Deep Latent Graph Matching

Tianshu Yu  1  Runzhong Wang  2  Junchi Yan  2  Baoxin Li  1

Abstract

Deep learning for graph matching (GM) has emerged as an important research topic due to its superior performance over traditional methods and insights it provides for solving other combinatorial problems on graph. While recent deep methods for GM extensively investigated effective node/edge feature learning or downstream GM solvers given such learned features, there is little existing work questioning if the fixed connectivity/topology typically constructed using heuristics (e.g., Delaunay or k-nearest) is indeed suitable for GM. From a learning perspective, we argue that the fixed topology may restrict the model capacity and thus potentially hinder the performance. To address this, we propose to learn the (distribution of) latent topology, which can better support the downstream GM task. We devise two latent graph generation procedures, one deterministic and one generative. Particularly, the generative procedure emphasizes the across-graph consistency and thus can be viewed as a matching-guided co-generative model. Our methods deliver superior performance over previous state-of-the-arts on public benchmarks, hence supporting our hypothesis.

1. Introduction

With the strong learning ability of deep networks, recent research on graph matching (GM) has migrated from traditional deterministic optimization (Schellewald & Schnörr, 2005; Cho et al., 2010; Zhou et al., 2015) towards learning-based methods (Zanfir & Sminchisescu, 2018; Wang et al., 2019; Yu et al., 2020). GM is a classic combinatorial and NP-hard problem (Loiola et al., 2007). As the mathematical cornerstone for a series of real-world applications (e.g., image matching (Wang et al., 2018b), social mining (Chiasserini et al., 2018)), GM has received persistent attention from the machine learning and optimization communities for many years. Formally, for two graphs with \( n \) nodes each, graph matching seeks to solve:

\[
\max_{z} z^\top M z \quad \text{s.t.} \quad Z \in \{0, 1\}^{n \times n}, \quad Hz = 1 \tag{1}
\]

where the affinity matrix \( M \in \mathbb{R}^{n \times n^2} \) encodes node (diagonal elements) and edge (off-diagonal elements) affinities/similarities and \( z \) is the column-wise vectorization form of the permutation matrix \( Z \). \( H \) is a selection matrix ensuring each row and column of \( Z \) summing to \( 1 \). \( 1 \) is a column vector filled with \( 1 \). Eq. (1) is the so-called quadratic assignment problem (QAP) (Cho et al., 2010). Maximizing Eq. (1) amounts to maximizing the sum of the similarity induced by the matching vector \( Z \).

Recently, deep learning based GM solvers (Zanfir & Sminchisescu, 2018; Wang et al., 2019; Yu et al., 2020; Fey et al., 2020; Rolínek et al., 2020) have enabled end-to-end training of GM on high-quality human labelled datasets (e.g., Pascal VOC (Everingham et al., 2010; Bourdev & Malik, 2009) and SPair-71k (Min et al., 2019)), which greatly improved the model capacity. Any of the aforementioned deep GM algorithms behaves as an integral framework, of which the main parts cover topology construction\(^1\), feature extraction and differentiable GM solver. In this line of works, affinity \( M \) (see Eq. (1)) is not obtained beforehand, but calculated using node/edge features from some feature backbones given heuristically constructed connectivity, then fed to subsequent GM solvers. Therefore, recent investigation on deep GM frameworks typically focuses on two essential parts: 1) node/edge feature backbone (e.g., graph convolutional networks (Wang et al., 2019), channel-independent embedding (Yu et al., 2020) and SplineCNN (Fey et al., 2018)); 2) GM solvers (e.g., spectral (Zanfir & Sminchisescu, 2018), linear (Wang et al., 2019) and black-box (Pogancic et al., 2020)). In particular, since the feature backbones are variants of

---

\(^1\)Without loss of generality, we discuss graph matching under the setting of equal number of nodes without outliers. The unequal case can be readily handled by introducing extra constraints or dummy nodes. Bipartite matching and graph isomorphism are subsets of this quadratic formulation (Loiola et al., 2007).

\(^2\)Topology in some GM problems is pre-defined and needs to be fixed, such as graph isomorphism. In this paper, we consider a more generic case where topology construction is necessary.
Deep Latent Graph Matching

Graph Neural Networks, they require initial heuristically-constructed connectivity/topology (e.g., Delaunay (Wang et al., 2019) or k-nearest (Zhang & Lee, 2019)), and the topology remains fixed throughout the training procedure in almost all the existing deep GM methods. In this sense, the construction of graph topology is only a pre-processing step, independent of the GM task. This fixed mechanism was adopted in many GM applications ranging from computer vision (Wang et al., 2019; Yu et al., 2020; Fey et al., 2020; Rolínek et al., 2020) to social networks (Zhang & Tong, 2016; Heimann et al., 2018; Xiong & Yan, 2020).

From a learning perspective, we argue that freezing the graph topology for matching can hinder the capacity of deep GM frameworks. For a pre-defined graph topology, the linked nodes sometimes result in less meaningful or even misleading interaction. See schematic demonstrations in Fig. 1 and Fig. 5. Though some earlier attempts (Cho & Lee, 2012; Cho et al., 2013) sought to adjust the graph topology under traditional learning settings, such procedures cannot be readily integrated into end-to-end deep frameworks due to the undifferentiable nature. Our method is built upon the following hypothesis:

- There exists some latent (distribution of) discrete topology better than what is heuristically created for GM.

Based on this, in this paper, we set out to learn the topology (or its distribution) that is more suitable for GM. We propose an end-to-end framework, termed as deep latent graph matching (DLGM), to jointly learn the latent graph topology and perform GM. We leverage the power of graph generative model to automatically produce graph topology from given features and their geometric relations, under two specific prior: locality and consistency. Different from generative learning on singleton graphs (Kipf & Welling, 2016; Bojchevski et al., 2018), our graph generative learning is performed in a pairwise fashion, leading to a novel matching-guided generative paradigm.

