distance layers either directly, on the raw data \( (i.e. L = 0, \phi_u = \text{id}) \), or after embedding the data with \( (L = 1) \) or \( (L = 2) \) GIN layers. For \( L = 0 \) we either fix the graph template weights \( \mathbf{F}_k \)
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 \( \phi_u \) 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 \( \alpha \) 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 \( \alpha \) 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, \( \alpha \) parameters are strictly between 0 and 1. One can notice the variances of those distributions illustrating the non-uniqueness
of this trade-off parameter coming from the non-convexity of our optimization problem (a given value of $\alpha$ 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 envelope" 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 $\phi_u$, 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.

### 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. 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 for sub-graph detection.

5 Acknowledgements

This work is partially funded through the projects OTTOPIA ANR20-CHIA-0030 and 31A Côte d’Azur Investments ANR-19-P31A-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’École 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


[33] A. Leman. The reduction of a graph to canonical form and the algebra which appears therein. 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 encapsulated 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 libraries 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 \( G \) with \( n \) nodes can be modeled in the OT context as a tuple \((C, F, h)\), where \( C \in S_n(\mathbb{R}) \) is a matrix encoding relationships between nodes, \( F = (f_1, \ldots, f_p) \) is a node feature matrix and \( h \in \Sigma_n \) 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 \( C \) and \( F \) are finite. Let us now consider two such graphs \( G = (C, F, h) \) and \( \overline{G} = (\overline{C}, \overline{F}, \overline{h}) \), of respective sizes \( n \) and \( \pi \) (with possibly \( n \neq \pi \)). The Fused Gromov-Wasserstein (FGW) distance is defined for \( \alpha \in [0, 1] \) as \([58, 59]\):

\[
\text{FGW}_\alpha(C, F, h, \overline{C}, \overline{F}, \overline{h}) = \min_{T \in \mathcal{U}(h, \overline{h})} \mathcal{E}_\alpha^{\text{FGW}}(C, F, \overline{C}, \overline{F}, T)
\]

with \( \mathcal{U}(h, \overline{h}) := \{ T \in \mathbb{R}^{n \times \pi} | T1_\pi = h, T^\top 1_n = \overline{h} \} \), the set of admissible coupling between \( h \) and \( \overline{h} \). For any \( T \in \mathcal{U}(h, \overline{h}) \), the FGW cost \( \mathcal{E}_\alpha^{\text{FGW}} \) can be decomposed as

\[
\mathcal{E}_\alpha^{\text{FGW}}(C, F, \overline{C}, \overline{F}, T) = \alpha \mathcal{E}_\alpha^{\text{GW}}(C, \overline{C}, T) + (1 - \alpha) \mathcal{E}_\alpha^{W}(F, \overline{F}, T)
\]

which respectively refers to a Gromov-Wasserstein matching cost \( \mathcal{E}_\alpha^{\text{GW}} \) between graph structures \( C \) and \( \overline{C} \) reading as

\[
\mathcal{E}_\alpha^{\text{GW}}(C, \overline{C}, T) = \sum_{ijkl} (C_{ij} - \overline{C}_{kl})^2 T_{ik} T_{jl}
\]

and a Wasserstein matching cost \( \mathcal{E}_\alpha^{W} \) between nodes features \( F \) and \( \overline{F} \),

\[
\mathcal{E}_\alpha^{W}(F, \overline{F}, T) = \sum_{ik} \|f_i - \overline{f}_k\|^2 T_{ik}
\]

6.2 Theoretical results

Preliminaries. Given two graphs \( G \) and \( \overline{G} \), we first provide a reformulation of each matching costs \( \mathcal{E}_\alpha^{\text{GW}} \) and \( \mathcal{E}_\alpha^{W} \) 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 \( T \in \mathcal{U}(h, \overline{h}) \), \( \mathcal{E}_\alpha^{\text{GW}} \) can be expressed as

\[
\mathcal{E}_\alpha^{\text{GW}}(C, \overline{C}, T) = \sum_{ij} C_{ij}^2 h_i h_j + \sum_{kl} C_{kl}^2 \overline{h}_k \overline{h}_l - 2 \sum_{ijkl} C_{ij} C_{kl} T_{ik} T_{jl} = \langle C^2, hh^\top \rangle + \langle \overline{C}^2, \overline{h}h^\top \rangle - 2\langle T^\top CT, \overline{C} \rangle
\]

