Supplementary material: Template based Graph Neural Network with Optimal Transport Distances

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 [14, 15]:

$$\mathrm{FGW}_\alpha(\boldsymbol{C}, \boldsymbol{F}, \boldsymbol{h}, \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \overline{\boldsymbol{h}}) = \min_{\boldsymbol{T} \in \mathcal{U}(\boldsymbol{h}, \overline{\boldsymbol{h}})} \mathcal{E}_\alpha^{FGW}(\boldsymbol{C}, \boldsymbol{F}, \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \boldsymbol{T}) \tag{1}$$

with $\mathcal{U}(\boldsymbol{h}, \overline{\boldsymbol{h}}) := \{\boldsymbol{T} \in \mathbb{R}_+^{n \times \overline{n}} | \boldsymbol{T}\boldsymbol{1}_{\overline{n}} = \boldsymbol{h}, \boldsymbol{T}^\top \boldsymbol{1}_n = \overline{\boldsymbol{h}}\}$, 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

$$\mathcal{E}_\alpha^{FGW}(\boldsymbol{C}, \boldsymbol{F}, \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \boldsymbol{T}) = \alpha \mathcal{E}^{GW}(\boldsymbol{C}, \overline{\boldsymbol{C}}, \boldsymbol{T}) + (1 - \alpha)\mathcal{E}^W(\boldsymbol{F}, \overline{\boldsymbol{F}}, \boldsymbol{T}) \tag{2}$$

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

$$\mathcal{E}^{GW}(\boldsymbol{C}, \overline{\boldsymbol{C}}, \boldsymbol{T}) = \sum_{ijkl} (C_{ij} - \overline{C}_{kl})^2 T_{ik} T_{jl} \tag{3}$$

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

$$\mathcal{E}^W(\boldsymbol{F}, \overline{\boldsymbol{F}}, \boldsymbol{T}) = \sum_{ik} \|\boldsymbol{f}_i - \overline{\boldsymbol{f}}_k\|_2^2 T_{ik} \tag{4}$$

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 (3) and using the marginal constraints over $\boldsymbol{T} \in \mathcal{U}(\boldsymbol{h}, \overline{\boldsymbol{h}})$, $\mathcal{E}^{GW}$ can be expressed as

$$\begin{aligned}\mathcal{E}^{GW}(\boldsymbol{C}, \overline{\boldsymbol{C}}, \boldsymbol{T}) &= \sum_{ij} C_{ij}^2 h_i h_j + \sum_{kl} \overline{C}_{kl}^2 \overline{h}_k \overline{h}_l - 2 \sum_{ijkl} C_{ij} \overline{C}_{kl} T_{ik} T_{jl} \\ &= \langle \boldsymbol{C}^2, \boldsymbol{h}\boldsymbol{h}^\top \rangle + \langle \overline{\boldsymbol{C}}^2, \overline{\boldsymbol{h}\boldsymbol{h}}^\top \rangle - 2\langle \boldsymbol{T}^\top \boldsymbol{C}\boldsymbol{T}, \overline{\boldsymbol{C}} \rangle \\ &= \langle \boldsymbol{T}^\top \boldsymbol{C}^2 \boldsymbol{T}, \boldsymbol{1}_{\overline{n} \times \overline{n}} \rangle + \langle \boldsymbol{T}\overline{\boldsymbol{C}}^2 \boldsymbol{T}^\top, \boldsymbol{1}_{n \times n} \rangle - 2\langle \boldsymbol{T}^\top \boldsymbol{C}\boldsymbol{T}, \overline{\boldsymbol{C}} \rangle\end{aligned} \tag{5}$$

