Learning Graph Distances via Graph Neural Networks and Node Matching

Johannes Pitz

Guided Research: Data Analytics and Machine Learning Group
TUM Department of Informatics

Abstract

Computing meaningful distances between graphs is often difficult due to the combinatorial explosion of possible transformations on graphs. Recently, researchers started using graph neural networks in an attempt to replace classical search algorithms with learning methods. However, these learning methods often rely on fixed-size graph embeddings, which prevent them from scaling to larger graphs. We propose to use soft matchings between nodes to overcome this problem. Our novel combination of a classical cost matrix, motivated by bipartite graph matching algorithms, and the modern Sinkhorn distance shows state-of-the-art results for predicting the ubiquitous graph edit distance. Additionally, we report strong empirical results at graph matching with the same model. Our implementation[1] efficiently handles sparse inputs and large real-world graphs.

1 Introduction

Most conceivable datasets can be represented as a graph or a set of graphs. Not only "graph-like" datasets such as social networks or chemical molecules fall into this category, but also, i.e. executable binaries can be represented by control flow graphs or pictures can be represented by the graph of keypoints. The broad applicability of graphs and the success of deep learning across different tasks in computer vision, reinforcement learning, and natural language processing has spurred research in the application of neural networks to graphs. Graph Neural Networks (GNN) utilize message passing between connected nodes to efficiently handle sparse inputs. They have already shown great success in predicting properties of nodes (Kipf and Welling (2017)) and entire graphs (Gilmer et al., 2017). However, only recently researchers started applying GNNs to tasks such as graph distance/similarity estimation and graph matching.

A metric or a distance function defines a distance between pairs of elements of a set, and the inverse of a distance function is often used as a similarity measure. Finding similar graphs can be crucial for fingerprint matching (Neuhaus and Bunke, 2005), image indexing (Xiao et al., 2008), or for searching similar chemical compounds (Birchall et al., 2006). Often the graph edit distance (GED) is used as a proxy for finding similar graphs. Since computing the GED is NP-complete (Bunke and Shearer, 1998), it usually needs to be approximated. There are multiple classical approximating algorithms, some of them are guaranteed to find a lower bound (Riesen and Bunke, 2009) or upper bound (Fischer et al., 2015). However, due to their origin in optimization there is always a tradeoff between speed and accuracy, in which the faster approximations consider only very local node structures (Riesen and Bunke, 2009). Recently, trained GNNs have been used to improve upon classical algorithms in terms of accuracy and inference time (Bai et al., 2019). One key advantage of neural network models is that they are not limited to computing the GED, but they can learn to approximate given similarity functions. Theoretical work of Loukas (2019) shows that graph neural networks can in principle compute any function on its inputs that is computable by a Turing machine. Practical examples include detecting fraudulent binaries (Li et al., 2019), or graph-based keyword spotting (Riba et al., 2018).

A graph matching establishes a node correspondence between graphs, maximizing corresponding node and edge affinity (Wang et al., 2019). Matching two graphs can be interesting for linking user accounts across different social network graphs (Zhang et al., 2016), matching features in images (Zhang and Lee, 2019), tracking keypoints of moving objects (Vento and Foggia, 2012), or aligning protein networks in bioinformatics (Singh et al., 2008).

[1] `https://gitlab.lrz.de/gdn/graph-distance`

Previously designed neural network models for graph distance estimation either reduce the entire graph to a single embedding vector creating an information bottleneck if scaling to larger graphs (Bai et al., 2019, Li et al., 2019), or they implicitly use a crude and strict matching (Riba et al., 2018).

We propose a model for learning metrics on graphs by using a novel combination of a classical cost matrix and an explicit soft matching. The main contributions of this report are:

- The Graph Distance Network (GDN): A model derived from properties of distance functions and classical algorithms for approximating the GED that uses a cost matrix and a soft matching with analytic gradients.
- Experiments showing that our model improves upon state-of-the-art neural networks in predicting the GED.
- Empirical evidence that our different application of the Sinkhorn normalization improves upon results of Fey et al. (2020) in graph matching.

2 Background

Let $\mathcal{G}$ be a set of graphs, where each graph $G = \{V, E\} \in \mathcal{G}$ with nodes V and edges E is drawn from a common distribution. The nodes and edges may have arbitrary features. Let N denote the number of nodes $|V|$. Usually we expect the number of edges $|E|$ to be much smaller than the number of all possible edges: N^2. Possible distributions include chemical molecules, control flow graphs, or synthetic preferential attachment graphs (Albert and Barabási, 2002).

Graph neural networks based on the message passing scheme (Gilmer et al., 2017) aggregate the features of nodes in the neighborhood $\mathcal{N}_1$ of node i along their edges with features $e_{j,i}$. The aggregate a_i^l is then used to update the node feature h_i^{l-1} in the next layer l.

$$a_i^l = \text{Aggregate}^l\left(\{\!\{(h_j^{l-1}, e_{j,i}) : j \in \mathcal{N}_1(i)\}\!\}\right), \quad h_i^l = \text{Update}^l\left(h_i^{l-1}, a_i^l\right) \tag{1}$$

Here $h_i^0 = v_i \in V$ and $\{\!\{\dots\}\!\}$ denotes a multiset.

