Stochastic Iterative Graph Matching

Linfeng Liu [1] Michael C. Hughes [1] Soha Hassoun [1 2] Li-Ping Liu [1]

Abstract

Recent works apply Graph Neural Networks (GNNs) to graph matching tasks and show promising results. Considering that model outputs are complex matchings, we devise several techniques to improve the learning of GNNs and obtain a new model, Stochastic Iterative Graph MAtching (SIGMA). Our model predicts a distribution of matchings, instead of a single matching, for a graph pair so the model can explore several probable matchings. We further introduce a novel multi-step matching procedure, which learns how to refine a graph pair's matching results incrementally. The model also includes dummy nodes so that the model does not have to find matchings for nodes without correspondence. We fit this model to data via scalable stochastic optimization. We conduct extensive experiments across synthetic graph datasets as well as biochemistry and computer vision applications. Across all tasks, our results show that SIGMA can produce significantly improved graph matching results compared to state-of-the-art models. Ablation studies verify that each of our components (stochastic training, iterative matching, and dummy nodes) offers noticeable improvement.

1. Introduction

Graph matching (Livi and Rizzi, 2013; Yan et al., 2016; Sun et al., 2020) aims to find node correspondence among two or more graphs. It has a wide range of applications such as computer vision (Sun et al., 2020), computational biology (Saraph and Milenković, 2014), and biochemistry (Kotera et al., 2004). Given that many practical graphs cannot be perfectly matched, graph matching often maximizes some matching objective, such as the total number of matched edges (Yan et al., 2020).

Learning-based graph matching (Caetano et al., 2009; Zanfir and Sminchisescu, 2018; Yu et al., 2020; Wang et al., 2019a; Fey et al., 2018) aims to learn a model that can take a pair of graphs and directly "predict" a matching between them. Such models extract information for matching from graph features and carry learned knowledge to new graph matching problems. Unlike labels in typical classification problems, the space of possible matchings is combinatorial, which poses difficulties for learning such models.

Recently there has been remarkable progress in optimizing distributions of discrete structures (Maddison et al., 2016; Paulus et al., 2020). This class of methods approximate discrete random variables with continuous ones and then use the reparameterization technique (Kingma and Welling, 2013; Rezende et al., 2014) to optimize the distributions. In particular, Linderman et al. (2018) and Mena et al. (2018) use this method to learn distributions of permutations and achieve good performances in tasks such as solving jigsaw puzzles. However, learning distributions of matchings is an area left to explore (Paulus et al., 2020).

In this work we apply the stochastic softmax trick (Paulus et al., 2020) to the distribution of matchings and then efficiently learn this distribution through reparameterization. We then use a learning model to parameterize such a distribution to address the graph matching problem, so that the learned model can be applied to any new input graphs. The model is learned to maximize the expected reward under the matching distribution. Comparing to models that directly predict only a single matching, this new model that produces a distribution over matchings is able to explore a wider range of solutions. Furthermore, the stochasticity of the predictive distribution increases the robustness of the learned model, because it is trained to predict a population of good solutions.

An optimal matching for a graph pair often cannot be discovered in one shot, and some refinement often improves the quality of the solution. This is particularly true for a learning model because of the generalization error: it is even hard to guarantee that the predicted matching is a local minimum on a new graph pair. Similar observations are also reported in amortized inference (Marino et al., 2018).

To address this issue, we also design a learnable architecture that can iteratively refine matchings. In addition to

[1] Department of Computer Science, Tufts University, MA, USA [2] Department of Chemical and Biological Engineering, Tufts University, MA, USA. Correspondence to: Linfeng Liu <linfeng.liu@tufts.edu>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

training the model's ability to predict matchings, we also train the model's ability to refine an existing solution for a given graph pair. By maintaining the best matching solution with different refinements, the final prediction will never be worse than the initial prediction.

In addition to our model design, we also have investigated the importance of using dummy nodes (Wang et al., 2019b) in graph matching. Our investigation is motivated by the intended application of matching reactants in molecular reactions. We have the prior knowledge that some nodes cannot be matched. Such nodes can find their place by matching to a dummy node. Our empirical study later indicates the effectiveness of the dummy node on this problem.

With all these considerations, we develop a unified model, Stochastic Iterative Graph MAtching (SIGMA). We then test this new model on three graph matching tasks. The results indicate that the proposed model produces better matching results than state-of-the-art models. We also do an extensive ablation study of the three elements of the model and show the value of each element in graph matching.

To summarize, our contributions in this work include

- the design of a stochastic softmax trick for learning a matching distribution;
- the proposal of a graph matching model that defines a matching distribution;
- the design of iterative refinement for graph matching;
- the premium performances of the proposed model in three graph matching tasks.

2. Related Work

Graph Matching. Traditionally graph matching has been treated as an optimization problem and addressed by various optimization methods. Livi and Rizzi (2013); Yan et al. (2016); Sun et al. (2020); Yan et al. (2020) have made extensive surveys on this topic. Among these traditional methods, the most related works include sampling methods (Lee et al., 2010; Suh et al., 2012). However, these sampling methods are usually computationally expensive.

Graph Neural Networks (GNNs) (Wu et al., 2020) were recently used as learning models for graph matching (Zanfir and Sminchisescu, 2018; Xu et al., 2019b; Wang et al., 2019a;b; Yu et al., 2020; Fey et al., 2020; Nowak et al., 2018; Rolínek et al., 2020). The main idea in this class of work is to use a GNN to encode graph structures into node representation, then nodes are matched based on their vector representation.

Distributions of Permutations. Recently Mena et al. (2018) and Paulus et al. (2020) have developed new methods of learning distributions of permutations. Sharing the idea of the Gumbel-softmax trick (Maddison et al., 2016), these methods devise a continuous sampling procedure that can approximately draw samples of discrete permutation matrices. The sampling procedure allows reparameterization (Rezende et al., 2014; Kingma and Welling, 2013), which enables efficient optimization of the distribution parameters. This work will develop a new distribution for matching from these distributions.

Iterative Refinement in Learning Models. Marino et al. (2018); Krishnan et al. (2018) pointed out that the generalization error of an amortized inference model impedes the inference accuracy on test instances. Then they proposed a learning model that can iteratively refine its solution. This idea is further applied to policy optimization (Marino et al., 2020). Chen et al. (2020) used a similar idea in the task of matching text to images. We will also incorporate the refinement mechanism into the model so it can improve graph matching iteratively.

3. Background

Suppose there are a pair of graphs, $G^s = (V^s, E^s, \mathbf{X}^s)$ and $G^t = (V^t, E^t, \mathbf{X}^t)$. Here $V^s = \{1, \ldots, n_s\}$ the node set of G^s, and E^s is the edge set. We assume graph nodes have known attributes or feature vectors; let $\mathbf{X}^s \in \mathbb{R}^{n_s \times d}$ denote these node features. Similarly, V^t, E^t, $\mathbf{X}^t$, and n_t are, respectively, the node set, the edge set, node features, and the number of nodes of graph G^t. Without loss of generality, we always assume $n_s \leq n_t$.