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

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

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

$$\begin{aligned}
\mathcal{E}_\alpha(\boldsymbol{C}, \boldsymbol{F}, \boldsymbol{h}, \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \overline{\boldsymbol{h}}, \boldsymbol{T}) &= \sum_i \|\boldsymbol{f}_i\|_2^2 h_i + \sum_k \|\overline{\boldsymbol{f}}_k\|_2^2 \overline{h}_k - 2\sum_{ik} \langle \boldsymbol{f}_i, \overline{\boldsymbol{f}}_k \rangle T_{ik} \\
&= \langle \boldsymbol{F}^2 \boldsymbol{1}_d, \boldsymbol{h} \rangle + \langle \overline{\boldsymbol{F}}^2 \boldsymbol{1}_d, \overline{\boldsymbol{h}} \rangle - 2 \langle \boldsymbol{F}\overline{\boldsymbol{F}}^\top, \boldsymbol{T} \rangle \\
&= \langle \boldsymbol{T}^\top \boldsymbol{F}^2, \boldsymbol{1}_{\overline{n} \times d} \rangle + \langle \boldsymbol{T}\overline{\boldsymbol{F}}^2, \boldsymbol{1}_{n \times d} \rangle - 2 \langle \boldsymbol{F}^\top \boldsymbol{T}, \overline{\boldsymbol{F}}^\top \rangle
\end{aligned} \tag{6}$$

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 $\phi : \mathbb{R}^{n \times d} \rightarrow \mathbb{R}^{n \times d'}$ an application such that $\phi(\boldsymbol{F}) = (\varphi(\boldsymbol{f}_1), ..., \varphi(\boldsymbol{f}_n))^\top$ with $\varphi : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$.

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 section 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 $\boldsymbol{h}_2 = \boldsymbol{P}\boldsymbol{h}_1$ [15, 4].

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*,

$$\phi(\boldsymbol{F}_2) = (\varphi(\boldsymbol{F}_{2,1}), ..., \varphi(\boldsymbol{F}_{2,n})) = \boldsymbol{P}(\varphi(\boldsymbol{F}_{1,1}), ..., \varphi(\boldsymbol{F}_{1,n})) = \boldsymbol{P}\phi(\boldsymbol{F}_1) \tag{7}$$

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 (2) which expresses FGW as a linear combination between both aforementioned costs.

First using the reformulation of $\mathcal{E}^{GW}$ of equation (5), we have

$$\begin{aligned}
\mathcal{E}^{GW}(\boldsymbol{C}_1, \overline{\boldsymbol{C}}, \boldsymbol{T}) &= \langle \boldsymbol{T}^\top \boldsymbol{C}_1^2 \boldsymbol{T}, \boldsymbol{1}_{\overline{n} \times \overline{n}} \rangle + \langle \boldsymbol{T}\overline{\boldsymbol{C}}^2 \boldsymbol{T}^\top, \boldsymbol{1}_{n \times n} \rangle - 2 \langle \boldsymbol{T}^\top \boldsymbol{C}_1 \boldsymbol{T}, \overline{\boldsymbol{C}} \rangle \\
&= \langle \boldsymbol{T}^\top \boldsymbol{P}^\top \boldsymbol{C}_2^2 \boldsymbol{P}\boldsymbol{T}, \boldsymbol{1}_{\overline{n} \times \overline{n}} \rangle + \langle \boldsymbol{T}\overline{\boldsymbol{C}}^2 \boldsymbol{T}^\top, \boldsymbol{P}^\top \boldsymbol{1}_{n \times n} \boldsymbol{P} \rangle - 2 \langle \boldsymbol{T}^\top \boldsymbol{P}^\top \boldsymbol{C}_2 \boldsymbol{P}\boldsymbol{T}, \overline{\boldsymbol{C}} \rangle \\
&= \langle (\boldsymbol{P}\boldsymbol{T})^\top \boldsymbol{C}_2^2 \boldsymbol{P}\boldsymbol{T}, \boldsymbol{1}_{\overline{n} \times \overline{n}} \rangle + \langle \boldsymbol{P}\boldsymbol{T}\overline{\boldsymbol{C}}^2 (\boldsymbol{P}\boldsymbol{T})^\top, \boldsymbol{1}_{n \times n} \rangle - 2 \langle (\boldsymbol{P}\boldsymbol{T})^\top \boldsymbol{C}_2 \boldsymbol{P}\boldsymbol{T}, \overline{\boldsymbol{C}} \rangle \\
&= \mathcal{E}^{GW}(\boldsymbol{C}_2, \overline{\boldsymbol{C}}, \boldsymbol{P}\boldsymbol{T})
\end{aligned} \tag{8}$$