3 Related Work

Researchers have already applied graph neural networks to the task of GED approximation. Bai et al. proposed in 2018 (published Bai et al., 2019) to apply the same network to both graphs and aggregate the resulting node embeddings with an attention layer into two global graph embeddings, which can be used to compute a distance. In addition to this fully differentiable and therefore trainable part, they compute a histogram of the pairwise distances between node embeddings to incorporate some of the local node level information. However, the histogram method is not differentiable and it is unclear if the model can properly exploit this information. Moreover, this process reduces any graph to fixed-size embeddings and is unlikely to scale to larger graphs without hand-tuning the embedding size. Despite these flaws the authors show that their method significantly outperforms classical methods while being orders of magnitude faster at inference.

Later Bai et al. (2018) proposed to apply a standard 2-D Convolutional Neural Networks (CNN) to the matrix of pairwise distances, which replaces the histogram. As the authors point out, the main problems with this approach are permutation invariance and spatial locality. Firstly, a CNN depends on the ordering of the nodes in the graph representation. Therefore, the model is not invariant to permutations. Secondly, CNN's introduce a strong structural prior by assuming that nearby points are strongly related while further apart points are not. This assumption increases the importance of node order further. The authors decided to use breadth-first-search node-ordering (You et al., 2018). They claim that nearby nodes are sorted close to each other. However, the ordering cannot be guaranteed to be unique. It is not yet known if there exists a polynomial algorithm to find a canonical ordering (Niepert et al., 2016) and therefore the predictions can depend on the representation of the graphs. Yet, the authors show an insightful connection between convolutional kernels and the optimal assignment problem and show strong empirical results compared to classical algorithms.

Independently, Riba et al. (2018) proposed a similar model to estimate graph distances. They apply the same message passing network to both graphs generating sets A and B of node embeddings (one for each graph). The distance between two graphs is then computed as:

$$d(G_1, G_2) = \frac{\sum_{a\in A} \inf_{b\in B} d(a,b) + \sum_{b\in B} \inf_{a\in A} d(a,b)}{N_1 + N_2} \tag{2}$$

This equation is derived as a soft variant of the Hausdorff distance, which is also known as the Chamfer distance in computer vision (Barrow et al., 1977). Instead of approximating the GED directly they validate their method on graph classification and keyword spotting datasets. Their empirical results are therefore difficult to compare to Bai et al. (2018; 2019). This method is interesting because it generates a complete matching between nodes and might scale to larger graphs without hand-tuning the embedding size. Local node structures are incorporated during the message passing, and global node correspondence is used for predicting the final distance.

Li et al. (2019) used a similar method, called Graph Matching Network (GMN), for graph classification and detection of vulnerable binaries. They search for vulnerable binaries by comparing the control flow graphs of functions across different compilers. The GMN also employs message passing layers to both graphs. However, they modify the message passing scheme to allow information of one graph to flow into the node updates of the other graph. This process is fully differentiable and makes use of local node level and global graph level information but still suffers from the problem of reducing the entire graph to a single embedding vector which is expected to scale unfavorably to larger graphs. Moreover, computing the cross-graph matching vector costs $\mathcal{O}(N_1N_2)$. While all previously described models have technically the same complexity ($\mathcal{O}(N_1N_2 + |E_1| + |E_2|)$), the GMN computes this cross-graph matching vector for each layer.

Fey et al. (2020) apply message passing networks to graph matching. In particular they test their method on keypoint matching of objects between pictures and cross-lingual knowledge graph alignment. Although this task is different from those mentioned previously, their model is extremely similar to the one of Riba et al. (2018). They also apply the same network to both graphs and compute a matching. Instead of the Chamfer distance Fey et al. (2020) use the Sinkhorn normalization (Cuturi, 2013) on the matrix of pairwise distances. The Sinkhorn normalization returns a doubly stochastic matrix, which is interpreted as a soft correspondence between nodes. One could immediately find a matching by taking the maximum likelihood estimate, but the authors then apply a newly proposed consensus step. They generate an injective node coloring, or random node embeddings in practice, for one graph and use the soft correspondence matrix to map these to the other graph. Using the generated node embeddings, they apply another message passing network to both graphs, yielding, after normalization, a new soft correspondence matrix. This step is repeated multiple times and the final answer is found from the last soft correspondence matrix. Unfortunately, the authors report that in practice they had to use the softmax operator on each row, instead of Sinkhorn normalization due to lacking gradient information. However, when using softmax the model is not necessarily symmetric anymore, i.e. $M(G_1, G_2) \neq M(G_2, G_1)$.

4 Learning Distances via Matching

A metric is defined by four properties: non-negativity, symmetry, the identity of indiscernibles and the triangle equation. A model for learning metrics should be designed to enforce as many of these properties as possible. However, guaranteeing the identity of indiscernibles implies solving the graph isomorphism problem, which is not possible with standard GNNs (Xu et al., 2019b). And the triangle inequality seems extremely challenging to enforce on a powerful neural network model. Therefore, we will focus on symmetry and non-negativity. Moreover, the model should be invariant to the representation of the graph, i.e. the order of nodes in memory.

While the model should ideally be able to learn any metric on a set of graphs, we will focus on the ubiquitous GED for the derivation. The GED between two graphs is defined analog to the String Edit Distance.

$$\mathrm{GED}(G_1, G_2) = \min_{(e_1,\ldots,e_k)\in\mathcal{P}(g_1,g_2)} \sum_{i=1}^{k} c(e_i) \tag{3}$$

where e_i are edit operations (edge/node addition, removal, and substitution), $c(e)$ is the cost of an edit operation, and $\mathcal{P}(G_1, G_2)$ is the set of all possible edit paths that transform G_1 into a graph isomorphic to G_2.