Graph matching identifies a set of node correspondences between V^s and V^t. Here we add a "dummy node" with id $(n_s + 1)$ to G^s and one with id $(n_t + 1)$ to G^t: if some nodes in one graph cannot be matched with a node from the other graph, they will be matched to the dummy node of the other graph. The correspondences between nodes from the two graphs are indicated by a matrix $\mathbf{M} \in \{0, 1\}^{(n_s+1)\times(n_t+1)}$,

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}^0 & \mathbf{m}_t \\ \mathbf{m}_s^\top & 0 \end{bmatrix},$$
$$\mathbf{1}^\top \mathbf{M}^0 + \mathbf{m}_s^\top = \mathbf{1}^\top, \qquad \mathbf{M}^0 \mathbf{1} + \mathbf{m}_t = \mathbf{1}. \qquad (1)$$

$M_{i,j} = 1$ indicates node i in G^s is matched to node j in G^t. The submatrix $\mathbf{M}^0$ is the matching between normal nodes in G_s and normal nodes in G_t. The two equality constraints say every normal node in G^s or G^t must match exactly one node (either a normal node or the dummy node) from the other graph.

Graph matching aims to find a matching $\mathbf{M}$ that is optimal with respect to an objective $f(\mathbf{M}; G^s, G^t)$, that is,

$$\arg\max_{\mathbf{M}} f(\mathbf{M}; G^s, G^t).$$

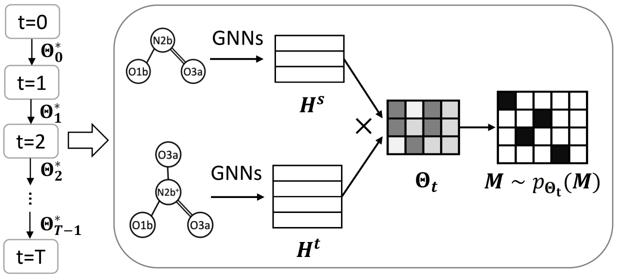

Figure 1. The model architecture of SIGMA. The model predicts Θ_0^* that specifies the initial distribution of matchings. Then each refinement iteration continues to improve the distribution in terms of maximizing the expected objective.

In practice, the objective $f(\cdot)$ can be instantiated in many ways. The Quadratic Assignment Problem (QAP) objective (Yan et al., 2020) is:

$$f_{\text{qap}}(\mathbf{M}; G^s, G^t) = \sum_{\substack{i,i' \in V^s \\ j,j' \in V^t}} K_{i,j,i',j'} M^0_{i,j} M^0_{i',j'}. \quad (2)$$

Here a $K_{i,j,i',j'}$ entry can be viewed as the reward of matching a node pair (i, i') in G^s to a node pair (j, j') in G^t. Different applications may define different reward functions to emphasize application-specific needs. For example, if each entry $K_{i,j,i',j'}$ is 1 when $(i, i') \in E^s$ and $(j, j') \in E^t$ and 0 otherwise, then the objective count the number of matched edges.

In the supervised learning task, we assume each training pair has a known ground truth matching $\mathbf{M}^*$. Then the matching $\mathbf{M}^0$ between normal graph nodes needs to approximate $\mathbf{M}^*$. Fey et al. (2020) relax $\mathbf{M}^0$ to be a continuous variable and compute the negative cross entropy between $\mathbf{M}^0$ and $\mathbf{M}^*$. Formally, the objective is defined as

$$f_{\text{sup}}(\mathbf{M}; \mathbf{M}^*) = \sum_{(i,j) \in V^s \times V^t} M^*_{i,j} \log\left(M^0_{i,j}\right). \quad (3)$$

4. Method

The core part of our model is a learnable distribution of the matching matrix $\mathbf{M}$ between two graphs. We first develop an efficient approach of computing gradients of the distribution parameters. We then develop a learning architecture that can iteratively refine a predicted matching. An overview of the model is given in Figure 1.

4.1. The Matching Distribution

We develop a model that predicts a distribution of matchings, instead of a single matching, for a training pair of graphs. Suppose the model is parameterized by γ and specifies a distribution $p_\gamma(\mathbf{M}|G^s, G^t)$, then we train the model by maximizing the expected matching objective:

$$\max_{\gamma} \mathcal{L} = \mathbb{E}_{p_\gamma(\mathbf{M}|G^s,G^t)}\left[f(\mathbf{M})\right]. \quad (4)$$

In the testing stage, a predicted matching matrix is the MAP estimate from the distribution.

We specify the model in two steps:

$$p_\gamma(\mathbf{M}|G^s, G^t) = p_{\boldsymbol{\Theta}}(\mathbf{M}), \quad \boldsymbol{\Theta} = \text{nn}(G^s, G^t; \gamma). \quad (5)$$

In the first step, neural network $\text{nn}(G^s, G^t; \gamma)$ computes a parameter $\boldsymbol{\Theta}$. In the second step, this parameter defines the distribution over matchings $p_{\boldsymbol{\Theta}}$. We first consider the form of $p_{\boldsymbol{\Theta}}$ and then design the network $\text{nn}(\cdot, \cdot; \gamma)$.

We first consider $p_{\boldsymbol{\Theta}}$ and the gradient $\partial\mathcal{L}/\partial\boldsymbol{\Theta}$ in the optimization (4). Since the integral is often intractable, the gradient with respect to the distribution parameter $\boldsymbol{\Theta}$ (computed from γ) is often estimated through Monte Carlo samples. Given that $p_{\boldsymbol{\Theta}}$ is a discrete probability distribution, an unbiased estimator of $\partial\mathcal{L}/\partial\boldsymbol{\Theta}$ is the score function estimator (Williams, 1992), which typically has large variances and often leads to suboptimal results. This work uses the stochastic softmax trick (Paulus et al., 2020) and derives a low-variance (though biased) estimator of $\partial\mathcal{L}/\partial\boldsymbol{\Theta}$.

We will find a continuous distribution $\hat{p}_{\boldsymbol{\Theta}}(\mathbf{M})$ to "imitate" $p_{\boldsymbol{\Theta}}(\mathbf{M})$: samples from $\hat{p}_{\boldsymbol{\Theta}}$ are approximately samples from $p_{\boldsymbol{\Theta}}$.[1] Then we can compute the expectation in (4) with $\hat{p}_{\boldsymbol{\Theta}}$ and optimize $\boldsymbol{\Theta}$ with the reparameterization technique. To achieve this goal, we leverage the Gumbel-Sinkhorn (GS) distribution described below.