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 (6) and $\phi(\boldsymbol{F}_2)^2 = (\boldsymbol{P}\phi(\boldsymbol{F}_1))^2 = \boldsymbol{P}\phi(\boldsymbol{F}_1)^2$ lead to,

$$\begin{aligned}
\mathcal{E}^W(\phi(\boldsymbol{F}_1), \overline{\boldsymbol{F}}, \boldsymbol{T}) &= \langle \boldsymbol{T}^\top \phi(\boldsymbol{F}_1)^2, \boldsymbol{1}_{\overline{n} \times d} \rangle + \langle \boldsymbol{T}\overline{\boldsymbol{F}}^2, \boldsymbol{1}_{n \times d} \rangle - 2 \langle \phi(\boldsymbol{F}_1)^\top \boldsymbol{T}, \overline{\boldsymbol{F}}^\top \rangle \\
&= \langle \boldsymbol{T}^\top \boldsymbol{P}^\top \phi(\boldsymbol{F}_2)^2, \boldsymbol{1}_{\overline{n} \times d} \rangle + \langle \boldsymbol{T}\overline{\boldsymbol{F}}^2, \boldsymbol{P}\boldsymbol{1}_{n \times d} \rangle - 2 \langle \phi(\boldsymbol{F}_2)^\top \boldsymbol{P}\boldsymbol{T}, \overline{\boldsymbol{F}}^\top \rangle \\
&= \langle (\boldsymbol{P}\boldsymbol{T})^\top \phi(\boldsymbol{F}_2)^2, \boldsymbol{1}_{\overline{n} \times d} \rangle + \langle \boldsymbol{P}\boldsymbol{T}\overline{\boldsymbol{F}}^2, \boldsymbol{1}_{n \times d} \rangle - 2 \langle \phi(\boldsymbol{F}_2)^\top \boldsymbol{P}\boldsymbol{T}, \overline{\boldsymbol{F}}^\top \rangle \\
&= \mathcal{E}^W(\phi(\boldsymbol{F}_2), \overline{\boldsymbol{F}}, \boldsymbol{P}\boldsymbol{T})
\end{aligned} \tag{9}$$

Therefore, the same result holds for FGW combining equations (2), (8) and (9) as

$$\begin{aligned}
\mathcal{E}_\alpha^{FGW}(\boldsymbol{C}_1, \phi(\boldsymbol{F}_1), \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \boldsymbol{T}) &= \alpha \mathcal{E}^{GW}(\boldsymbol{C}_1, \overline{\boldsymbol{C}}, \boldsymbol{T}) + (1 - \alpha) \mathcal{E}^W(\phi(\boldsymbol{F}_1), \overline{\boldsymbol{F}}, \boldsymbol{T}) \\
&= \alpha \mathcal{E}^{GW}(\boldsymbol{C}_2, \overline{\boldsymbol{C}}, \boldsymbol{P}\boldsymbol{T}) + (1 - \alpha) \mathcal{E}^W(\phi(\boldsymbol{F}_2), \overline{\boldsymbol{F}}, \boldsymbol{P}\boldsymbol{T}) \\
&= \mathcal{E}_\alpha^{FGW}(\boldsymbol{C}_2, \phi(\boldsymbol{F}_2), \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \boldsymbol{P}\boldsymbol{T})
\end{aligned} \tag{10}$$