The paper makes the following contributions:

- We explore a new direction for more flexible GM by actively learning latent topology, in contrast to previous works using fixed topology as input;
- Under this setting, we propose a deterministic optimization approach to learn graph topology for matching;
- We further present a generative way to produce latent topology, which can be adapted to other problems where graph topology is the latent structure to infer;
- With minimal modification to state-of-the-art GM pipelines, our method achieves superior performance on public benchmarks.

2. Related Works

In this section, we first discuss existing works for graph topology and matching updating, whose motivation is somewhat similar to ours while the technique is largely different. Then we discuss relevant works in learning graph matching and generative graph models from the technical perspective.

Topology updating and matching. There are a few works for joint graph topology updating and matching, in the context of network alignment. Specifically, given two initial networks for matching, Du et al. (2019) showed how to alternatively perform link prediction within each network and node matching across networks based on the observation that these two tasks can benefit each other. In their extension (Du et al., 2020), a skip-gram embedding framework was further established under the same setting. These works involve random-walk-based node embedding updating and classification-based link prediction, and the whole algorithm runs in a one-shot optimization fashion. There is neither explicit training dataset nor trained matching model (except for the link classifier), which bears less flavor of machine learning. In contrast, our method involves training an explicit model for topology recovery and matching solving. Specifically, our deterministic technique (see Sec. 3.4.1) solves graph topology and matching in one-shot, while the proposed generative method alternatively estimates the topology and matching (see Sec. 3.4.2). Our approach allows the algorithm to fully leverage multiple training samples to boost the performance on the test set.
Moreover, the combinatorial nature of the matching problem is not addressed in (Du et al., 2019; 2020), where a greedy selection strategy was adopted instead. In contrast, we develop a principled combinatorial learning approach to this challenge. Also their methods rely on a considerable amount of seed matchings, yet this paper directly learns the latent topology from scratch which is more challenging and practical but yet seldom studied.

Learning of graph matching. Early shallow models sought to learn effective metric (e.g. weighted Euclidian distance) for node and edge features or affinity kernel (e.g. Gaussian kernel) in a parametric fashion (Caetano et al., 2009; Cho et al., 2013). Recent deep graph matching methods have shown how to extract more dedicated features. The work (Zanfir & Sminchisescu, 2018) adopts VGG16 (Simonyan & Zisserman, 2014) as the backbone for feature extraction on images. Other efforts have been devoted to developing more advanced pipelines, where graph embedding (Wang et al., 2019; Yu et al., 2020; Fey et al., 2020) and geometric learning (Zhang & Lee, 2019; Fey et al., 2020) are involved. Rolínek et al. (2020) studied the way of incorporating traditional non-differentiable combinatorial solvers by introducing a differentiable blackbox GM solver (Pogancic et al., 2020). Recent works in tackling combinatorial problem with deep learning (Huang et al., 2019; Kool & Welling, 2018) also inspired development of combinatorial deep solvers for GM problems formulated by both Koopmans-Beckmann’s QAP (Nowak et al., 2018; Wang et al., 2019) and Lawler’s QAP (Wang et al., 2021). Specifically, Wang et al. (2019) devised a permutation loss for supervised learning, with an improvement in (Yu et al., 2020) via Hungarian attention. Wang et al. (2021) solved the most general Lawler’s QAP with a graph embedding technique.

Generative graph model. Early generative models for graph can be dated back to (Erdos & Renyi, 1959), in which edges are generated with fixed probability. Recently, Kipf & Welling (2016) presented a graph generative model by re-parameterizing the edge probability from Gaussian noise. Johnson (2017) proposed to generate graph in an incremental fashion, and in each iteration a portion of the graph is produced. Gómez-Bombarelli et al. (2018) utilized recurrent neural network to generate graph from a sequence of molecule representation. Adversarial graph generation was considered in (Pan et al., 2018; Wang et al., 2018a; Bojchevski et al., 2018). Specifically, Wang et al. (2018a); Bojchevski et al. (2018) sought to unify graph generative model and generative adversarial networks. In parallel, reinforcement learning has been adopted to generate discrete graphs (De Cao & Kipf, 2018).

3. Learning Latent Topology for GM

In this section, we describe details of the proposed framework with two specific algorithms derived from deterministic and generative perspectives, respectively. Both algorithms are motivated by the hypothesis that there exists some latent topology more suitable for matching rather than a fixed one. Note that the proposed deterministic algorithm performs a standard forward-backward pass to jointly learn the topology and matching, while our generative algorithm consists of an alternative optimization procedure between estimating latent topology and learning matching under an Expectation-Maximization (EM) interpretation. In general, the generative algorithm assumes that a latent topology is sampled from a latent distribution, where the expected matching accuracy under this distribution is maximized. Therefore, we expect to learn a topology generator under such distribution. We reformulate GM in a Bayesian fashion for consistent discussion in Sec. 3.1, detail deterministic/generative latent module in Sec. 3.2 and discuss the loss functions from a probabilistic perspective in Sec. 3.3. We finally elaborate on the holistic framework and the optimization procedure for both algorithms (deterministic and generative) in Sec. 3.4.

3.1. Problem Definition and Background

Learning-based GM problem can be viewed as an extension to Eq. (1). Let \( G^{(s)} \) represent respectively the source and target graphs for matching. We represent a graph as \( G := \{X, E, A\} \), where \( X \in \mathbb{R}^{n \times d_1} \) is the representation of \( n \) nodes with dimension \( d_1 \), \( E \in \mathbb{R}^{m \times d_2} \) are \( d_2 \)-dimensional features of \( m \) edges and \( A \in \{0, 1\}^{n \times n} \) is initial connectivity (i.e., topology) matrix by heuristics (e.g., Delaunay triangulation). For notational brevity, we assume \( d_1 \) and \( d_2 \) remain intact after updating the features across each convolutional layers of GNN (i.e., feature dimensions of both nodes and edges will not change after each layer’s update). Denote the matching \( Z \in \{0, 1\}^{n \times n} \) between two graphs, where \( Z_{ij} = 1 \) indicates a correspondence exists between node \( i \) in \( G^{(s)} \) and node \( j \) in \( G^{(t)} \), and \( Z_{ij} = 0 \) otherwise. Given training samples \( \{Z_k, G^{(s)}_k, G^{(t)}_k\} \) with \( k = 1, 2, ..., N \), the objective of learning-based GM aims to maximize the likelihood:

\[
\max_{\theta} \prod_k P_{\theta}(Z_k|G^{(s)}_k, G^{(t)}_k) \tag{2}
\]

where \( \theta \) denotes model parameters. \( P_{\theta}(\cdot) \) measures the probability of matching \( Z_k \) given the \( k \)-th pair, and is instantiated via a network parameterized by \( \theta \).