A simple model to predict the GED might be to embed both graphs into a common vector space and compute the distance between these two vectors. One can use message passing networks to obtain permutation equivariant node embeddings. They can then be aggregated into a single graph embedding and, finally, any distance between two vectors (cosine, p-norm, etc.) will ensure non-negative and symmetric predictions. The main problem with this approach is that the embedding size becomes an information bottleneck for larger graphs. Working with varying sizes of graphs becomes impractical because increasing or decreasing the embedding size requires the entire model to be trained from scratch.

To solve this problem we propose[2] to match the nodes of both graphs, similar to classical GED algorithms (Riesen and Bunke, 2009; Fankhauser et al., 2011). These algorithms are based on bipartite graph matching. They set up a cost Matrix $C \in \mathbb{R}^{N \times N}$, where $N = N_1 + N_2$ and solve the following assignment problem:

$$C = \left[\begin{array}{ccc|ccc} C_{1,1} & \cdots & C_{1,N_2} & C_{1,\epsilon} & \cdots & \infty \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ C_{N_1,1} & \cdots & C_{N_1,N_2} & \infty & \cdots & C_{N_1,\epsilon} \\ \hline C_{\epsilon,1} & \cdots & \infty & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \infty & \cdots & C_{\epsilon,N_2} & 0 & \cdots & 0 \end{array}\right] \qquad \begin{gathered} d(C) = \min \langle M, C \rangle_{\mathrm{F}} \\ \text{subject to} \\ \sum_{i=1}^{N} M_{ij} = 1 \quad \forall j \in \{1 \dots N\} \\ \sum_{j=1}^{N} M_{ij} = 1 \quad \forall i \in \{1 \dots N\} \\ M_{ij} \in \{0, 1\} \end{gathered} \tag{4}$$

Here $\langle \cdot, \cdot \rangle_{\mathrm{F}}$ denotes the Frobenius inner product. $C_{i,j}$ is the cost for replacing node i with node j, $C_{\epsilon,j}$ the cost for inserting node j, and $C_{i,\epsilon}$ the cost for deleting node i. These costs depend on the direct neighborhood of the node, and can even be extended with random walks around it, but they still lack accuracy on graphs with complex global structures (Riesen and Bunke, 2009). The assignment problem can be solved exactly by algorithms such as the Hungarian method, also known as the Kuhn-Munkres algorithm (Kuhn, 1955) or the VJ algorithm (Jonker and Volgenant, 1987).

To allow the model to also learn other metrics on graphs we use a message passing network to learn the entries of the cost matrix C. The network generates node embeddings that contain information about their local neighborhood. We then compute pairwise distances between nodes of both graphs yielding a trainable equivalent to the upper left bock of the cost matrix. The corresponding values for insertion and deletion can also be learned by taking some norm of the embeddings. Analogous to the classical approach we compute the distance as $d = \langle M^*, C \rangle_{\mathrm{F}}$.

A matching M^* can be found by:

- **Nearest Neighbor.** Taking the sum of the row-wise minima (and, for the sake of symmetry, the sum of the column-wise minima) yields what is known in computer vision as the **Chamfer Distance**. Therefore, one can compute the Chamfer Distance using the max operator, without an explicit matching. An explicit matching, to use in the Frobenius inner product above, can be found with the arg max operator. Either way the process runs in $\mathcal{O}(N^2)$.
- **Optimal Assignment.** Applying the Kuhn-Munkres algorithm results in an optimal assignment, $M_{ij} \in \{0, 1\}$, in roughly $\mathcal{O}(N^3)$ operations.
- **Optimal Transport.** The optimal transport problem is a continuous relaxation of the optimal assignment problem, where $M_{ij} \in [0, 1]$. In this setting usually called the **Earth Mover's Distance**, it can be found in roughly $\mathcal{O}(N^3 \log(N))$ operations.
- **Regularized Optimal Transport.** Cuturi (2013) introduced an algorithm for solving an entropy regularized version of the optimal transport problem minimizing $\langle M, C \rangle_{\mathrm{F}} + \lambda \mathrm{H}(M)$. One simply computes the kernel matrix $K := e^{-\frac{C}{\lambda}}$ and applies Sinkhorn's iterative matrix

[2] Riba et al. (2018) already used an implicit matching for predicting the GED with neural networks but we improve on their work by utilizing a classical cost matrix and modern matching tools.

scaling. This is called the **Sinkhorn Distance** and can be computed extremely fast in practice with a running time of $\mathcal{O}(N^2)$.

While the max operation is discrete, in deep learning frameworks its gradient is implemented as the piecewise function to which the gradient of the softmax converges as base $b \to \infty$. That allows for direct backpropagation through the Chamfer distance. Also the Sinkhorn scaling and therefore the entire computation of the Sinkhorn distance is differentiable. However, backpropagating through the iterative scaling process is not efficient, and for the other methods we cannot compute proper gradients due to discrete operations. Fortunately, we can apply Danskin's theorem (Danskin, 1967) to calculate analytic gradients for the optimization problems above.