Discrete distribution over permutations. We can first view the graph matching problem as finding a good permutation. Let $\mathbf{S} \in \{0, 1\}^{m \times m}$ define a valid binary permutation matrix, which satisfies $\{\mathbf{S1} = \mathbf{1}, \mathbf{S}^\top\mathbf{1} = \mathbf{1}\}$. To interpret $\mathbf{S}$ for our problem, we set $m = n_s + n_t$. The rows of $\mathbf{S}$ are ordered to represent the nodes of G^s followed by G^t, while the columns are ordered to represent G^t followed by G^s. By selecting this ordering, the top left $n_s \times n_t$ block of $\mathbf{S}$ determines the matching $\mathbf{M}_0$ between G_s and G_t. The bottom n_t rows of $\mathbf{S}$ can be viewed as n_t indistinguishable "dummy nodes" in G^s so that the n_t nodes in G^t have some chance that they all match to these dummy nodes. Similarly the last n_s columns can be considered as n_s dummy nodes in G^t.

We can define a *distribution* over permutations as

$$p_{\boldsymbol{\Phi}}(\mathbf{S}) \propto \exp\left(\text{trace}\left(\mathbf{S}^\top\boldsymbol{\Phi}\right)\right). \quad (6)$$

where $\boldsymbol{\Phi} \in \mathbb{R}^{m \times m}$ is a parameter, and the normalization is over the set of valid permutations.

[1] Here we use the same notation $\mathbf{M}$ for random variables in both $p_{\boldsymbol{\Theta}}$ and $\hat{p}_{\boldsymbol{\Theta}}$ to reduce the load of notations.

Relaxation to continuous variables. The GS distribution (Mena et al., 2018) is developed to imitate the discrete distributions above but uses continuous random variables that make gradient-based training easier. The GS distribution $\hat{p}_{\mathbf{\Phi}}(\mathbf{S})$ is over the space of doubly stochastic matrices of the same size as $\mathbf{S}$, and it is reparameterizable with standard i.i.d. Gumbel noise. We can draw samples from $\hat{p}_{\Phi}(\mathbf{S})$ by running a Sinkhorn procedure over a random matrix (Mena et al., 2018). The resulting samples can accurately approximate samples from $p_{\mathbf{\Phi}}(\mathbf{S})$.

We define our own distribution $p_{\mathbf{\Theta}}(\mathbf{M})$ through $p_{\mathbf{\Phi}}(\mathbf{S})$: use $\mathbf{\Theta}$ to decide $\mathbf{\Phi}$, draw samples from $p_{\mathbf{\Phi}}(\mathbf{S})$, and then transform these samples to samples of $p_{\mathbf{\Theta}}(\mathbf{M})$. Let's first consider the transformation, then we can decide $\mathbf{\Phi}$ in a meaningful way. Suppose $\mathbf{S}$ is a permutation matrix of size (n_s+n_t), then the transformation $\mathbf{M} = \mathbf{BSC}$ gives a sample $\mathbf{M}$:

$$\mathbf{B} = \begin{bmatrix} \mathbf{I}_{n_s} & \\ & \mathbf{1}_{n_t}^{\top} \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} \mathbf{I}_{n_t} & \\ & \mathbf{1}_{n_s} \end{bmatrix}. \tag{7}$$

Here $\mathbf{I}$ is the identity matrix and $\mathbf{1}$ is a column vector, with sizes indicated by subscripts. The rest of the matrix is all zeros. After applying the transformation $\mathbf{M} = \mathbf{BSC}$, the top-left $(n_s \times n_t)$ block of $\mathbf{M}^0$ is the same as in the top-left block of $\mathbf{S}$, which denotes the correspondence between normal nodes in G^s and G^t. The last row and column of $\mathbf{M}$ simply condense all dummy nodes to one aggregate dummy node in each graph.

With this transformation, the matching distribution $p_{\mathbf{\Theta}}(\mathbf{M})$ is automatically defined.

$$p_{\mathbf{\Theta}}(\mathbf{M}) = \sum_{\mathbf{S}:\mathbf{M}=\mathbf{BSC}} p_{\mathbf{\Phi}}(\mathbf{S}).$$

With the same linear transformation $\mathbf{M} = \mathbf{BSC}$, we convert a sample from *the continuous distribution* $\hat{p}_{\mathbf{\Phi}}(\mathbf{S})$ to a sample from $\hat{p}_{\mathbf{\Theta}}(\mathbf{M})$. Then samples of $\hat{p}_{\mathbf{\Theta}}(\mathbf{M})$ are approximate samples from $p_{\mathbf{\Theta}}(\mathbf{M})$. Because the linear transformation is constant, the approximation error of $\hat{p}_{\mathbf{\Theta}}$ is bounded by the approximation error of $\hat{p}_{\mathbf{\Phi}}$ by a constant factor.

We now show how to decide the parameter $\mathbf{\Phi}$ from $\mathbf{\Theta}$,

$$\mathbf{\Phi} = \begin{bmatrix} \mathbf{\Theta} & \mathbf{0}_{n_s \times n_s} \\ \mathbf{0}_{n_t \times n_t} & \mathbf{0}_{n_t \times n_s} \end{bmatrix}. \tag{8}$$

The matrix $\mathbf{\Theta}$ as a block of $\mathbf{\Phi}$ mainly affects the top-left $(n_s \times n_t)$ block of $\mathbf{S}$. Every element $\Theta_{i,j}$ indicates the preference of matching i in G^s to j in G^t: a large positive value favors the matching while a negative value is against the matching. It is not necessary to differentiate nodes matched to either dummy node, so their corresponding parameters are set to zero.

The computation of $\mathbf{\Theta}$. Finally we design the neural network $\mathrm{nn}(G^s, G^t; \gamma)$ that computes $\mathbf{\Theta}$. A GNN is a powerful model for encoding graph structures into vector forms. GNNs have been previously used for graph matching (Fey et al., 2020). We compute the distribution parameter $\mathbf{\Theta}$ via a GNN:

$$\mathbf{\Theta} = \mathbf{H}^s(\mathbf{H}^t)^{\top}, \ \ \mathbf{H}^s = \mathrm{GNN}(G^s; \gamma), \\ \mathbf{H}^t = \mathrm{GNN}(G^t; \gamma). \tag{9}$$

Here $\mathbf{H}^s$ and $\mathbf{H}^t$ are node representations of the graph pair. γ denotes all parameters of the GNN.

Now we have completed the two steps in (5) and have a learning model $p_{\gamma}(\mathbf{M}; G^s, G^t)$ for graph matching. To summarize, we use the GNN with weights γ to compute $\mathbf{\Theta}$ which in turn determines the parameter $\mathbf{\Phi}$ of the permutation distribution. Then, we draw samples from $\hat{p}_{\mathbf{\Phi}}(\mathbf{S})$, and apply the transformation $\mathbf{M} = \mathbf{BSC}$ to obtain approximate samples from $p_{\gamma}(\mathbf{M}; G^s, G^t)$. We can estimate the gradient of an expectation over this distribution with respect to $\mathbf{\Phi}$ using the reparameterization trick.

4.2. Iterative Refinement of the Matching

So far we have a learning model that can predict $\mathbf{\Theta}$ for a pair of graphs. However, it is difficult to produce an optimal matching for two complex graphs in one step. A strategy in searching algorithms is to match "easy" nodes first and gradually expand the matching. We want our learning model to mimic this strategy and refine $\mathbf{\Theta}$ in multiple steps.