Being a generic module for producing latent topology, our method can be flexibly and easily integrated into existing deep GM frameworks. We build up our method based on state-of-the-art (Rolínek et al., 2020), which uti-
lizes SplineCNN (Fey et al., 2018) for node/edge feature learning and black-box GM solver (Pogancic et al., 2020). SplineCNN is a specific graph neural networks which updates a node representation via a weighted summation of its neighbors. The update rule at node $i$ of a standard SplineCNN reads:

$$(x * g)(i) = \frac{1}{|N(i)|} \sum_{j=1}^{d} \sum_{j \in N(i)} x_i(j) \cdot g_i(e(i, j))$$

where $x_i(j)$ performs the convolution on node $j$ and outputs a $d_i$-dimensional feature. $g_i(\cdot)$ delivers the message weight given the edge feature $e(i, j)$. $N(i)$ refers to $i$’s neighboring nodes. Summation over neighbors follows the topology $A$.

Since our algorithm learns to generate topology, we need to explicitly express Eq. (3) in a differentiable way w.r.t. $A$. To this end, we rewrite Eq. (3) as:

$$(x * g)(A) = (\hat{A} \circ G)\hat{X}$$

where $\hat{A}$ is the normalized connectivity with each row normalized by the degree $|N(i)|$ (see Eq. (3)) of the corresponding node $i$. G and $\hat{X}$ correspond to outputs of $g_i(\cdot)$ and $x_i(\cdot)$ operators, respectively. $(\cdot \circ \cdot)$ is the Hadamard product. With Eq. (4), we thus can perform back-propagation on connectivity/topology $A$. See more details in Appendix A.3.

### 3.2. Latent Topology Learning

Existing learning-based graph matching algorithms consider $A$ to be fixed throughout the computation without questioning if the input topology is optimal or not. This can be problematic since input graph construction is heuristic, and it never takes into account how suitable it is for the subsequent GM task. In our framework, instead of utilizing a fixed pre-defined topology, we consider to produce latent topology under two settings: 1) a deterministic and 2) a generative way. The former is often more efficient while the latter method can be more accurate at the cost of exploring more latent topology. Note that both methods produce discrete topology to verify our hypothesis about the existence of more suitable discrete latent topology for GM. The corresponding two deep structures are described below.

**Deterministic learning**: Given input features $X$ and initial topology $A$, the deterministic way of generating latent topology $\hat{A} \in \{0, 1\}^{n \times n}$ is:

$$\hat{A}_{ij} = \text{Rounding}(\text{sigmoid}(y_i^\top W y_j))$$

with $Y = \text{GCN}(X, A)$

$$\text{(5)}$$

where $\text{GCN}(\cdot)$ is the graph convolutional networks (GCN) (Kipf & Welling, 2017) and $y_i$ corresponds to the feature of node $i$ in feature map $Y$. $W$ is the learnable parameter matrix. Note that function $\text{Rounding}(\cdot)$ is undifferentiable, and will be discussed in Sec. 3.4.1.

**Generative learning**: We reparameterize the representation:

$$P(y_i | X, A) = \mathcal{N}(y_i | \mu_i, \text{diag}({\sigma^2}))$$

with $\mu = \text{GCN}_\mu(X, A)$ and $\sigma = \text{GCN}_\sigma(X, A)$ are two GCNs producing mean and covariance. It is equivalent to sampling a random vector from i.i.d. uniform distribution $s \sim \mathcal{U}(0, 1)$, then applying $y = \mu + s \cdot \sigma$, where $(\cdot)$ is element-wise product.

Similar to Eq. (5), by introducing learnable parameter $W$, the generative latent topology is sampled following i.i.d. distribution over each edge $(i, j)$:

$$P(A | Y) = \prod_i \prod_j P(A_{ij} | y_i, y_j)$$

$$\text{(7)}$$

with $P(A_{ij} = 1 | y_i, y_j) = \text{sigmoid}(y_i^\top W y_j)$

Since sigmoid$(\cdot)$ maps any input into $(0, 1)$, Eq. (7) can be interpreted as the probability of sampling edge $(i, j)$. As the sampling procedure is undifferentiable, we apply Gumbel-softmax trick (Jang et al., 2017) as another reparameterization procedure. As such, a latent graph topology $A$ can be sampled fully from distribution $P(A)$ and the procedure becomes differentiable.

### 3.3. Loss Functions

In this section, we explain three loss functions and the underlying motivation: **matching loss**, **locality loss** and **consistency loss**. The corresponding probabilistic interpretation of each loss function can be found in Sec. 3.4.2. These functions are selectively activated in DLGM-D and DLGM-G (see Sec. 3.4). In DLGM-G, different loss functions are activated in inference and learning steps.

**i) Matching loss.** This common term measures how the predicted matching $\hat{Z}$ diverges from ground-truth $Z$. Following Rolínek et al. (2020), we adopt Hamming distance on node-wise matching:

$$\mathcal{L}_M = \text{Hamming}(\hat{Z}, Z)$$

**ii) Locality loss.** This loss is devised to account for the general prior that the produced/learnt graph topology should advocate local connections rather than distant ones, since two nodes may have less meaningful interaction once they are too distant from each other. In this sense, locality loss serves as a prior or regularizer in GM. As shown in multiple GM methods (Yu et al., 2018; Wang et al., 2019; Fey et al., 2020), Delaunay triangulation is an effective way to deliver good locality. Therefore in our method, the locality loss is the Hamming distance between the initial topology $A$.
We emphasize that the locality loss serves as a prior for latent graph. It focuses on advocating locality, but not re-constructing the initial Delaunay triangulation (as in Graph VAE (Kipf & Welling, 2016)).