Theorem 1. *If $\phi(x, z)$ is a continuous function of two arguments, $\phi : \mathbb{R}^n \times Z \to \mathbb{R}$ where $Z \subset \mathbb{R}^m$ is a compact set, and $\phi(x, z)$ is convex in x for every $z \in Z$. Then the derivate of the function $f(x) = \max_{z \in Z} \phi(x, z)$ is:*

$$Df(x) = \phi'(x, z^*), \textit{ where } z^*(x) = \underset{z \in Z}{\arg\max}(\phi(x, z)) \tag{5}$$

Since $d(C, M)$ is affine, $-d(M, C)$ is also affine and convex. Moreover, the set of all possible matchings M is compact. We can apply Danskin's theorem which yields that the derivate of $d(C) = \min_M \langle M, C \rangle_{\mathrm{F}}$ is the derivate of the Frobenius inner product between C and the optimal matching M^*, which we have computed already for the forward pass. And the final gradient is simply the matrix M^* itself. For the discrete assignment of the Kuhn-Munkres algorithm this gradient is a piecewise function similar to the gradient of the max operator. Armed with the analytic gradient we can quickly compute gradient updates, and learn in an end-to-end fashion.

Preliminary experiments indicated that the Sinkhorn distance is not only similarly fast to compute as the Chamfer distance, but also yields better results than the other much slower methods. The regularization results in softer matchings compared to hard assignments or the still very peaky matchings of the earth mover's distance. We believe these soft matchings propagate more gradient information to the message passing network allowing it to thoroughly adapt to the given distribution of graphs. Note that by adjusting the regularization parameter λ we can control how soft the matchings are.

The final objective function depends on the specific task. For learning the GED we simply use the mean squared error between prediction and ground truth values. More specialized objectives such as the contrastive loss proposed by Riba et al. (2018) for keyword spotting, or pairwise margin loss, or triplet loss proposed by Li et al. (2019) for detecting fraudulent binaries can also be used with our model. Additionally to the experiments in this report we applied our model to a similar dataset of control flow graphs. Details can be found in Appendix B.

In response to Fey et al. (2020), who used a similar model for graph matching, we decided to evaluate our model on a graph matching task as well. Another graph matching dataset we investigated shortly can be found in Appendix C. Because graph matching uses the matching matrix directly in the objective function, we are not able to apply Danskin's theorem in this setting. Therefore, we do have to backpropagate through the Sinkhorn iterations. However, we found that using 50 iterations is sufficient to get a good matching and computing the gradient updates is reasonably fast.

5 The Graph Distance Network

We propose the Graph Distance Network (GDN), which uses a siamese network structure with multiple shared message passing layers, the previously derived learnable cost matrix, and the Sinkhorn distance with analytic gradients.

Let d_h be an arbitrary embedding size, d_{node} the number of node features of the input graphs, d_{edge} the number of edge types, $W_l \in \mathbb{R}^{d_h \times d_h}$, $b_l \in \mathbb{R}^{d_h}$ a weight matrix and bias, $h_i^l \in \mathbb{R}^{d_h}$ the embedding of node i at layer l for $l > 0$, $\sigma(\cdot)$ an elementwise non-linearity, and $E \in \mathbb{R}^{d_{\text{edge}} \times d_h \times d_h}$ a weight tensor ($h_i^0 = v_i \in \mathbb{R}^{d_{\text{node}}}$, $W_0 \in \mathbb{R}^{d_{\text{node}} \times d_h}$, $b_0 \in \mathbb{R}^{d_{\text{node}}}$). We then use the following message passing function:

$$h_i^l = \sigma(W_l h_i^{l-1} + b_l) + \sum_{j \to i} \sigma(W_l h_j^{l-1} + b_l) E_{e_{j \to i}} \tag{6}$$

where $E_{e_{j\to i}} \in \mathbb{R}^{d_h \times d_h}$ is the weight tensor indexed by the discrete feature of the edge connecting node j with node i. As activation function $\sigma(\cdot)$ we use the leaky-relu non-linearity.

To compute the entries of the cost matrix we concatenate all node embeddings and take the norm of pairwise distances across both graphs, or the embeddings scaled by a trainable $\alpha \in \mathbb{R}^{d_{\text{final}}}$:

$$\begin{gathered} h_i^{\text{final}} = [h_i^L; \ldots; h_i^1; \sigma(W_1 h_i^0 + b_0)] \in \mathbb{R}^{d_{\text{final}}} \\ C_{i,j} = \left\| h_i^{\text{final}} - h_j^{\text{final}} \right\| \quad C_{i,\epsilon} = \left\| \alpha h_i^{\text{final}} \right\| \quad C_{\epsilon,j} = \left\| \alpha h_j^{\text{final}} \right\| \quad i \in \{1 \ldots N_1\}, j \in \{1 \ldots N_2\} \end{gathered} \tag{7}$$

In the following experiment we tested three different norms: the euclidean norm ($p = 2$), the manhattan norm ($p = 1$), and a simple two layer perceptron (MLP). The first layer of this MLP does not change the embedding size, then we apply a relu non-linearity. The second layer reduces the embedding size to a single number to which we apply the softplus non-linearity, $\ln(1 + e^x)$.