A Refinement Model. We first design a learning model $\mathbf{\Theta}_1 = \mathrm{nnr}(\mathbf{\Theta}_0; G^s, G^t)$ that can refine a prediction $\mathbf{\Theta}_0$. The goal for $\mathrm{nnr}(\mathbf{\Theta}_0; G^s, G^t)$ is to move $\mathbf{\Theta}_0$ toward a "better" value, that is, increasing the expected objective.

$$\mathbb{E}_{p_{\mathbf{\Theta}_1}}\left[f(\mathbf{M})\right] \geq \mathbb{E}_{p_{\mathbf{\Theta}_0}}\left[f(\mathbf{M})\right]. \tag{10}$$

The model $\mathrm{nnr}(\mathbf{\Theta}; G^s, G^t)$ and the previous network $\mathrm{nn}(G^s, G^t)$ share the same goal: maximizing the expected objective. The difference is that $\mathrm{nnr}(\cdot)$ can get information about the previous matching from $\mathbf{\Theta}_0$, which helps to revise prior matchings.

Ideally the model $\mathrm{nnr}(\cdot)$ should preserve good partial matching and use it to inform further matching of more nodes. Guided by this principle, we give more weights to nodes that are better matched in the previous step. We first compute weight vectors for the two graphs from $\mathbf{\Theta}_0$. Let $\bar{\mathbf{M}}$ be an average of ℓ samples of $p_{\mathbf{\Theta}_0}(\mathbf{M})$, and let

$$\mathbf{a}_s = \bar{\mathbf{M}}_{1:n_s, 1:n_t}\mathbf{1}, \quad \mathbf{a}_t = \bar{\mathbf{M}}_{1:n_s, 1:n_t}^{\top}\mathbf{1}. \tag{11}$$

Each entry in vector $\mathbf{a}_s \in [0, 1]^{n_s}$ indicates the probability that a node in G^s is matched to a normal node in G^t. It is similar for the vector $\mathbf{a}_t \in [0, 1]^{n_t}$.

Then we use the two vectors to reweight the importance of node features in the GNN computation.

$$\boldsymbol{\Theta}_1 = \mathrm{nn}(G^s_u, G^t_u), \quad G^s_u = (V^s, E^s, \mathrm{diag}(\mathbf{a}_s)\mathbf{X}^s), \\ G^t_u = (V^t, E^t, \mathrm{diag}(\mathbf{a}_t)\mathbf{X}^t). \tag{12}$$

Here the neural network $\mathrm{nn}(\cdot,\cdot)$ is the same one in (9). For simplicity, we train one $\mathrm{nn}(\cdot,\cdot)$ model, but we do make sure the model capacity is enough in practice (e.g. try large hidden dimensions and deep models). Putting (11) and (12) together, we have the refinement model $\mathrm{nnr}(\boldsymbol{\Theta}; G^s, G^t)$.

In this design, clustered matched nodes tend to have small changes in their representations so their matching is unlikely to change in the refinement step. Furthermore, these nodes encode the matched structure in their messages to their unmatched neighbors. Note that the message passing in the first layer of the GNN is as if from a weighted graph now. Then the refinement can expand the matching using information from the matched structure. It shares the same principle as searching algorithms (McCreesh et al., 2017).

Note that the iterative procedure works on the distribution parameter $\boldsymbol{\Theta}$ instead of discrete matchings. The computation $\mathrm{nnr}(\cdot)$ is thus stochastic since $\bar{\mathbf{M}}$ in (11) is the average of a small number of samples. The stochasticity allows the model to explore different directions in multiple steps.

Multi-step Refinement. We apply the refinement model to a multi-step procedure. Let $\boldsymbol{\Theta}^*_0 = \mathrm{nn}(G^s, G^t)$ be the initial matching distribution. At each step $t = 1, \ldots, T$,

$$\boldsymbol{\Theta}_t = \mathrm{nnr}(\boldsymbol{\Theta}^*_{t-1}; G^s, G^t) \tag{13}$$

$$\boldsymbol{\Theta}^*_t = \begin{cases} \boldsymbol{\Theta}_t & \text{if } \mathbb{E}_{p_{\boldsymbol{\Theta}_t}}[f(\mathbf{M})] \geq \mathbb{E}_{p_{\boldsymbol{\Theta}^*_{t-1}}}[f(\mathbf{M})], \\ \boldsymbol{\Theta}^*_{t-1} & \text{otherwise.} \end{cases} \tag{14}$$

The two expectations are estimated by samples. To train the refinement model, we need to track gradients for each $\boldsymbol{\Theta}^*_t$. We do not track gradients with respect to the previous iteration parameter $\boldsymbol{\Theta}^*_{t-1}$ to stabilize the training procedure.

The refinement procedure will not return a solution worse than the beginning one since the best $\boldsymbol{\Theta}^*_t$ is always kept. If the beginning solution is already very good and hard to improve, then the model will make multiple attempts to improve it. These attempts will not be the same due to the randomness of $\bar{\mathbf{M}}$. Differing from DGMC's deterministic refinement (Fey et al., 2020), our refinement is stochastic. Algorithm 1 summarizes our multi-step matching refinement.

4.3. Training and Prediction

The goal of training is to optimize the parameter γ, the parameters of the GNN used in both the first-step prediction as well as the refinement procedure.

Algorithm 1 Iterative Refinement

```
Input: $G^s$, $G^t$, $T$, $f(\cdot)$
Initialize $\boldsymbol{\Theta}^*_0 = \mathrm{nn}(G^s, G^t)$
for $t = 1$ to $T$ do
    Compute $\boldsymbol{\Theta}_t = \mathrm{nnr}(\boldsymbol{\Theta}^*_{t-1}; G^s, G^t)$
    if $\mathbb{E}_{p_{\boldsymbol{\Theta}_t}}[f(\mathbf{M})] \geq \mathbb{E}_{p_{\boldsymbol{\Theta}^*_{t-1}}}[f(\mathbf{M})]$ then
        $\boldsymbol{\Theta}^*_t = \boldsymbol{\Theta}_t$
    else
        $\boldsymbol{\Theta}^*_t = \boldsymbol{\Theta}^*_{t-1}$
    end if
end for
Return $\{\boldsymbol{\Theta}^*_0, \boldsymbol{\Theta}^*_1, \ldots, \boldsymbol{\Theta}^*_T\}$
```

This work considers the following matching objective,

$$f(\mathbf{M}) = f_{\mathrm{qap}}(\mathbf{M}) + \lambda f_{\mathrm{sup}}(\mathbf{M}). \tag{15}$$

Here $f_{\mathrm{qap}}(\mathbf{M})$ and $f_{\mathrm{sup}}(\mathbf{M})$ are from (2) and (3). It covers a wide range of applications. In unsupervised problems, we don't have the second term; in supervised problems, the hyperparameter λ balances the importance of the two objectives. We will give more details in the calculation of these objectives in the experiment section.