**iii) Consistency loss.** One can imagine that a GM solver is likely to deliver better performance if two graphs in a training pair are similar. In particular, we anticipate the latent topology $\mathbf{A}^{(s)}$ and $\mathbf{A}^{(t)}$ to be isomorphic under a specific matching, since isomorphic topological structures tend to be easier to match. Driven by this consideration, we devise the consistency loss which measures the level of isomorphism between latent topology $\mathbf{A}^{(s)}$ and $\mathbf{A}^{(t)}$:

$$L_C(\mathbf{Z}) = |\mathbf{Z}^T \mathbf{A}^{(s)} \mathbf{Z} - \mathbf{A}^{(t)}| + |\mathbf{Z} \mathbf{A}^{(t)} \mathbf{Z}^T - \mathbf{A}^{(s)}| \quad (10)$$

Note that $\mathbf{Z}$ does not necessarily refer to the ground-truth, but can be any predicted matching. In this sense, latent topology $\mathbf{A}^{(s)}$ and $\mathbf{A}^{(t)}$ can be generated jointly given the matching $\mathbf{Z}$ as guidance. This term can also serve as a consistency prior or regularization. We give a schematic example showing the merit of introducing the consistency loss in Fig. 2(b).

### 3.4. Framework

A schematic diagram of our framework is given in Fig. 2(a) which consists of a singleton pipeline for processing a single image. It consists of three essential modules: a feature backbone ($N_B$), a latent topology module ($N_G$) and a feature refinement module ($N_R$). Specifically, module $N_G$ corresponds to Sec. 3.2 with deterministic or generative implementations. Note that the geometric relations of keypoints provide some prior for generating topology $A$. We employ VGG16 (Simonyan & Zisserman, 2014) as $N_B$ and feed the produced node feature $X$ and edge feature $E$ to $N_G$. $N_B$ also produces a global feature for each image. After generating the latent topology $A$, we pass over $X$ and $E$ together with $A$ to $N_R$ (SplineCNN (Fey et al., 2018)). The holistic pipeline handling pairwise graph inputs can be found in Fig. 5 in Appendix A.2, which consists of two copies of singleton pipeline processing source and target data (in a Siamese fashion), respectively. Then the outputs of two singleton pipelines are formulated into affinity matrix, followed by a differentiable Blackbox GM solver (Pogancic et al., 2020) with message-passing mechanism (Swoboda et al., 2017). Note that, if $N_G$ is removed, the holistic pipeline with only $N_B + N_R$ is identical to the method in (Rolínek et al., 2020). Readers are referred to this strong baseline (Rolínek et al., 2020) for more algorithmic details.

### 3.4.1. Optimization with Deterministic Latent Graph

We now show how to optimize with the deterministic latent graph module, where the topology $\mathbf{A}$ is produced by Eq. (5).

The objective of matching conditioned on the produced latent topology $\mathbf{A}$ becomes:

$$\max \prod_k P(\mathbf{Z}|\mathbf{A}^{(s)}_k, \mathbf{A}^{(t)}_k, \mathbf{g}^{(s)}_k, \mathbf{g}^{(t)}_k) \quad (11)$$

Eq. (11) can be optimized with standard back-propagation with three loss terms activated, except for the Rounding function (see Eq. (5)), which makes the procedure undifferentiable. To address this, we use straight-through operator (Bengio et al., 2013) which performs a standard rounding during the forward pass but approximates it with the gradient of identity during the backward pass on $[0, 1]$:

$$\partial \text{Rounding}(x)/\partial x = 1 \quad (12)$$

Though there exist some unbiased gradient estimators (e.g., REINFORCE (Williams, 1992)), the biased straight-through estimator proved to be more efficient and has been successfully applied in several applications (Chung et al., 2017; Campos et al., 2018). All the network modules ($N_G + N_B + N_R$) are simultaneously learned during the training. All three losses are activated in the learning procedure (see Sec. 3.3), which are applied on the predicted matching $\mathbf{Z}$, the latent topology $\mathbf{A}^{(s)}$ and $\mathbf{A}^{(t)}$. We term the algorithm under this setting DLGM-D.

### 3.4.2. Optimization with Generative Latent Graph

See more details in Appendix A.4. In this setting, the source and target latent topology $\mathbf{A}^{(s)}$ and $\mathbf{A}^{(t)}$ are sampled according to Eq. (6) and (7). The objective becomes:

$$\max \prod_k \int_{\mathbf{A}^{(s)}_k, \mathbf{A}^{(t)}_k} P_\theta(\mathbf{Z}|\mathbf{A}^{(s)}_k, \mathbf{A}^{(t)}_k, \mathbf{g}^{(s)}_k, \mathbf{g}^{(t)}_k) \quad (13)$$

Unfortunately, directly optimizing Eq. (13) is difficult due to the integration over $\mathbf{A}$, which is intractable. Instead, we maximize the evidence lower bound (ELBO) (Bishop, 2006) as follows:

$$\log P_\theta(\mathbf{Z}|\mathbf{g}^{(s)}, \mathbf{g}^{(t)}) \geq \mathbb{E}_{Q_\phi(\mathbf{A}^{(s)}_k, \mathbf{A}^{(t)}_k|\mathbf{g}^{(s)}_k, \mathbf{g}^{(t)}_k)} \left[ \log P_\theta(\mathbf{Z}, \mathbf{A}^{(s)}_k, \mathbf{A}^{(t)}_k|\mathbf{g}^{(s)}_k, \mathbf{g}^{(t)}_k) \right. \\
- \left. \log Q_\phi(\mathbf{A}^{(s)}_k, \mathbf{A}^{(t)}_k|\mathbf{g}^{(s)}_k, \mathbf{g}^{(t)}_k) \right]$$