As for the matrix C we tested two variants. The first one we call BP-Distance on account of bipartite graph matching. Due to implementation advantages we decided to pad the pairwise distances with rows and columns of $C_{i,\epsilon}, C_{\epsilon,j}$ respectively, instead of the ∞-values typically used.

$$C_{\text{BP}} = \left[\begin{array}{ccc|ccc} C_{1,1} & \cdots & C_{1,N_2} & C_{1,\epsilon} & \cdots & C_{1,\epsilon} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ C_{N_1,1} & \cdots & C_{N_1,N_2} & C_{N_1,\epsilon} & \cdots & C_{N_1,\epsilon} \\ \hline C_{\epsilon,1} & \cdots & C_{\epsilon,N_2} & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ C_{\epsilon,1} & \cdots & C_{\epsilon,N_2} & 0 & \cdots & 0 \end{array} \right] \in \mathbb{R}^{(N_1+N_2)\times(N_1+N_2)} \tag{8}$$

This does lead to different results after Sinkhorn normalization due to the entropy regularization, but the network can easily adapt by increasing $\|\alpha\|$. Therefore, this modification should not affect our trainable model. The second variant of the cost matrix is the smallest square sub-matrix of the one above that contains all pairwise distances, such that $C_{\text{no-BP}} \in \mathbb{R}^{\max(N_1,N_2)\times\max(N_1,N_2)}$.

The final distance d is then computed and trained via the mean squared error between target distance and prediction:

$$\begin{gathered} M = \text{sinkhorn}\left(e^{-\frac{C}{\lambda}}\right) \\ d = \langle M, C \rangle_{\text{F}} \\ \text{loss} = (d - d_{\text{target}})^2 \end{gathered} \tag{9}$$

Implementation. Our GNN is implemented with PyTorch (Paszke et al., 2019), PyTorch Scatter, and PyTorch-geometric (Fey and Lenssen, 2019) and can process sparse mini-batches with parallel GPU acceleration. We have two implementations for the matching. The first version pads matrices to form batches and can be used in cases where all graphs have a similar number of nodes. The second version uses sparse matrices in COOrdinate format, which is less efficient with evenly sized graphs but can be crucial for parallelizing workloads with skewed graph size distributions.

6 Experiment Setup

For our first experiment we use synthetic preferential attachment graphs (Albert and Barabási, 2002) and real-world chemical molecules of the AIDS dataset[3]. Both datasets have discrete node and edge features, and graphs are limited to at most 30 nodes. We provide some basic dataset statists in the Appendix A.

We compare our results to the implementations of Riba et al. (2018), Bai et al. (2019), and Li et al. (2019) in Table 1. All methods are described in the Related Work Section. We train for 500 epochs and report the best error on the validation set and the corresponding test error. We ran 3 trials of every setting and searched over the cross product of 2 different batch sizes (depending on the implementation), 32 and 64-dimensional node embeddings, 1 and 3 layers of

[3] `https://wiki.nci.nih.gov/display/NCIDTPdata/AIDS+Antiviral+Screen+Data`

message passing, 4 learning rates, and 4 degrees of regularization. Implementation details can be found in the respective GitLab repositories[4].

For our model, the graph distance network (GDN), we fixed the batch size to 1024, the embedding size to 32, and the number of layers to 3, and added 3 choices for the Sinkhorn regularization constant λ. Moreover, we include an ablation study over the choice of the cost matrix and different norms in Table 2.

7 Results

Table 1 shows that our method clearly outperforms all baselines. Best results on both datasets were found with the no-BP cost matrix, MLP-norm, and small weight decay for regularization. Interestingly, the only difference between both configurations is the Sinkhorn regularization constant λ, which is smaller for the synthetic graphs. However, somewhat concerning is the gap between the validation and test performance on the real-world dataset, which also appears with the model of Bai et al. (2019). This phenomenon will be discussed in the ablation study.

	Pref-Attachment		AIDS	
	Val	Test	Val	Test
Riba et al. (2018)	12.2 ± 0.2	12.1 ± 0.6	15.5 ± 0.3	15.6 ± 0.3
Bai et al. (2019)	7.7 ± 1.0	9.6 ± 2.5	4.2 ± 0.3	8.7 ± 0.1
Li et al. (2019)	5.5 ± 0.1	7.8 ± 0.3	10.6 ± 0.3	11.7 ± 0.9
GDN	4.2 ± 0.1	$\mathbf{4.5 \pm 0.2}$	3.3 ± 0.04	$\mathbf{6.2 \pm 1.0}$

Table 1: RMSE to ground truth GED with standard deviation across 3 runs.

The ablation study in Table 2 indicates that:

- **The Euclidean norm is superior to the manhattan norm, and a trainable MLP can improve results even further.**
- **There comes no immediate benefit from the full-size cost matrix.** In all but one case the no-BP cost matrix is at least as good as the BP variant in validation and test performance. Even though intuitively it feels convenient to have the option to explicitly add or remove nodes, it may be the case that we can get the same result by substituting one node for another in the case of the GED. However, using the no-BP matrix could certainly lead to problems when attempting to learn "sparse matchings" where only a few nodes have a match in the other graph. Moreover, it is not surprising that the results of both variants are very similar because the Sinkhorn normalization will place a lot of weight onto the 0 entries in the cost matrix. It does that to a point where summing over the part of the rows (or columns depending on which graph has more nodes) which had 0 entries before normalization yields values close to 1. Therefore, the values that contribute to the distance come mostly from entries which also show up in our no-BP variant matrix.
- **The AIDS validation set does not properly represent the entire distribution.** The difference in validation and test performance on the synthetic graphs can be explained by the variation between consecutive epochs. Since we stop on the best validation result it is no surprise that the test result at that exact point is slightly worse on average. In contrast, the massive gap in performance, and the high standard deviation on the AIDS test set indicate a shift of distributions between the validation and the test set. Some runs generalized well from the training data to the validation set, but they consistently performed worse on the test set across all epochs. Interestingly, not all runs behaved that way. We also collected many runs where the test performance is close to validation performance[5]. Apparently the manhattan norm is more susceptible to overfit in this manner than the MLP norm.