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

Then through similar operations \( \mathcal{E}_\alpha^{W} \) can be expressed as

\[
\mathcal{E}_\alpha(C, F, h, \overline{C}, \overline{F}, \overline{h}, T) = \sum_i \|f_i\|^2 h_i + \sum_k \|\overline{f}_k\|^2 \overline{h}_k - 2 \sum_{ik} \langle f_i, \overline{f}_k \rangle T_{ik} = \langle F^2, 1_d, h \rangle + \langle \overline{F}^2, 1_d, \overline{h} \rangle - 2 \langle FF^\top, T \rangle = \langle T^\top F^2, 1_{\pi \times d} \rangle + \langle TF^2, 1_{n \times d} \rangle - 2 \langle F^\top T, F \rangle
\]

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 \( F = (f_1, \ldots, f_n)^\top \in \mathbb{R}^{n \times d} \), we denote by \( \phi : \mathbb{R}^{n \times d} \rightarrow \mathbb{R}^{n \times d'} \) an application such that \( \phi(F) = (\varphi(f_1), \ldots, \varphi(f_n))^\top \) with \( \varphi : \mathbb{R}^d \rightarrow \mathbb{R}^{d'} \).

Let us now consider any pair of graphs \( G_1 = (C_1, F_1, h_1) \) and \( G_2 = (C_2, F_2, h_2) \) defined as in the subsection 6.1. Assume that \( G_1 \) and \( G_2 \) are strongly isomorphic. This is equivalent to assuming that they have the same number of nodes \( n \) and there exists a permutation matrix \( P \in \{0, 1\}^{n \times n} \) such that \( C_2 = PC_1P^\top \), \( F_2 = PF_1 \), and \( h_2 = Ph_1 \). 

First observe that the application \( \phi \) preserves the relation of strong isomorphism. Indeed, as \( \phi \) operates on each node independently through \( \varphi \), we have \( \phi(F_2) = P\phi(F_1) \) i.e.,

\[
\phi(F_2) = \varphi(F_{2,1}), \ldots, \varphi(F_{2,n}) = P\varphi(F_{1,1}), \ldots, \varphi(F_{1,n}) = P\phi(F_1)
\]

Therefore the embedded graphs \( (C_1, \phi(F_1), h_1) \) and \( (C_2, \phi(F_2), h_2) \) are also strongly isomorphic and are associated by the same permutation \( P \) linking \( G_1 \) and \( G_2 \).

Let us consider any graph template \( \mathcal{G} = (\overline{C}, \overline{F}, \overline{h}) \). We will prove now that the FGW cost from \( (C_1, \phi(F_1), h_1) \) to \( \mathcal{G} \) applied in \( T \) is the same than the FGW cost from \( (C_2, \phi(F_2), h_2) \) to \( \mathcal{G} \) applied in \( PT \). 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 which expresses FGW as a linear combination between both aforementioned costs.

First using the reformulation of \( E^{GW} \) of equation 8 we have

\[
E^{GW}(C_1, \overline{C}, T) = (T^\top C_1^2 T, 1_{n \times n}) + (\overline{C} \overline{C}^2 \overline{T}, 1_{n \times n}) - 2(T^\top C_1^\top C_1, \overline{C})
\]

where we used \( C_1^2 = (P^\top C_2 P)^2 = P^\top C_2^2 P \) and the invariance to permutations of \( 1_{n \times n} \).

Then, for \( E^W \) similar operations using equation 9 and \( \phi(F_2)^2 = (P\phi(F_1))^2 = P\phi(F_1)^2 \) lead to,

\[
E^W(\phi(F_1), \overline{F}, T) = (T^\top \phi(F_1)^2, 1_{n \times d}) + (\overline{T} \overline{F}^2, 1_{n \times d}) - 2(\phi(F_1)^\top T, \overline{F}^\top)
\]

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

\[
E_{\alpha}^{FGW}(C_1, \phi(F_1), \overline{C}, \overline{F}, T) = \alpha E^{GW}(C_1, \overline{C}, T) + (1 - \alpha) E^W(\phi(F_1), \overline{F}, T)
\]

Following an analog derivation than above, one can easily prove for \( T \in \mathcal{U}(h_2, \overline{h}) \) that

\[
E_{\alpha}^{FGW}(C_2, \phi(F_2), \overline{C}, \overline{F}, T) = E_{\alpha}^{FGW}(C_2, \phi(F_2), \overline{C}, \overline{F}, P^\top T)
\]

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

\[
FGW_{\alpha}(C_1, \phi(F_1), h_1, \overline{C}, \overline{F}, \overline{h}) = FGW_{\alpha}(C_2, \phi(F_2), h_2, \overline{C}, \overline{F}, \overline{h})
\]