(14)

where $Q_\phi(\mathbf{A}^{(s)}_k, \mathbf{A}^{(t)}_k|\mathbf{g}^{(s)}_k, \mathbf{g}^{(t)}_k)$ can be any joint distribution of $\mathbf{A}^{(s)}_k$ and $\mathbf{A}^{(t)}_k$ given the input graphs.
\( G^{(s)} \) and \( G^{(t)} \). Equality of Eq. (14) holds when 
\[ Q_\phi(\mathbf{A}^{(s)}, \mathbf{A}^{(t)} | g^{(s)}, g^{(t)}) = P_\theta(\mathbf{A}^{(s)}, \mathbf{A}^{(t)} | Z, g^{(s)}, g^{(t)}) \]
for tractability, we introduce the independence by assuming that we can use an identical latent topology module \( Q_\phi \) (corresponding to \( N_G \) in Fig. 2(a)) to separately handle each input graph:
\[ Q_\phi(\mathbf{A}^{(s)}, \mathbf{A}^{(t)} | g^{(s)}, g^{(t)}) = Q_\phi(\mathbf{A}^{(s)} | g^{(s)})Q_\phi(\mathbf{A}^{(t)} | g^{(t)}) \]  
(15)
which can greatly simplify the model complexity. Then we can utilize a neural network to model \( Q_\phi \) (similar to modeling \( P_\theta \)). The optimization of Eq. (14) is studied in (Neal & Hinton, 1998), known as the Expectation-Maximization (EM) algorithm. Optimization of Eq. (14) alternates between E-step and M-step. During E-step (inference), \( P_\theta \) is fixed and the algorithm seeks to find an optimal \( Q_\phi \) to approximate the true posterior distribution (see Appendix A.4 for explanation):
\[ P_\theta(\mathbf{A}^{(s)}, \mathbf{A}^{(t)} | Z, g^{(s)}, g^{(t)}) \]  
(16)
During M-step (learning), \( Q_\phi \) is instead fixed and algorithm alters to maximize the likelihood:
\[ \mathbb{E}_{Q_\phi(\mathbf{A}^{(s)} | g^{(s)}), Q_\phi(\mathbf{A}^{(t)} | g^{(t)})} \left[ \log P_\theta(\mathbf{Z}, \mathbf{A}^{(s)}, \mathbf{A}^{(t)} | g^{(s)}, g^{(t)}) \right] = - \mathcal{L}_M \]  
(17)
We detail on the inference and learning steps as follows.

**Inference.** This step focuses on deriving posterior distribution \( P_\theta(\mathbf{A}^{(s)}, \mathbf{A}^{(t)} | Z, g^{(s)}, g^{(t)}) \) using its approximation \( Q_\phi \). To this end, we fix the parameters \( \theta \) in modules \( N_B \) and \( N_T \), and only update the parameters \( \phi \) in module \( N_G \) corresponding to \( Q_\phi \). As stated in Sec. 3.2, we employ the Gumbel-softmax trick for sampling discrete \( \mathbf{A} \) (Jang et al., 2017). To this end, we can formulate a 2D vector \( \mathbf{a}_{ij} = [P(\mathbf{A}_{ij} = 1), 1 - P(\mathbf{A}_{ij} = 1)]^T \). Then the sampling becomes:
\[ \text{softmax}(\log(\mathbf{a}_{ij}) + \mathbf{h}_{ij}; \tau) \]  
(18)
where \( \mathbf{h}_{ij} \) is a random 2D vector from Gumbel distribution, and \( \tau \) is a small temperature parameter. We further impose prior on latent topology \( \mathbf{A} \) given \( \mathbf{A} \) through *locality loss*:
\[ \log \prod_{i,j} P(\mathbf{A}_{ij} | \mathbf{A}_{ij}) \propto -\mathcal{L}_L(\mathbf{A}, \mathbf{A}) \]  
(19)
which is to preserve the locality in initial topology \( \mathbf{A} \). It should also be noted that \( \mathbf{Z} \) is the *predicted* matching from current \( P_\theta \), as \( Q_\phi \) is an approximation. Besides, we also anticipate two generated topology \( \mathbf{A}^{(s)} \) and \( \mathbf{A}^{(t)} \) from a graph pair should be similar (isomorphic) given \( \mathbf{Z} \):
\[ \log P \left( \mathbf{A}^{(s)}, \mathbf{A}^{(t)} | \mathbf{Z} \right) \propto -\mathcal{L}_C \left( \mathbf{A}^{(s)}, \mathbf{A}^{(t)} | \mathbf{Z} \right) \]  
(20)
In summary, we activate *locality loss* and *consistency loss* as \( \alpha \mathcal{L}_L + \beta \mathcal{L}_C \) during the inference step, where the latter loss is conditioned with the predicted matching rather than the ground-truth. Note that the inference step involves twice re-parameterization tricks corresponding to Eq. (6) and (18), respectively. While the first generates the continuous topology distribution under edge independence assumption, the second performs discrete sampling according to the generated topology distribution.