[4]Riba: `https://gitlab.lrz.de/gdn/siamese_ged`, Bai `https://gitlab.lrz.de/gdn/simgnn`, Li `https://gitlab.lrz.de/gdn/gmn/tree/gr_report`

[5]Often these runs had higher Sinkhorn regularization but we did not run further experiments to verify that impression.

		Pref-Attachment		AIDS	
BP	Norm	Val	Test	Val	Test
yes	$p=1$	5.2 ± 0.1	6.4 ± 0.7	3.9 ± 0.1	17.1 ± 10.9
	$p=2$	4.8 ± 0.1	5.7 ± 0.2	3.9 ± 0.2	8.6 ± 2.6
	MLP	4.5 ± 0.1	$\mathbf{4.5 \pm 0.3}$	3.5 ± 0.03	$\mathbf{4.7 \pm 0.1}$
no	$p=1$	4.9 ± 0.2	5.8 ± 0.2	3.9 ± 0.3	16.2 ± 6.0
	$p=2$	4.8 ± 0.1	5.7 ± 0.6	3.7 ± 0.2	6.8 ± 3.4
	MLP	4.2 ± 0.1	$\mathbf{4.5 \pm 0.2}$	3.3 ± 0.04	6.2 ± 1.0

Table 2: Ablation Study: RMSE with standard deviation across 3 runs.

8 Graph Matching

As described in 3 Related Work, Fey et al. (2020) have recently proposed a similar model, the Deep Graph Consensus (DGC), to predict soft matchings between nodes. They train on the negative log-likelihood between the soft matching and some ground truth matching (following Wang et al., 2019). The final performance is measured by how often the row-wise maximum in the soft matching yields the correct match (Hits@1).

To show that it is advantageous to apply the Sinkhorn normalization to the kernel matrix instead of the cost matrix directly, we also run the keypoint matching experiment of Fey et al. (2020) on the PASCALVOC dataset (Everingham et al., 2010) with Berkeley annotations (Bourdev and Malik, 2009).

To align our model as much as possible with their work we use the same SplineCNN (Fey et al., 2018) with the same embedding size and number of layers to generate node embeddings. We use $p = 2$ for our norm (to avoid using additional parameters), and the no-BP variant of the cost matrix because the graphs are built such that every node in the first (source) graph has a match in the second (target) graph. Therefore, the network would only learn to predict 0's for the additional values. This left us with only the learning rate and the regularization constant λ to tune. For numeric stabilization we scaled the kernel matrix by the sum of all entries divided by the number of entries: $\text{reg}^l = \frac{\|C^l\|_1}{\max(N_1,N_2)^2}$. We also implemented a consensus step. For a detailed comparison of the two models see Appendix D.

In Table 3 we report results on the test set of the model with the best validation performance across 18 runs (3 trials, 2 learning rates, and 3 regularization constants), and compare these to the numbers reported by Fey et al. (2020). The table shows that using the kernel matrix and the inherent ability to tune the regularization constant improves the base model without consensus. The 3 percent points gain constitute over half of the gains made by the original consensus step. Unfortunately, applying the consensus step to our model was not successful. In fact, by adding our consensus step the performance drops slightly. We believe that the consensus step should be orthogonal to the gains due to the kernel matrix. However, so far, we were not able to improve upon the poor performance even after multiple tweaks to the cost matrix generation.

Method		Aero	Bike	Bird	Boat	Bottle	Bus	Car	Cat	Chair	Cow	Table	Dog	Horse	M-Bike	Person	Plant	Sheep	Sofa	Train	TV	Mean
DGC	L=0	42.1	57.5	49.6	59.4	83.8	84.0	78.4	67.5	37.3	60.4	85.0	58.0	66.0	54.1	52.6	93.9	60.2	85.6	87.8	82.5	67.3
GDN	L=0	45.3	63.8	55.5	72.6	88.9	88.5	79.5	69.1	36.8	58.5	82.8	59.2	70.1	59.8	53.4	96.3	63.7	92.5	91.7	85.0	70.5
DGC	L=10	45.5	67.6	56.5	66.8	86.9	85.2	84.2	73.0	43.6	66.0	92.3	64.0	79.8	56.6	56.1	95.4	64.4	95.0	91.3	86.3	72.8
GDN	L=10	40.2	59.6	50.1	65.0	87.8	86.7	82.3	69.6	37.4	59.0	93.1	56.7	73.1	52.9	58.5	96.3	62.1	90.0	93.4	87.3	70.0

Table 3: Hits@1 (%) on the PASCALVOC dataset with Berkeley keypoint annotations

9 Conclusion

We proposed to use a cost matrix, motived by bipartite graph matching, and an explicit soft matching between nodes to learn an accurate and scalable estimation of metrics on graphs. Our model has the same theoretical complexity $\mathcal{O}(N_1 N_2)$ as previous neural network methods designed for this task. In practice the model runs quickly because our implementation can efficiently handle large mini-batches of arbitrarily sized graphs making it well suited for modern GPU setups.