After we have included the refinement procedure, the training objective of the entire model for one graph pair (G_s, G_t) becomes $\sum_{t=0}^{T} \mathbb{E}_{p_{\boldsymbol{\Theta}_t}}[f(\mathbf{M})]$. Here each $\boldsymbol{\Theta}_t$ is computed by the GNN with parameter γ. We weigh loss at each step t equally: the first steps are important because they impact later steps, while the last steps are also important because they are likely to give the final solution.

After training, we need to compute a single matching as the prediction of the model. Our matching model uses the same principle as a typical classification model: using the mode of the predictive distribution as the prediction, though it is harder to find the mode of the matching distribution. The prediction from the distribution $p_{\boldsymbol{\Theta}^*_T}(\mathbf{M})$ is computed from the following problem,

$$\mathbf{M} = \arg\max_{\mathbf{M}} \ \mathrm{trace}(\mathbf{M}^\top \mathrm{pad}(\boldsymbol{\Theta}^*_T)) \tag{16}$$

$$= \arg\max_{\mathbf{M}=\mathbf{BSC}} \ \mathrm{trace}(\mathbf{S}^\top \boldsymbol{\Phi}). \tag{17}$$

Here $\mathrm{pad}(\boldsymbol{\Theta}^*_T)$ adds a row and a column of zeros to $\boldsymbol{\Theta}^*_T$; $\boldsymbol{\Phi}$ contains $\boldsymbol{\Theta}^*_T$ as its top-left block as in (8). We use Hungarian algorithm to solve the form in the second line.

Note that when making predictions of the matching, we do not assume any access to ground truth matchings because they will not be available at test time. Thus, our iterative refinement reward function only includes unsupervised terms.

Algorithm	BA(500)§	BA(500)	BA(1000)	BA(2000)	PPI§	PPI
MCSPLIT	**100.0±0.0** (1)	43.7±0.3 (1000)	16.9±8.9 (1000)	33.0±7.3 (1000)	**100.0±0.0** (1)	60.3 (1000)
S-GWL	47.5±0.5 (3)	32.0±1.5 (3)	22.9±0.3 (14)	16.2±0.4 (129)	83.1 (50)	81.1 (50)
SIGMA	99.2±0.2 (10)	**93.8±0.3** (11)	**97.3±0.2** (33)	**99.0±0.1** (179)	99.2±0.2 (53)	**84.7±0.4** (67)

Table 1. Node correctness (%) and runtime in parenthesis (in seconds). Datasets with § means 0% noises; without § means 5% noises.

Setting	BA(500)	BA(1000)	BA(2000)	PPI
T=0	93.5±0.1	97.2±0.1	98.9±0.1	83.1±0.3
T=3	93.7±0.2	97.2±0.3	98.9±0.1	83.8±0.2
T=4 (default)	93.8±0.3	97.3±0.2	99.0±0.1	84.7±0.4
T=9	93.8±0.2	97.3±0.1	99.1±0.1	85.0±0.4
Remove S	93.6±0.1	97.1±0.2	98.9±0.1	84.1±0.4
Remove D	50.3±4.1	75.0±5.9	80.4±3.1	83.2±0.2

Table 2. Ablation results on common graph matching. Results are reported in node correctness (%). T: the number of refinement after initial prediction; S: stochastic framework; D: dummy nodes.

5. Experiments

We test our model on three tasks: a common graph matching task, a biochemistry application of matching reaction centers among molecular reactants, and a computer vision application of matching keypoints between images.

5.1. Common Graph Matching

Dataset We use two datasets in this task, and follow the experiment setting from Xu et al. (2019a). In the first dataset, we use a Barabási-Albert (BA) model to generate graphs of $\{500, 1000, 2000\}$ nodes. To create matching graph pairs, we first sample a source graph from the BA model, then corrupted the source graph by adding 5% noisy edges as a target graph[2]. In the second dataset, we start from the Protein-Protein Interaction (PPI) network of yeast (1,004 proteins and 4,920 interactions), and align its 5% noisy version provided in Saraph and Milenković (2014). In both datasets, each node's input feature is assigned according to its node degree. We also include the noise-free versions of the two datasets to match, where the target graph is the same as the source graph.

Experiment Setting We compare our model with MCSPLIT (McCreesh et al., 2017) and S-GWL (Xu et al., 2019a). MCSPLIT, which uses branching heuristic to reduce the search space, is a state-of-the-art heuristic method to find an isomorphic subgraph, but it performs poorly when a few "noise" edges are added to the graph. S-GWL, on the other hand, solves the matching problem under the Gromov-Wasserstein discrepancy (Chowdhury and Mémoli, 2019) and has shown robustness to moderate amount of "noise" edges added to a graph. For these baselines, we use the authors' implementations with their default hyperparameters. Xu et al. (2019a) shows that S-GWL has outperformed most heuristic methods on matching noisy graphs.

For our model, we instantiate the GNN as a 5-layer Graph Isomorphism Network (GIN) (Xu et al., 2018). Each layer of GIN has a one-layer MLP with hidden dimension of 256 followed by a $\mathrm{tanh}(\cdot)$ activation. The model is optimized by an Adam optimizer (Kingma and Ba, 2014) at a learning rate 10^{-4} and trains for 100 epochs. For each dataset, the epoch that produces the best objective is used for testing. We use 10 samples of $\mathbf{M}$. T is set to 4 (1 initial prediction followed by 4 iterations of refinement). To evaluate, we report node correctness (NC) as in Xu et al. (2019a), which denotes the percentage of nodes that have the same matching as ground truth.

Our model is implemented in PyTorch (Paszke et al., 2017). Each model runs on a server with 32 cores and an NVIDIA A100 (40GB) GPU.

Results Results are shown in Table 1. Our model, SIGMA, outperforms baselines in matching noisy graphs. As expected, the heuristic method MCSPLIT fails to cope with noisy graphs. SIGMA outperforms S-GWL on all datasets. We conjecture our improvement stems from learning a GNN that can provide discriminative node embeddings suited for matching. We see comparable matching results between SIGMA and MCSPLIT when matching noise-free graphs. SIGMA does not attain a perfect match; we speculate some nodes lie in symmetric structures in both graphs, and the underlying GNN cannot provide distinguishable embeddings for them. The runtime of SIGMA is competitive to the other two baselines.

We provide an ablation study of the three components: multi-step matching (T), stochastic framework (S), and dummy nodes (D). The ablation study is conducted on noisy graphs to show noticeable performance differences. Results are given in Table 2. By comparing different T, we see more refinement steps yields noticeable improvements on the PPI dataset, though minor improvements on the BA datasets. Using dummy nodes has a clear advantage on this task. When removing the dummy node, the prediction correctness drops up to 43% on the BA dataset, and 1.5% on the PPI dataset. Some "hard" nodes may have been aligned with dummy nodes and make the matching problem easier. Performance gain using stochasticity is limited in this task.

[2] We follow the script of S-GWL (Xu et al., 2019a): `https://github.com/HongtengXu/s-gwl`.