**Learning.** This step optimizes \( P_\theta \) by fixing \( Q_\phi \). We sample discrete graph topologies \( \mathbf{A}^{(s)} \) completely from the probability of edge \( P(\mathbf{A}_{ij} = 1) \). Once latent topology \( \mathbf{A}^{(s)} \) are sampled, we feed them to module \( N_R \) together with the
Deep Latent Graph Matching

Algorithm 1 DLGM-D

1: Input: $G^s$, $G^t$ and ground-truth $Z$;
2: Output: matching $Z$;
3: Pretrain $P_0$ using Eq. (11), given Delaunay as input topology;
4: repeat
5:  # Inference (E-step):
6:  Obtain predicted matching $\hat{Z}$ using fixed $P_0$;
7:  Update $Q_0$ (i.e. $N_C$) with loss $L_L + L_C(\cdot |\hat{Z})$ according to Eq. (16);
8:  # Learning (M-step):
9:  Obtain predicted graph topology $A^{(s)}$ and $A^{(t)}$ using $Q_0$;
10: Update $P_0$ (i.e. $N_B$ and $N_R$) with loss $L_M$ given $A^{(s)}$ and $A^{(t)}$ according to Eq. (17);
11: until converge
12: Predict topology and the matching $\hat{Z}$ with whole network activated (i.e. $N_C + N_B + N_R$);

node-level features from $N_R$. Only $N_B$ and $N_R$ are updated in this step, and only matching loss $L_M$ is activated.

Remark. Note for each pair of graphs in training, we use an identical random vector $s$ for generating both graphs’ topology (see Eq. (6)). We pretrain the network $P_0$ before alternatively training $P_0$ and $Q_0$. During pretraining, we activate $N_B + N_R$ modules and $L_M$ loss during pretraining, and feed the network the topology obtained from Delaunay as the latent topology. After pretraining, the optimization will switch between inference and learning steps until convergence. We term the setting of generative latent graph matching as DLGM-G and summarize it in Alg. 1.

4. Experiment

We conduct experiments on datasets including Pascal VOC with Berkeley annotation (Everingham et al., 2010; Bourdev & Malik, 2009), Willow ObjectClass (Cho et al., 2013) and SPair-71K (Min et al., 2019). We report the per-category and average performance. The objective of all experiments is to maximize the average matching accuracy. Both our DLGM-D and DLGM-G are tested. Except for the ablation study, we consistently conduct experiments under $\alpha = 5.0$ and $\beta = 0.3$. We will test different combinations of $\alpha$s and $\beta$s in the ablation study (Sec. 4.4).

Peer methods. We conduct comparison experiments against the following algorithms: 1) GMN (Zanfir & Sminchisescu, 2018), which is a seminal work incorporating graph matching into deep learning framework equipped with a spectral solver (Egozi et al., 2012); 2) PCA (Wang et al., 2019). This method treats graph matching as feature matching problem and employs GCN (Kipf & Welling, 2017) to learn better features; 3) CIE4/GAT-H (Yu et al., 2020).

This paper develops an embedding and attention mechanism, where GAT-H is the version by replacing the basic embedding block with Graph Attention Networks (Velicković et al., 2018); 4) DGMC (Fei et al., 2020). This method devises a post-processing step by emphasizing the neighborhood similarity; 5) BBGM (Rolínek et al., 2020). It integrates a differentiable linear combinatorial solver (Pogančić et al., 2018) into a deep learning framework and achieves state-of-the-art performance.

4.1. Results on Pascal VOC.

This dataset (Everingham et al., 2010; Bourdev & Malik, 2009) consists of 7,020 training images and 1,682 testing images with 20 classes in total, together with the object bounding box for each. Following the data preparation in (Wang et al., 2019), each object within the bounding box is cropped and resized to $256 \times 256$. The number of nodes per graph ranges from 6 to 23. We further follow (Rolínek et al., 2020) under two evaluating metrics: 1) Accuracy: this is the standard metric evaluated on the keypoints by filtering out the outliers; 2) F1-score: this metric is evaluated without keypoint filtering, being the harmonic mean of precision and recall. Therefore, task 2) can be viewed as common sub-graph matching with outliers. Experimental results on the two setting are shown in Tab. 1 and Tab. 2. The proposed method under either settings of DLGM-D and DLGM-G outperforms counterparts by accuracy and f1-score. DLGM-G generally outperforms DLGM-D. Discussion can be found in Appendix A.5.

Quality of generated topology. We further show the consistency/locality curve vs epoch in Fig. 3, since both consistency and locality losses can somewhat reflect the quality of topology generation. It shows that both locality and consistency losses descend during the training. Note that the consistency loss with Delaunay triangulation (green dashed line) is far more larger than our generated ones (blue/red dashed line). This clearly supports the claim that our method generates similar (more isomorphic) typologies, as well as preserving locality.

4.2. Results on Willow Object.

The benchmark (Cho et al., 2013) consists of 256 images in 5 categories, where two categories (car and motorbike) are subsets from Pascal VOC. Following the protocol in Wang et al. (2019), we crop the image within the object bounding box and resize it to $256 \times 256$. Since the dataset is relatively small, we conduct the experiment to verify the transfer ability of different methods under two settings: 1) trained on Pascal VOC and directly applied to Willow (P1); 2) trained on Pascal VOC then finetuned on Willow (W1). Results under the two settings are shown in Tab. 4. Since this dataset is relatively small, further improvement
is difficult. It is shown both DLGM-D and DLGM-G have good transfer ability.

### 4.3. Results on SPair-71K

This dataset (Min et al., 2019) is much larger than Pascal VOC and WillowObject. It consists of 70,958 image pairs collected from Pascal VOC 2012 and Pascal 3D+ (53,340 for training, 5,384 for validation and 12,234 for testing). It improves Pascal VOC by removing ambiguous categories sofa and dining table. This dataset is considered to contain more difficult matching instances and higher annotation quality. Results are summarized in Tab. 3. Our method consistently improves the matching performance, agreeing with those in Pascal VOC and Willow.

Table 1. F1-score (%) on Pascal VOC. Experiment are performed on a pair of images where both inlier and outlier keypoints are considered. BBGM-max is a setting in Rolínek et al. (2020).