We showed empirically that the model improves upon state-of-the-art methods in predicting the ubiquitous graph edit distance.

Closely following the setup of Fey et al. (2020), we applied our model to graph matching. In particular, motivated by the Sinkhorn distance, we proposed to apply Sinkhorn scaling to the kernel of the cost matrix. We showed empirically that with little tuning of the regularization constant this model improves upon the best results of Fey et al. (2020) without the consensus step.

10 Outlook

The first problem that should be tackled is the gap between validation and test results on the AIDS dataset. Increasing the number of graphs could potentially solve the issue already. However, to verify any improvement we should also consider working with multiple completely distinct validation sets at once.

Also the underperforming consensus step should be improved in the future. We can produce a solid initial matching, and simply applying the same "softmax-no-norms" consensus of Fey et al. (2020) should yield at least as good results as they report. Therefore, exactly replicating their model might be a good start and allow us to then investigate which modifications can further improve these results.

After that we might want to look back at the four defining properties of metrics:

- $d(x, y) \geq 0$ non-negativity
- $d(x, y) = 0 \Leftrightarrow x = y$ identity of indiscernibles
- $d(x, y) = d(y, x)$ symmetry
- $d(x, y) \leq d(x, z) + d(z, y)$ subadditivity or triangle inequality

We already ensured non-negativity and symmetry, but it would certainly be interesting to investigate if we can ensure the triangle inequality. Perhaps that could lead to a small modification of the model that induces a strong and useful prior. Also investigating the identity of indiscernibles could lead to interesting results. If it was possible to ensure this property, it would solve the graph isomorphism problem, and finally answer the question of whether the problem is NP-complete or not. However, doing so will require the use of more powerful embedding networks because a standard GNN cannot distinguish between certain graphs[6] and will, therefore, yield the same set of embeddings in such a case. Maybe easier to handle and therefore more promising is translational invariance, $d(x, y) = d(x + a, y + a)$, which might be a valid assumptions in many cases. Graph sums can be defined by adding the adjacency matrices of equal-sized graphs (Beineke et al., 2004) or, in the context of the graph edit distance, maybe by adding a node that is connected to all other nodes. If it is not possible to derive enforcing model modifications for translational invariance or the triangle inequality then one could still try using the properties for data augmentation, which could certainly lead to better generalization.

A different direction for further research is to speed up the current model. It seems inevitable that the whole process is quadratic in the number of nodes due to the use of the cost matrix $C \in \mathbb{R}^{N \times N}$. However, we might be able to achieve linear run time by using the Nyström method or multiscale matching. Altschuler et al. (2019) show that using the Nyström method and Sinkhorn scaling one can accurately approximate the Sinkhorn distance without ever computing the full cost matrix. The Nyström method is based on a set of landmarks that are used to speed up the matching. Similarly multiscale matching uses clusters of nodes that are then matched with each other (Schmitzer, 2016). Ideally one would combine both approaches to increase the accuracy since landmarks and clusters each have their merits and demerits. Using landmarks one can vastly overestimate small distances, and multiscale matching never computes distances that exceed some threshold but approximates them with infinity.

To show off the advantage of our full BP cost matrix it would be interesting to investigate "sparse matchings". Fey et al. (2020) use only pairs of graphs where every node in the source graph finds a match in the target graph. We could simply add the other pairs to the training set, but validate on both groups individually. There should be a good chance that we significantly outperform any previous methods on this specialized task.

[6]For example, a GNN cannot distinguish between a hexagon and two disconnected triangles.

References

Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74:47–97, Jan 2002. doi: 10.1103/RevModPhys.74.47.

Jason Altschuler, Francis Bach, Alessandro Rudi, and Jonathan Niles-Weed. Massively scalable sinkhorn distances via the nyström method. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 4429–4439, 2019.

Yunsheng Bai, Hao Ding, Yizhou Sun, and Wei Wang. Convolutional set matching for graph similarity. *CoRR*, abs/1810.10866, 2018. URL `http://arxiv.org/abs/1810.10866`.

Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. Simgnn: A neural network approach to fast graph similarity computation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, WSDM 2019, Melbourne, VIC, Australia, February 11-15, 2019*, pages 384–392, 2019. doi: 10.1145/3289600.3290967.

Harry G. Barrow, Jay M. Tenenbaum, Robert C. Bolles, and Helen C. Wolf. Parametric correspondence and chamfer matching: Two new techniques for image matching. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence. Cambridge, MA, USA, August 22-25, 1977*, pages 659–663, 1977.

Lowell W. Beineke, Robin J. Wilson, and Peter J. Cameron. *Topics in Algebraic Graph Theory.* New York: Cambridge University Press, 2004.

Kristian Birchall, Valerie J. Gillet, Gavin Harper, and Stephen D. Pickett. Training similarity measures for specific activities: Application to reduced graphs. In *Journal of Chemical Information and Modeling, J. Chem. Inf. Model. 2006*, pages 577–586, 2006.

Lubomir D. Bourdev and Jitendra Malik. Poselets: Body part detectors trained using 3d human pose annotations. In *IEEE 12th International Conference on Computer Vision, ICCV 2009, Kyoto, Japan, September 27 - October 4, 2009*, pages 1365–1372, 2009. doi: 10.1109/ICCV.2009.5459303.