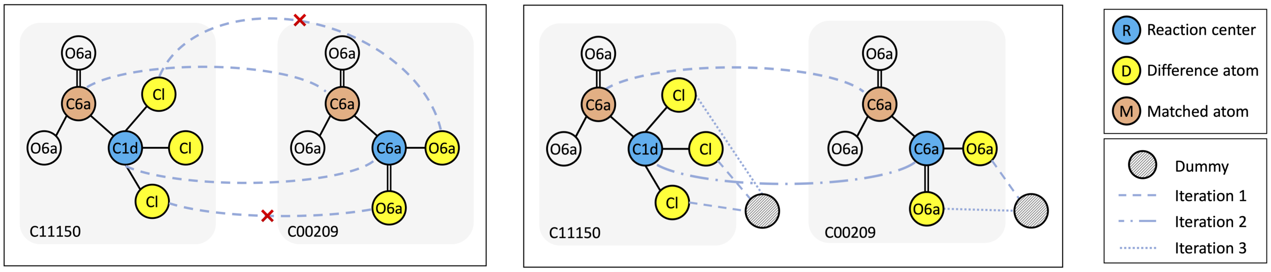

Figure 2. An example of matching RDM between C11150 (Trichloroacetate acid) and C00209 (Oxalate acid). Left: DGMC. Right: SIGMA. In SIGMA, difference atoms (D) are correctly aligned to dummy nodes. SIGMA finds the best match in 3 iterations (1 initial prediction followed by 2 refinements): SIGMA matched *matched atoms (M)* and some *difference atoms (D)* in iteration 1; matched *reaction center (R)* in iteration 2; and matched the rest of *difference atoms (D)* in iteration 3. Dummy nodes are located for the visualization purpose.

Algorithm	Hard Match	Soft Match	NC
MCSPLIT	36.0±1.5	54.1±1.2	56.0±1.2
DGMC	37.3±1.5	66.0±1.0	66.9±0.9
SIGMA^U	39.5±1.5	63.1±1.1	66.2±1.0
SIGMA (T=0)	48.3±1.5	74.8±0.9	76.4±0.8
SIGMA (*w/o* S)	50.6±1.4	75.1±0.9	76.0±0.8
SIGMA (*w/o* D)	32.3±1.4	59.7±1.0	63.6±0.9
SIGMA	**58.0±1.3**	**78.2±0.9**	**78.3±0.8**

Table 3. Results (in %) on RDM pattern matching. SIGMA^U: Training SIGMA with the QAP objective only (unsupervised).

5.2. RDM Pattern Matching in KEGG RPAIR

Dataset In the KEGG database[3], RDM stands for the reaction center atom (R), the difference atoms (D), and the matched atoms (M). RDM patterns record the structural transformation patterns between a reactant pair in an enzymatic reaction (Kotera et al., 2004). The RPAIR database in KEGG specifies the RDM pattern alignment between two reactant molecules. The difference atoms are those connected to the reaction center that cannot be matched between the reactants. An example is presented in Figure 2. In this task, the aim is to predict the RDM pattern given reactant pairs (and not the overall alignment).

For each reactant in a reactant pair, a graph is constructed from the corresponding molecular description using data collected from the KEGG database. Nodes in the graph represent atoms and edges represent bonds. We set node features as the KEGG atom types, which encode for an atomic species (e.g., Carbon, Oxygen, etc) and its neighboring atoms (Kotera et al., 2004). The edge features are the bond types (single, double, or triple bonds – three types in total). The ground truth RDM pattern for each reactant pair is a set of node correspondences. These node correspondences are used as the supervised objective during training (the calculation of objective is the "soft match" we will define in the experiment setting), and used as the label during testing. We collect 10,366 reactant pairs. On average, each reactant contains 23 nodes and 48 edges. We split the dataset into training, validation, testing at ratio 8:1:1.

Experiment Setting We compare with MCSPLIT (unsupervised) and Deep Graph Matching Consensus (DGMC, supervised) (Fey et al., 2020). As edges have discrete features, a 3-layer Relational GCN (RGCN) (Schlichtkrull et al., 2018) is used as the backbone GNN. We adopt DGMC to use the same GNN structure as ours for a fair comparison. Node features are augmented with a positional feature, which is the eigenvectors of the Laplacian matrix that correspond to the largest 50 eigenvalues (padding zeros otherwise). We found using the positional features improves the matching quality. Separate MLPs are used to embed node's atom feature and position feature. The concatenation of the two resultant embeddings are used as GNN's input. We apply batch normalization between RGCN layers, and use dropout at rate 0.5 and L2 regularizer at weight 10^{-4}. The model is trained for 30 epochs. Other settings remain the same as in section 5.1.

To evaluate, we consider a hard match, a soft match, and the node correctness. For each reactant pair, the hard match denotes whether the prediction fully matches the RDM pattern; it takes value 1 or 0. The soft match is a relaxed version of the hard match; it shows the fraction of node pairs matched in RDM and takes a value between 0 and 1. During training, the soft match is the supervised objective.

Results Results are given in Table 3. SIGMA attains the top performance across all metrics. We also run SIGMA in an unsupervised setting (denoted as SIGMA^U), where SIGMA is trained with the QAP objective only. As expected, the performance degenerates. This result emphasizes the demand of designing a learning model that supports various objective function. Ablation results show each of our three components plays a vital role.

[3] https://www.genome.jp/kegg/reaction

Algorithm	Aero	Bike	Bird	Boat	Bottle	Bus	Car	Cat	Chair	Cow	Table	Dog	Horse	M-Bike	Person	Plant	Sheep	Sofa	Train	TV	Mean
GMN	31.1	46.2	58.2	45.9	70.6	76.5	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
PCA-GM	40.9	55.0	65.8	47.9	76.9	77.9	63.5	67.4	33.7	66.5	63.6	61.3	58.9	62.8	44.9	77.5	67.4	57.5	86.7	90.9	63.8
CIE	51.2	69.2	70.1	55.0	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
DGMC	47.0	65.7	56.8	67.6	86.9	87.7	85.3	72.6	42.9	69.1	84.5	63.8	78.1	55.6	58.4	98.0	68.4	92.2	94.5	85.5	73.0
DGMC*	46.3	64.5	54.9	69.4	85.7	87.8	85.2	73.4	38.2	64.0	92.4	63.6	74.7	60.5	61.6	96.6	63.7	97.6	94.0	86.2	73.0
SIGMA^U	26.5	43.1	40.3	66.5	85.3	86.3	73.4	43.8	29.9	40.7	94.5	33.8	61.0	39.8	52.1	95.8	40.9	95.5	92.2	84.6	61.3
SIGMA (T=0)	54.0	69.9	59.9	72.3	87.4	87.9	87.0	74.0	46.2	68.5	92.4	67.9	77.3	66.9	64.4	97.2	71.6	96.7	94.7	84.6	76.0
SIGMA (*w/o* S)	49.6	65.9	55.0	69.7	86.8	86.4	84.6	71.6	42.0	64.8	91.1	64.8	75.3	60.6	59.8	96.5	66.7	93.7	94.4	83.8	73.1
SIGMA (*w/o* D)	56.6	71.1	59.6	71.6	87.6	83.9	89.0	74.4	46.9	71.7	86.9	69.2	80.0	69.4	64.1	96.8	71.4	96.7	94.9	87.3	**76.5**
SIGMA	55.1	70.6	57.8	71.3	88.0	88.6	88.2	75.5	46.8	70.9	90.4	66.5	78.0	67.5	65.0	96.7	68.5	97.9	94.3	86.1	76.2

Table 4. Hits@1 (%) on PASCAL VOC with Berkeley annotations. Compared methods are GMN (Zanfir and Sminchisescu, 2018), PCA-GM (Wang et al., 2019a), CIE (Yu et al., 2020), and DGMC (Fey et al., 2020).