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

$$\mathcal{E}_{\alpha}^{FGW}(\boldsymbol{C}_2, \phi(\boldsymbol{F}_2), \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \boldsymbol{T}) = \mathcal{E}_{\alpha}^{FGW}(\boldsymbol{C}_1, \phi(\boldsymbol{F}_1), \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \boldsymbol{P}^{\top}\boldsymbol{T}) \tag{11}$$

Using the relations (10) and (11), we will now prove the following equality

$$\mathrm{FGW}_{\alpha}(\boldsymbol{C}_1, \phi(\boldsymbol{F}_1), \boldsymbol{h}_1, \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \overline{\boldsymbol{h}}) = \mathrm{FGW}_{\alpha}(\boldsymbol{C}_2, \phi(\boldsymbol{F}_2), \boldsymbol{h}_2, \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \overline{\boldsymbol{h}}) \tag{12}$$

First of all, the existence of optimal solutions for both FGW problems is ensured by the Weierstrass theorem [11]. 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

$$\mathcal{E}_{\alpha}^{FGW}(\boldsymbol{C}_2, \phi(\boldsymbol{F}_2), \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \boldsymbol{T}_2^{\star}) < \mathcal{E}_{\alpha}^{FGW}(\boldsymbol{C}_2, \phi(\boldsymbol{F}_2), \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \boldsymbol{P}\boldsymbol{T}_1^{\star}) \tag{13}$$

then using the equalities (11) for the l.h.s and (10) for the r.h.s, we have

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

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 (10):

$$\begin{aligned} &\mathcal{E}_{\alpha}^{FGW}(\boldsymbol{C}_1, \phi(\boldsymbol{F}_1), \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \boldsymbol{T}_1^{\star}) = \mathcal{E}_{\alpha}^{FGW}(\boldsymbol{C}_2, \phi(\boldsymbol{F}_2), \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \boldsymbol{P}\boldsymbol{T}_1^{\star}) \\ \Leftrightarrow\ &\mathrm{FGW}_{\alpha}(\boldsymbol{C}_1, \phi(\boldsymbol{F}_1), \boldsymbol{h}_1, \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \overline{\boldsymbol{h}}) = \mathrm{FGW}_{\alpha}(\boldsymbol{C}_2, \phi(\boldsymbol{F}_2), \boldsymbol{h}_2, \overline{\boldsymbol{C}}, \overline{\boldsymbol{F}}, \overline{\boldsymbol{h}}) \end{aligned} \tag{15}$$

□

3 Complements on our experimental results on synthetic datasets

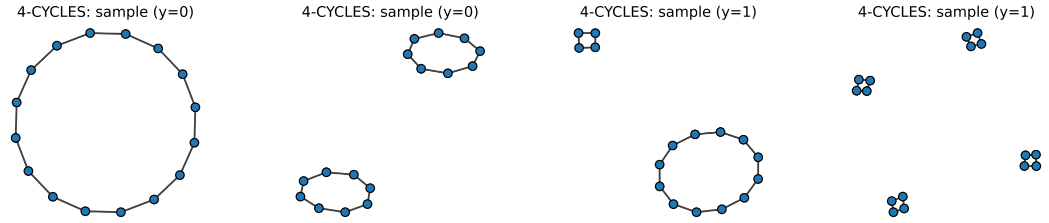

Figure 1: Few samples with different labels $y \in \{0, 1\}$ from the dataset 4-CYCLES.

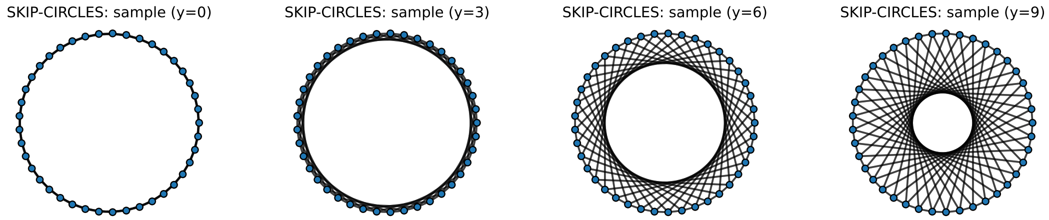
SKIP-CIRCLES: sample (y=0)
SKIP-CIRCLES: sample (y=3)
SKIP-CIRCLES: sample (y=6)
SKIP-CIRCLES: sample (y=9)

Figure 2: 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 section 3.1 of the main paper.