<table>
<thead>
<tr>
<th>METHOD</th>
<th>AERO BIKE BIRD BOAT BOTTLE BUS CAR CAT CHAIR COW TABLE DOG HORSE MBIKE PERSON PLANT SHEEP SOFA TRAIN TV</th>
<th>AVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>GMN</td>
<td>31.1 46.2 58.2 45.9 70.6 76.4 61.2 61.7 35.5 53.7 58.9 57.5 56.9 49.3 34.1 77.5 57.1 53.6 83.2 88.6 57.9</td>
<td></td>
</tr>
<tr>
<td>GAT-H</td>
<td>47.2 61.6 63.2 53.3 79.7 70.1 65.3 70.5 38.4 64.7 62.9 65.1 66.2 62.5 41.1 78.8 67.1 61.6 81.4 91.0 64.6</td>
<td></td>
</tr>
<tr>
<td>PCA</td>
<td>40.9 55.0 65.8 47.9 76.9 77.9 63.5 67.4 33.7 65.5 63.6 61.3 68.9 62.8 44.9 77.5 67.4 57.5 86.7 90.9 63.8</td>
<td></td>
</tr>
<tr>
<td>CIE-H</td>
<td>51.2 69.2 70.1 55.9 82.8 72.8 69.0 74.2 39.6 68.8 71.8 70.0 71.8 66.8 44.8 85.2 69.9 65.4 85.2 92.4 68.9</td>
<td></td>
</tr>
<tr>
<td>DGM</td>
<td>50.4 67.6 70.7 70.5 87.2 85.2 82.5 74.3 46.2 69.4 69.9 73.9 73.8 65.4 51.6 98.0 73.2 69.6 94.3 89.6 73.2</td>
<td></td>
</tr>
<tr>
<td>BBGM</td>
<td>61.5 75.0 78.1 80.0 87.4 93.0 89.1 80.2 58.1 77.6 76.5 79.3 78.6 78.8 66.7 97.4 76.4 77.5 97.7 94.4 80.1</td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Accuracy (%) on SPair-71K compared with state-of-the-art methods (best in bold).

<table>
<thead>
<tr>
<th>METHOD</th>
<th>AERO BIKE BIRD BOAT BOTTLE BUS CAR CAT CHAIR COW TABLE DOG HORSE MBIKE PERSON PLANT SHEEP SOFA TRAIN TV</th>
<th>AVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBGM-MAX</td>
<td>35.5 68.6 46.7 36.1 85.4 58.1 25.6 51.7 27.3 51.0 46.0 46.7 48.9 58.9 29.6 93.6 42.6 35.3 70.7 79.5 51.9</td>
<td></td>
</tr>
<tr>
<td>BBGM</td>
<td>42.7 70.9 57.5 46.6 85.8 64.1 51.0 63.8 42.4 63.7 47.9 61.5 63.4 69.0 46.1 94.2 57.4 39.0 78.0 82.7 61.4</td>
<td></td>
</tr>
</tbody>
</table>

Table 3. Accuracy (%) on Willow Object.

<table>
<thead>
<tr>
<th>METHOD</th>
<th>AERO BIKE BIRD BOAT BOTTLE BUS CAR CAT CHAIR COW TABLE DOG HORSE MBIKE PERSON PLANT SHEEP SOFA TRAIN TV</th>
<th>AVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DGM</td>
<td>54.8 44.8 80.3 70.9 65.5 90.1 78.5 66.7 66.4 73.2 66.2 66.5 65.7 59.1 98.7 68.5 84.9 98.0 72.2</td>
<td></td>
</tr>
<tr>
<td>BBGM</td>
<td>66.9 57.7 85.8 78.5 66.9 95.4 86.1 74.6 68.3 78.9 73.0 67.5 79.3 73.0 99.1 74.8 95.0 98.6 78.9</td>
<td></td>
</tr>
</tbody>
</table>

Table 4. Accuracy (%) on Willow Object.

<table>
<thead>
<tr>
<th>METHOD</th>
<th>SETTING</th>
<th>FACE</th>
<th>MBIKE</th>
<th>CAR</th>
<th>DUCK</th>
<th>WBOTTLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>GMN</td>
<td>Pt</td>
<td>98.1</td>
<td>65.0</td>
<td>72.9</td>
<td>74.3</td>
<td>70.5</td>
</tr>
<tr>
<td>PCA</td>
<td>Wt</td>
<td>99.3</td>
<td>71.4</td>
<td>74.3</td>
<td>82.8</td>
<td>76.7</td>
</tr>
<tr>
<td>CIE-H</td>
<td>Pt</td>
<td>99.9</td>
<td>71.5</td>
<td>75.4</td>
<td>73.2</td>
<td>97.6</td>
</tr>
<tr>
<td>DGM</td>
<td>Wt</td>
<td>100.0</td>
<td>76.7</td>
<td>84.0</td>
<td>93.5</td>
<td>96.9</td>
</tr>
<tr>
<td>BBGM</td>
<td>Pt</td>
<td>98.6</td>
<td>69.8</td>
<td>84.6</td>
<td>76.8</td>
<td>90.7</td>
</tr>
<tr>
<td>Wt</td>
<td>100.0</td>
<td>98.8</td>
<td>96.5</td>
<td>93.2</td>
<td>99.9</td>
<td>97.6</td>
</tr>
<tr>
<td>DLGM-D (OURS)</td>
<td>Pt</td>
<td>100.0</td>
<td>95.5</td>
<td>91.3</td>
<td>91.4</td>
<td>97.9</td>
</tr>
<tr>
<td>Wt</td>
<td>100.0</td>
<td>99.4</td>
<td>95.9</td>
<td>92.8</td>
<td>99.3</td>
<td>97.9</td>
</tr>
<tr>
<td>DLGM-G (OURS)</td>
<td>Pt</td>
<td>99.9</td>
<td>96.4</td>
<td>92.0</td>
<td>91.8</td>
<td>98.0</td>
</tr>
<tr>
<td>Wt</td>
<td>100.0</td>
<td>99.3</td>
<td>96.5</td>
<td>93.7</td>
<td>99.3</td>
<td>98.0</td>
</tr>
</tbody>
</table>

Figure 3. Consistency loss (Eq. (9) and locality loss (Eq. 10)) keep decrease over training which suggests the effectiveness for adaptive topology learning for matching.
Table 5. Selectively deactivating loss functions on Pascal VOC. $\mathcal{L}_M$, $\mathcal{L}_C$ and $\mathcal{L}_L$ are selectively activated in DLGM-D and DLGM-G. “full” indicates all loss functions are activated. Average accuracy (%) is reported.