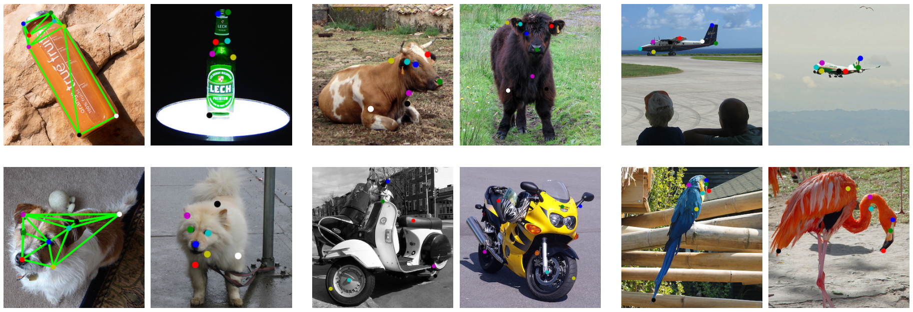

Figure 3. Qualitative examples of SIGMA^U (the first two column) and SIGMA (the last four columns). In each pair of the images, the left shows the source image and the right shows the target. Dots with the same color denote matched nodes. Green lines in the first column are edges of between keypoints. SIGMA correctly identifies keypoints with changed poses.

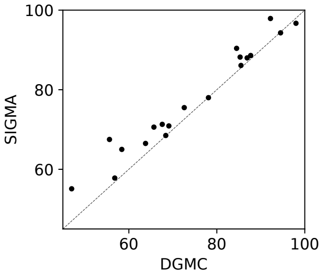

Figure 4. Comparison of Hit@1 % between SIGMA and DGMC over each category.

In Figure 4, we illustrate a comparison between DGMC and SIGMA. SIGMA correctly matches the RDM pattern, where difference atoms are aligned to dummy nodes. The result further shows the effectiveness of our multi-step refinement: the refinement process matches the three different RDM regions in a progressive way.

5.3. Image Keypoints Matching

Dataset In this task, we match keypoints on PASCAL VOC (Everingham et al., 2010) with Berkeley keypoint annotations (Bourdev and Malik, 2009). The dataset is difficult, because it contains images of various scales, poses and illuminations (Wang et al., 2019a). In total, the dataset contains 20 classes of objects. On average, each class contains 348 training graphs and 84 test graphs, where each graph contains 1 to 12 keypoints and 1 to 27 edges. We follow the experimental settings as Deep Graph Matching Consensus (DGMC) (Fey et al., 2020). The original dataset is filtered to 6,953 images for training and 1,671 images for testing. Difficult, occluded, and truncated objects are excluded, and each image has at least one keypoint. For keypoints features, we use a concatenation of the output of `relu4_2` and `relu5_1` from a pre-trained VGG16 (Simonyan and Zisserman, 2014) on ImageNet (Deng et al., 2009). Edge features are normalized 2D Cartesian coordinates (the anisotropic setting from DGMC).

Experiment Setting We compare with four state-of-the-art baselines: GMN (Zanfir and Sminchisescu, 2018), PCA-GM (Wang et al., 2019a), CIE (Yu et al., 2020), and DGMC (Fey et al., 2020). Following DGMC, we set our backbone GNN as a SplineCNN (Fey et al., 2018). Most of the hyperparameters are kept the same as DGMC. The kernel size is 5 in each dimension. We stack 2 convolutional layers, followed by a dropout layer with probability 0.5, which in turn is followed by a linear layer that outputs node embeddings $\mathbf{H}$. Unlike DGMC, we found a hidden dimension of 512 and a LeakyReLU activation with a negative slope 0.1 work well for our model (DGMC uses 256 hidden layers and a ReLU activation). For a fair comparison, we also run DGMC with this setting and name it as DGMC*. Following DGMC, the supervised objective is defined as in (3). We report Hit@1 to evaluate the performance. Hit@1 shows the percentage of correct matched instances over the whole.

Results Results are given in Table 4. SIGMA significantly outperforms GMN and PCA-GM by over 10% of average Hit@1 score, and a 7.3% improvement upon CIE. We also observe a 3.2% of Hit@1 improvement over the DGMC. In Figure 4, we further confirm SIGMA produces better matching over most of the categories. In this dataset, the stochastic framework brings the most performance gain. Removing the stochasticity, which means optimizing (through relaxation) a single matching $\mathbf{M}$ in (1), drops the Hit@1

Algorithm	PASCAL VOC	SPair-71k
BB-GM	80.1±0.6	78.9±0.4
SIGMA	**81.2±0.2**	**79.8±0.2**

Table 5. Mean Hits@1 (%) on PASCAL VOC and SPair-71k. Compared method is BB-GM (Rolínek et al., 2020). SIGMA results are reported following BB-GM's experiment setting.

score by 3.1%. On average, the effectiveness of dummy nodes and multi-step matching is minor on this task.

Our unsupervised setting, SIGMA^U, shows a 3.4% improvement over GMN on the average score and is on par performance with PCA-GM. Note that both GMN and PCA-GM are supervised. The QAP objective determines the predictive capacity of SIGMA^U. Once the QAP objective can recognize the input graph's topology, SIGMA^U has the potential to perform matching well. In the first two columns of Figure 3, we see SIGMA^U successfully matches the bottle image pair but partially matches the dog image pair. We guess that the bottle image pair presents a more recognizable graph structure to the QAP objective (edges in green lines) than the dog image pair. The last four columns in Figure 3 visualize four correctly matched samples from SIGMA. SIGMA recovers node correspondences under various pose changes.

Lastly, we compare SIGMA with recent method BB-GM (Rolínek et al., 2020) using BB-GM's representation learning method. Note that BB-GM focuses on learning representations and uses a match solver as a blackbox. We follow BB-GM's experiment setting, and the main differences from the previous experiment include: 1) fine-tuning VGG16's weights and 2) computing node affinities $\mathbf{\Theta}$ from a weighted inner product (the weights of the inner product are from the final VGG16 layer). Then we evaluate both models on PASCAL VOC and SPair-71 (Min et al., 2019). SPair-71k is similar to PASCAL VOC, but contains higher quality images and richer keypoints annotations. Table 5 shows that SIGMA outperforms BB-GM in terms of mean Hits@1.