Datasets. We considered two synthetic datasets:

- 4-CYCLES [7, 10] contains graphs with (possibly) disconnected cycles where the label y_i is the presence of a cycle of length 4, as illustrated in Figure 1.
- SKIP-CIRCLES [3] 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 2.

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 Adam optimizer with an initial learning rate of 0.01 and taking the whole train dataset as a batch. For DropGIN [10] and GIN [17] we replicated their experiments taking the same settings than the ones described by [10]. 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 [10] 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 3, 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.

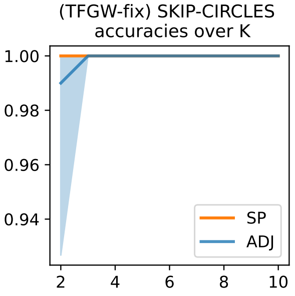

Figure 3: Test accuracies on SKIP-CIRCLES of TFGW-fix for $K \in \{2, ..., 10\}$.

4 Complement on the experiments on real datasets

We detail here few aspects of our experiments on real datasets reported in section 3.2 and 3.3 of the main paper. We first report some statistics on these datasets in Table 1.

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

Table 1: 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

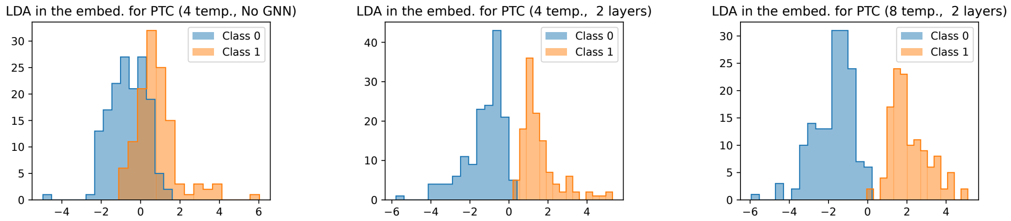

Figure 4: LDA 1D projections of the distance embeddings for different models learned on PTC.

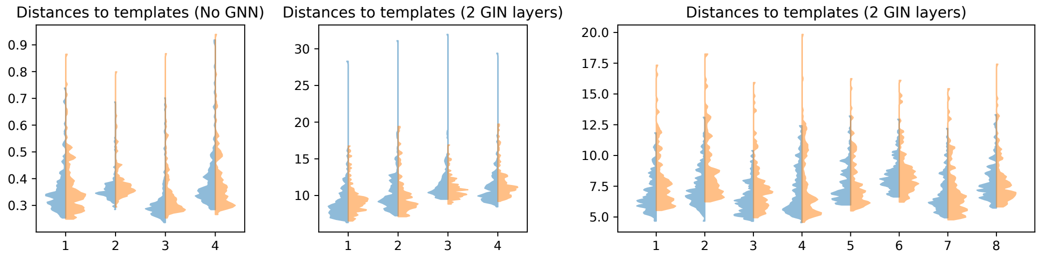

Figure 5: Distributions of the distance to the templates fo each templates for different models learned on PTC.

for each class. We also learn $\phi_{\boldsymbol{u}}$ taken as a GIN architecture [17] composed of $L = 2$ GIN layers aggregated using the Jumping Knowledge (concatenation) scheme [18]. 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 [17]. 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 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 [2] which is the approach the most similar to TFGW, the exact same settings and validated hyper-parameters are considered. For WEGL [6], 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 [17] and DropGIN [10] 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 [8], we validated the same 3 architectures than authors, as the learning rates taken in $\left\{5.10^{-5}, 10^{-4}, 5.10^{-4}, 10^{-3}\right\}$ 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 [9], we validated their receptive field parameter in $k \in \{5, 10, 10^{E}\}$ considering the same other settings than the authors. For DIFFPOOL [19], 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 [1]. Then for the FGW kernel [15], 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 [12] and WWL [13], the number of WL step is validated in $\{1, ..., 10\}$ while taking the discrete and continuous versions of the WL refinement suggested in [13].