<table>
<thead>
<tr>
<th>METHOD</th>
<th>AVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLGM-D ($\mathcal{L}_M + \mathcal{L}_C$)</td>
<td>79.8</td>
</tr>
<tr>
<td>DLGM-D ($\mathcal{L}_M + \mathcal{L}_L$)</td>
<td>79.5</td>
</tr>
<tr>
<td>DLGM-G ($\mathcal{L}_M + \mathcal{L}_C$)</td>
<td>80.9</td>
</tr>
<tr>
<td>DLGM-G ($\mathcal{L}_M + \mathcal{L}_L$)</td>
<td>80.4</td>
</tr>
<tr>
<td>DLGM-D (FULL)</td>
<td>82.9</td>
</tr>
<tr>
<td>DLGM-G (FULL)</td>
<td>83.8</td>
</tr>
</tbody>
</table>

4.4. Ablation study

We conduct ablation to show the effectiveness of some factors involved in our framework (e.g., sampling size of the generator and varying loss strength $\alpha$ and $\beta$).

In the first part, we evaluate the performance of DLGM-D and DLGM-G by selectively deactivating different loss functions $\mathcal{L}_M$, $\mathcal{L}_C$ and $\mathcal{L}_L$. Since our method involves a sampling procedure, we also conduct the test on DLGM-G using different sample size of the generator. This ablation test is conducted on Pascal VOC dataset and average accuracy is reported in Tab. 5 and 6.

We first test the performance of both settings of DLGM by selectively activate the designated loss functions. Experimental results are summarized in Tab. 5. As matching loss $\mathcal{L}_M$ is essential for GM task, we constantly activate this loss for all settings. Note once $\mathcal{L}_C$ and $\mathcal{L}_L$ are both deactivated, our method will degenerate into BBGM (Rolínek et al., 2020). In this case, there will be no need to train the generator $Q_\phi$. We see that the proposed novel losses $\mathcal{L}_C$ and $\mathcal{L}_L$ can consistently enhance the matching performance. Besides, DLGM-G indeed delivers better performance than DLGM-D under fair comparison.

We then test the impact of sample size from the generator $Q_\phi$ under DLGM-G. Experimental results are summarized in Tab. 6. We see that along with the increasing sample size, the average accuracy ascends. The performance becomes stable when the sample size reaches over 16.

Remark. In terms of the time efficiency, if we consider the training time of the baseline (Rolínek et al., 2020) to be 1x, the training time of our method under discriminative setting is around 1.2x-1.3x. The time cost of our method under generative setting is around 8x-9x with sample size 16. We didn’t observe any obvious efficiency gap for the testing stage.

In the second part, we present more detailed results by varying the loss strength $\alpha$s and $\beta$s for DLGM-G. Letting the loss at inference step be $\alpha\mathcal{L}_L + \beta\mathcal{L}_C$, Tab. 7 shows the performance of DLGM-G with varying $\alpha$ and $\beta$ on Pascal VOC with only inliers (Note we reported $\alpha = 5.0$ and $\beta = 0.3$ in all the previous experiments on each dataset):

Table 6. Average matching accuracy under different sampling sizes from the generator $Q_\phi$ with “full” DLGM-G setting.

<table>
<thead>
<tr>
<th>#SAMPLE</th>
<th>AVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>82.5</td>
</tr>
<tr>
<td>2</td>
<td>83.2</td>
</tr>
<tr>
<td>4</td>
<td>83.2</td>
</tr>
<tr>
<td>8</td>
<td>83.5</td>
</tr>
<tr>
<td>16</td>
<td>83.8</td>
</tr>
<tr>
<td>32</td>
<td>83.7</td>
</tr>
</tbody>
</table>

Table 7. Ablation study of DLGM-G on Pascal VOC dataset. $\alpha$ and $\beta$ correspond to the strength of locality loss $\mathcal{L}_L$ and consistency loss $\mathcal{L}_C$, respectively. Average accuracy (%) is reported.

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>$\beta$</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.0</td>
<td></td>
<td>82.0</td>
<td>81.8</td>
<td>82.4</td>
<td>82.1</td>
<td>81.9</td>
</tr>
<tr>
<td>4.5</td>
<td></td>
<td>82.2</td>
<td>82.6</td>
<td>82.9</td>
<td>82.5</td>
<td>82.5</td>
</tr>
<tr>
<td>5.0</td>
<td></td>
<td>82.3</td>
<td>83.3</td>
<td>83.8</td>
<td>83.1</td>
<td>82.5</td>
</tr>
<tr>
<td>5.5</td>
<td></td>
<td>82.0</td>
<td>82.9</td>
<td>83.3</td>
<td>83.0</td>
<td>82.7</td>
</tr>
</tbody>
</table>

5. Conclusion

Recent deep GM methods have delivered significant performance gain over traditional ones through learning node/edge features and GM solvers. However, beyond relying on heuristics, there is little work on learning more effective topology for improved matching. In this paper, we hypothesize that learning a better (distribution of) discrete graph topology can significantly improve the matching, thus being essential. As such, we propose to incorporate a latent topology module under an end-to-end deep framework that learns to produce better graph topology. We present the interpretation and optimization of the topology learning module from deterministic and generative perspectives respectively. Experimental results show that, by learning the latent topology, the matching performance can be consistently and significantly enhanced on several public datasets, with only minimal modification to existing method.

Acknowledgements

This work was supported in part by a grant from ONR. Any opinions expressed in this material are those of the authors and do not necessarily reflect the views of ONR.
References


Deep Latent Graph Matching


Xiong, H. and Yan, J. Btwalk: Branching tree random walk for multi-order structured network embedding. IEEE Transactions on Knowledge and Data Engineering, 2020.