6. Conclusion

We have introduced a new learning model, SIGMA, that addresses graph matching problems. We presented two innovations in the design of this new model. First, the model learns a distribution of matchings, instead of a single matching, between a pair of graphs. Second, the model learns to refine matchings through attending to matched nodes. SIGMA consistently shows better performance than other methods in terms of the matching quality, and at the same time, retains a comparable running speed as the baselines.

SIGMA opens many possible directions. Since matchings sampled from SIGMA are still like discrete variables, high-level graph structure can still be well defined on these samples. Therefore, SIGMA can be applied to match high-order graph structures such as paths and hyper-edges. With slight modification, SIGMA can be applied to matching problems beyond graph matching, such as matching tabular data to knowledge graphs. We hope that this work not only brings a new tool for graph matching but also inspires further research in this direction.

Acknowledgements

We thank all reviewers for their insightful comments. This research was supported by NSF 1908617 and NSF 1909536. Soha Hassoun's research is also supported by the NIGMS of the National Institutes of Health, Award R01GM132391. The content is solely the responsibility of the authors and does not necessarily represent the official views of the NIH.

References

Bourdev, L. and Malik, J. (2009). Poselets: Body part detectors trained using 3d human pose annotations. In *2009 IEEE 12th International Conference on Computer Vision*, pages 1365–1372. IEEE.

Caetano, T. S., McAuley, J. J., Cheng, L., Le, Q. V., and Smola, A. J. (2009). Learning graph matching. *IEEE transactions on pattern analysis and machine intelligence*, 31(6):1048–1058.

Chen, H., Ding, G., Liu, X., Lin, Z., Liu, J., and Han, J. (2020). Imram: Iterative matching with recurrent attention memory for cross-modal image-text retrieval. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12655–12663.

Chowdhury, S. and Mémoli, F. (2019). The gromov–wasserstein distance between networks and stable network invariants. *Information and Inference: A Journal of the IMA*, 8(4):757–787.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338.

Fey, M., Lenssen, J. E., Morris, C., Masci, J., and Kriege, N. M. (2020). Deep graph matching consensus. In *International Conference on Learning Representations*.

Fey, M., Lenssen, J. E., Weichert, F., and Müller, H. (2018). Splinecnn: Fast geometric deep learning with continuous b-spline kernels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 869–877.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kotera, M., Okuno, Y., Hattori, M., Goto, S., and Kanehisa, M. (2004). Computational assignment of the ec numbers for genomic-scale analysis of enzymatic reactions. *Journal of the American Chemical Society*, 126(50):16487–16498.

Krishnan, R., Liang, D., and Hoffman, M. (2018). On the challenges of learning with inference networks on sparse, high-dimensional data. In *International Conference on Artificial Intelligence and Statistics*, pages 143–151. PMLR.

Lee, J., Cho, M., and Lee, K. M. (2010). A graph matching algorithm using data-driven markov chain monte carlo sampling. In *2010 20th International Conference on Pattern Recognition*, pages 2816–2819. IEEE.

Linderman, S., Mena, G., Cooper, H., Paninski, L., and Cunningham, J. (2018). Reparameterizing the birkhoff polytope for variational permutation inference. In *International Conference on Artificial Intelligence and Statistics*, pages 1618–1627. PMLR.

Livi, L. and Rizzi, A. (2013). The graph matching problem. *Pattern Analysis and Applications*, 16(3):253–283.

Maddison, C. J., Mnih, A., and Teh, Y. W. (2016). The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*.

Marino, J., Piché, A., Ialongo, A. D., and Yue, Y. (2020). Iterative amortized policy optimization. *arXiv preprint arXiv:2010.10670*.

Marino, J., Yue, Y., and Mandt, S. (2018). Iterative amortized inference. *arXiv preprint arXiv:1807.09356*.

McCreesh, C., Prosser, P., and Trimble, J. (2017). A partitioning algorithm for maximum common subgraph problems. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 712–719.

Mena, G., Belanger, D., Linderman, S., and Snoek, J. (2018). Learning latent permutations with gumbel-sinkhorn networks. *arXiv preprint arXiv:1802.08665*.

Min, J., Lee, J., Ponce, J., and Cho, M. (2019). Spair-71k: A large-scale benchmark for semantic correspondence. *arXiv preprint arXiv:1908.10543*.

Nowak, A., Villar, S., Bandeira, A. S., and Bruna, J. (2018). Revised note on learning quadratic assignment with graph neural networks. In *2018 IEEE Data Science Workshop (DSW)*, pages 1–5. IEEE.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch. *NIPS-W*.

Paulus, M. B., Choi, D., Tarlow, D., Krause, A., and Maddison, C. J. (2020). Gradient estimation with stochastic softmax tricks. *arXiv preprint arXiv:2006.08063*.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pages 1278–1286. PMLR.

Rolínek, M., Swoboda, P., Zietlow, D., Paulus, A., Musil, V., and Martius, G. (2020). Deep graph matching via blackbox differentiation of combinatorial solvers. In *European Conference on Computer Vision*, pages 407–424. Springer.

Saraph, V. and Milenković, T. (2014). Magna: maximizing accuracy in global network alignment. *Bioinformatics*, 30(20):2931–2940.

Schlichtkrull, M., Kipf, T. N., Bloem, P., Van Den Berg, R., Titov, I., and Welling, M. (2018). Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Suh, Y., Cho, M., and Lee, K. M. (2012). Graph matching via sequential monte carlo. In *European Conference on Computer Vision*, pages 624–637. Springer.

Sun, H., Zhou, W., and Fei, M. (2020). A survey on graph matching in computer vision. In *2020 13th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pages 225–230. IEEE.

Wang, R., Yan, J., and Yang, X. (2019a). Learning combinatorial embedding networks for deep graph matching. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3056–3065.

Wang, R., Yan, J., and Yang, X. (2019b). Neural graph matching network: Learning lawler's quadratic assignment problem with extension to hypergraph and multiple-graph matching. *arXiv preprint arXiv:1911.11308*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. (2020). A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*.

Xu, H., Luo, D., and Carin, L. (2019a). Scalable gromov-wasserstein learning for graph partitioning and matching. In *Advances in neural information processing systems*, pages 3052–3062.

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

Xu, K., Wang, L., Yu, M., Feng, Y., Song, Y., Wang, Z., and Yu, D. (2019b). Cross-lingual knowledge graph alignment via graph matching neural network. pages 3156–3161. Association for Computational Linguistics.

Yan, J., Yang, S., and Hancock, E. R. (2020). Learning for graph matching and related combinatorial optimization problems. In *International Joint Conference on Artificial Intelligence*. York.

Yan, J., Yin, X.-C., Lin, W., Deng, C., Zha, H., and Yang, X. (2016). A short survey of recent advances in graph matching. In *Proceedings of the 2016 ACM on International Conference on Multimedia Retrieval*, pages 167–174.

Yu, T., Wang, R., Yan, J., and Li, B. (2020). Learning deep graph matching with channel-independent embedding and hungarian attention. In *International conference on learning representations*.

Zanfir, A. and Sminchisescu, C. (2018). Deep learning of graph matching. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2684–2693.