First of all, the existence of optimal solutions for both FGW problems is ensured by the Weierstrass theorem 49. We denote an optimal coupling \( T^*_1 \in \mathcal{U}(h_1, \overline{h}) \) for \( FGW_{\alpha}(C_1, \phi(F_1), h_1, \overline{C}, \overline{F}, \overline{h}) \). Assume there exists an optimal coupling \( T^*_2 \) for \( FGW_{\alpha}(C_2, \phi(F_2), h_2, \overline{C}, \overline{F}, \overline{h}) \) such that

\[
E_{\alpha}^{FGW}(C_2, \phi(F_2), \overline{C}, \overline{F}, T^*_2) < E_{\alpha}^{FGW}(C_2, \phi(F_2), \overline{C}, \overline{F}, PT^*_1)
\]
then using the equalities\cite{13} for the l.h.s and \cite{14} for the r.h.s, we have
\[ E_{\alpha}^{F_{G,W}}(C_1, \phi(F_1), \overline{C}, \overline{F}, P^T T_2^*) < E_{\alpha}^{F_{G,W}}(C_1, \phi(F_1), \overline{C}, \overline{F}, T_1^*) \] (17)
which contradicts the optimality of $T_1^*$. Therefore such $T_2^*$ can not exist and necessarily $PT_1^*$ is an optimal coupling for $F_{G,W}(C_2, \phi(F_2), h_2, \overline{C}, \overline{F}, \overline{h})$. Finally, we can conclude using the optimality of $T_1^*$ and $PT_1^*$ for their respective FGW matching problems and the equality\cite{13}
\[ E_{\alpha}^{F_{G,W}}(C_1, \phi(F_1), \overline{C}, \overline{F}, T_1^*) = E_{\alpha}^{F_{G,W}}(C_2, \phi(F_2), \overline{C}, \overline{F}, PT_1^*) \]
\[ \iff FGW_{\alpha}(C_1, \phi(F_1), h_1, \overline{C}, \overline{F}, \overline{h}) = FGW_{\alpha}(C_2, \phi(F_2), h_2, \overline{C}, \overline{F}, \overline{h}) \] (18)

\[ \Box \]

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 \cite{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 \cite{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 $C_i$ 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 ($\psi_v$) 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.

<table>
<thead>
<tr>
<th>datasets</th>
<th>features</th>
<th>#graphs</th>
<th>#classes</th>
<th>mean #nodes</th>
<th>min #nodes</th>
<th>max #nodes</th>
<th>median #nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUTAG</td>
<td>[0.6]</td>
<td>188</td>
<td>2</td>
<td>17.93</td>
<td>10</td>
<td>28</td>
<td>17.5</td>
</tr>
<tr>
<td>PTC-MR</td>
<td>[1.5, 11]</td>
<td>188</td>
<td>2</td>
<td>19.29</td>
<td>2</td>
<td>64</td>
<td>13</td>
</tr>
<tr>
<td>ENZYMES</td>
<td>[2, 10]</td>
<td>600</td>
<td>6</td>
<td>32.63</td>
<td>2</td>
<td>126</td>
<td>32</td>
</tr>
<tr>
<td>PROTEIN</td>
<td>[2, 10]</td>
<td>1113</td>
<td>2</td>
<td>29.96</td>
<td>4</td>
<td>620</td>
<td>26</td>
</tr>
<tr>
<td>NCI1</td>
<td>[0, 30]</td>
<td>4110</td>
<td>2</td>
<td>29.87</td>
<td>3</td>
<td>111</td>
<td>27</td>
</tr>
<tr>
<td>IMDb-B</td>
<td>None</td>
<td>1000</td>
<td>2</td>
<td>19.77</td>
<td>12</td>
<td>136</td>
<td>17</td>
</tr>
<tr>
<td>IMDb-M</td>
<td>None</td>
<td>1500</td>
<td>3</td>
<td>13.00</td>
<td>7</td>
<td>89</td>
<td>10</td>
</tr>
<tr>
<td>COLLAB</td>
<td>None</td>
<td>5000</td>
<td>3</td>
<td>74.5</td>
<td>32</td>
<td>492</td>
<td>52</td>
</tr>
</tbody>
</table>

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 $C_v$, 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 $K \in \{2, \ldots, 10\}$ 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.

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 $\{\beta \mid \mathcal{Y}\}$, 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 $\mathcal{C}_b$, feature matrix $\mathcal{F}_b$, the weights $\bar{h}_b$, and finally a single trade-off parameter $\alpha$. Moreover these templates are initialized by randomly sampling from the train dataset, a same number of graphs for each class. We also learn $\phi_u$ 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 $\psi_v$ than for the experiments on synthetic datasets is used. Finally a dropout technique is applied to $\psi_v$.

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

 están implementados en la tabla 5 de la página 5. 