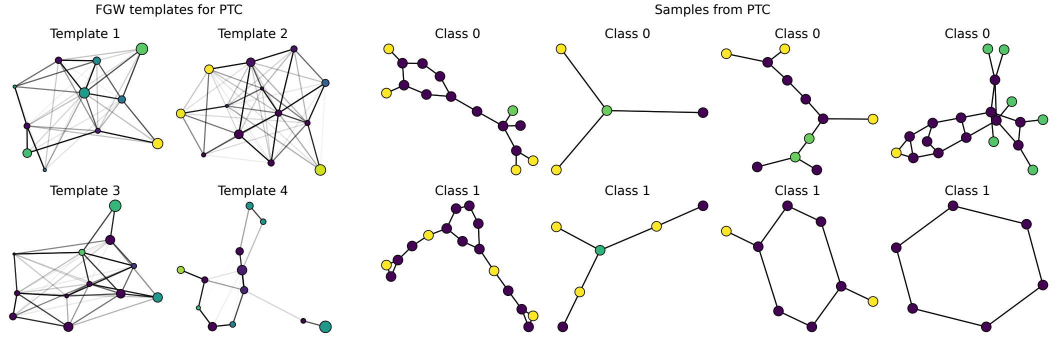

Figure 6: Illustration of the templates learned on PTC with $K = 4$ and $L = 0$ (on the left side), and some samples from this dataset (on the right sight). The graph structures are represented using the entries of $\overline{\boldsymbol{C}}_k$ (resp. $\boldsymbol{C}_i$) as repulsive strength and the 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. $\boldsymbol{h}_i$).

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 4 and the distributions 5 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 4 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 5 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 6. By comparing these templates (on 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 section 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 7. First, the model with $L = 4$ (default for the method [17]) 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) [18] as argued by the authors.

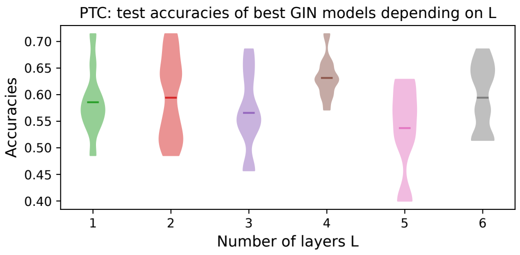

Figure 7: Test accuracies on PTC of GIN for $L \in \{1, ..., 6\}$.

Complements on runtimes. We complete in this paragraph the benchmark on runtimes provided in the section 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

Table 2: 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

taken on CPUs (Intel Core i9-9900K CPU, 3.60 GHz) or a GPU Quatro RTX 4000. As illustrated in Table 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 [5]). 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.

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 [16, 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 3. 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 3: 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)	91.1(2.5)	52.0(4.0)	76.3(3.1)
GAT (L=3)	88.9(1.7)	53.4(3.8)	75.9(2.3)
GAT (L=4)	**91.2(2.8)**	50.9(5.8)	77.6(2.7)
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

Table 4: 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	96.2(3.0)	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)**	82.9(2.7)
		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)	70.9(5.5)	82.1(3.4)

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.

References

[1] 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.

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

[3] 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.

[4] 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.

[5] 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.

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

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

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

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

[10] 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.

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

[12] 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.

[13] 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.

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

[15] 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.

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

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

[18] 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.

[19] 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.