Además de los experimentos. Para enfatizar la capacidad discriminativa de los enmascaramientos TFGW, reportamos aquí experimentos adicionales efectuados en los datasets simulados SKIP-CIRCLES. Para matrices de adyacencia (ADJ) y camino más corto (SP) como $C_v$, en vez de usar un template por clase ($K = 10$) como reportado en el documento principal por razones de concisión, nos enfocamos en los modelos TFGW-fix aprendiendo en $K \in \{2, \ldots, 10\}$ templates fijos extraídos de los datos. Reportamos en la figura 8 las precisiones promedio sobre 10 simulaciones con las mismas otras configuraciones que para los experimentos previamente reportados. Las precisiones promedio se muestran en negrita, mientras que las diferencias entre la mínima y la máxima precisión a lo largo de las corridas se muestran con una intensidad menor. Podemos ver que ambos métodos distinguen perfectamente las clases usando hasta 3 templates. Además, solo 2 suficiente para alcanzar tal desempeño usando matrices SP, lo cual no es el caso para las matrices ADJ. Estos resultados respaldan nuestros análisis detallados en la sección 3 del documento donde se muestra que las matrices SP son mejores que las ADJ en general, cuando no se procesa previamente las características de los nodos.

6.4 Complemento sobre los experimentos con datos reales

Detallamos aquí algunos aspectos de nuestros experimentos con datos reales reportados en las secciones 3.2 y 3.3 del documento principal. Primero reportamos algunas estadísticas sobre estos datasets en la tabla 5.

Bench del clasificador de grafos. Finalmente completamos aquí la descripción de las configuraciones y los parámetros de nuestro benchmark cuyos resultados se reportaron en la tabla 2 del documento principal.

Para nuestro método TFGW, validamos el número de templates $K$ en $\{\beta \mid \mathcal{Y}\}$, con $\beta \in \{2, 4, 6, 8\}$ y $|\mathcal{Y}|$ el número de clases. Sólo para ENZYMES con 6 clases de 100 grafos cada uno, validamos $\beta \in \{1, 2, 3, 4\}$. Todos los parámetros del modelo TFGW se aprenden, es decir, el structure $\mathcal{C}_b$, la matriz de características $\mathcal{F}_b$, los pesos $\bar{h}_b$, y finalmente un solo parámetro de ajuste $\alpha$. Además, estas templates son inicializadas al azar al tomar muestras de los datos de entrenamiento, un mismo número de grafos para cada clase. Asimismo, aprendemos $\phi_u$ como una arquitectura GIN, compuesta de $L = 2$ capas GIN agrupadas mediante el esquema Jumping Knowledge (concatenación) [68]. Cada capa GIN es una MLP de 2 capas con normalización de lote, cuya cantidad de unidades se valida en $\{16, 32\}$ para los datasets de bioinformática y se fija en 64 para los datasets de red social, como en [67]. Para predicciones, la misma MLP $\psi_v$ que para los experimentos en datasets sintéticos se utiliza. Finalmente, una técnica de dropout se aplica a $\psi_v$.
Figure 9: LDA 1D projections of the distance embeddings for different models learned on PTC.

Figure 10: Distributions of the distance to the templates for 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 \( L \in \{1, 2, \ldots, 6\} \) 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 \times 10^{-5}, 10^{-4}, 5 \times 10^{-5}, 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 \( k \in \{5, 10, 10^2\} \) 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 \( C \in \{10^{-7}, 10^{-6}, \ldots, 10^7\} \) and \( \gamma \in \{2^{-10}, 2^{-9}, \ldots, 2^{10}\} \) using the scikit-learn implementation [9].

Then for the FGW kernel [59], we validated 15 values of the trade-off parameter \( \alpha \) 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, \ldots, 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
FGW templates for PTC

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 $C_k$ (resp. $C_i$) 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 $F_k$ (resp. $F_i$). The nodes size are made proportional to the weights $h_k$ (resp. $h_i$).

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.

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.

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

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.

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

<table>
<thead>
<tr>
<th>Method</th>
<th>MUTAG</th>
<th>PTC</th>
<th>PROTEIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAT (L=1)</td>
<td>89.4(1.0)</td>
<td>53.1(3.4)</td>
<td>77.8(1.7)</td>
</tr>
<tr>
<td>GAT (L=2)</td>
<td>91.1(2.5)</td>
<td>52.0(4.0)</td>
<td>76.3(3.1)</td>
</tr>
<tr>
<td>GAT (L=3)</td>
<td>88.9(1.7)</td>
<td>50.9(5.8)</td>
<td>75.9(2.3)</td>
</tr>
<tr>
<td>GAT (L=4)</td>
<td>91.2(2.8)</td>
<td>51.6(5.8)</td>
<td>77.6(2.7)</td>
</tr>
<tr>
<td>GIN</td>
<td>90.1(4.4)</td>
<td>63.1(3.9)</td>
<td>76.2(2.8)</td>
</tr>
</tbody>
</table>

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 \( \psi \). 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 \( \psi \) 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.

Finally, we investigate the merge of our TFGW layer with GAT layers, as \( \phi_u \) 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.