Horst Bunke and Kim Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19(3-4):255–259, 1998. doi: 10.1016/S0167-8655(97)00179-7.

Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 2292–2300, 2013.

J.M. Danskin. *The theory of Max-Min and its application to weapons allocation problems.* Econometrics and operations research. Springer-Verlag, 1967.

Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman. The pascal visual object classes (VOC) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010. doi: 10.1007/s11263-009-0275-4.

Stefan Fankhauser, Kaspar Riesen, and Horst Bunke. Speeding up graph edit distance computation through fast bipartite matching. In *Graph-Based Representations in Pattern Recognition - 8th IAPR-TC-15 International Workshop, GbRPR 2011, Münster, Germany, May 18-20, 2011. Proceedings*, pages 102–111, 2011. doi: 10.1007/978-3-642-20844-7_11.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 869–877, 2018. doi: 10.1109/CVPR.2018.00097.

Matthias Fey, Jan Eric Lenssen, Christopher Morris, Jonathan Masci, and Nils M. Kriege. Deep graph matching consensus. *CoRR*, abs/2001.09621, 2020. URL `https://arxiv.org/abs/2001.09621`.

Andreas Fischer, Ching Y. Suen, Volkmar Frinken, Kaspar Riesen, and Horst Bunke. Approximation of graph edit distance based on hausdorff matching. *Pattern Recognition*, 48(2):331–343, 2015. doi: 10.1016/j.patcog.2014.07.015.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 1263–1272, 2017.

Roy Jonker and A. Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38(4):325–340, 1987. doi: 10.1007/BF02278710.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. doi: 10.1002/nav.3800020109.

Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 3835–3845, 2019.

Andreas Loukas. What graph neural networks cannot learn: depth vs width. *CoRR*, abs/1907.03199, 2019. URL `http://arxiv.org/abs/1907.03199`.

Michel Neuhaus and Horst Bunke. A graph matching based approach to fingerprint classification using directional variance. In *Audio- and Video-Based Biometric Person Authentication AVBPA 2005*, pages 191–200, 2005.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 2014–2023, 2016.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 8024–8035, 2019.

Pau Riba, Andreas Fischer, Josep Lladós, and Alicia Fornés. Learning graph distances with message passing neural networks. In *24th International Conference on Pattern Recognition, ICPR 2018, Beijing, China, August 20-24, 2018*, pages 2239–2244, 2018. doi: 10.1109/ICPR.2018.8545310.

Kaspar Riesen and Horst Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image Vision Comput.*, 27(7):950–959, 2009. doi: 10.1016/j.imavis.2008.04.004.

Bernhard Schmitzer. A sparse multiscale algorithm for dense optimal transport. *Journal of Mathematical Imaging and Vision*, 56(2):238–259, 2016. doi: 10.1007/s10851-016-0653-9.

Rohit Singh, Jinbo Xu, and Bonnie Berger. Global alignment of multiple protein interaction networks with application to functional orthology detection. *Proc. Natl. Acad. Sci. U.S.A.*, 105 (35):12763–12768, 2008. doi: 10.1073/pnas.0806627105.

Mario Vento and P. Foggia. Graph matching techniques for computer vision. *Graph-Based Methods in Computer Vision: Developments and Applications*, 1:1–41, 01 2012. doi: 10.4018/978-1-4666-1891-6.ch001.

Runzhong Wang, Junchi Yan, and Xiaokang Yang. Learning combinatorial embedding networks for deep graph matching. *CoRR*, abs/1904.00597, 2019. URL `http://arxiv.org/abs/1904.00597`.

Bing Xiao, Xinbo Gao, Dacheng Tao, and Xuelong Li. Hmm-based graph edit distance for image indexing. *International Journal of Imaging Systems and Technology*, 18:209 – 218, 08 2008. doi: 10.1002/ima.20146.

Hongteng Xu, Dixin Luo, and Lawrence Carin. Scalable gromov-wasserstein learning for graph partitioning and matching. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 3046–3056, 2019a.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019b.

Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 5694–5703, 2018.

Jiawei Zhang, Qianyi Zhan, and Philip S. Yu. Concurrent alignment of multiple anonymized social networks with generic stable matching. In *Theoretical Information Reuse and Integration*, pages 173–196. 2016. doi: 10.1007/978-3-319-31311-5_8.

Zhen Zhang and Wee Sun Lee. Deep graphical feature learning for the feature matching problem. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 5086–5095, 2019. doi: 10.1109/ICCV.2019.00519.

Appendices

In this report I summarize the work I did during my guided research. The main part of the report focuses on findings that are relevant to a potential paper submission, but here in the appendix I will add some information about the datasets we used, and some we did not use (for completeness).

A Graph Edit Distance Datasets

Dataset	Node Feat	Edge Feat	#Train	#Val/Test	Max Nodes	Avg Nodes	Avg Edges
Pref-Att	6	4	144	48	30	20.6	75.4
AIDS	53	4	144	48	30	20.6	44.6

Table 4: Datasets

Note #Train and #Val/Test are the number of graphs. We train and validate/test on all possible pairs of graphs, except those where a graph is paired with itself.

B Control Flow Dataset

Li et al. (2019) used control flow graphs to evaluate their model. They used metric learning (margin-loss with pairs, and triplets) to recognize a function compiled from different compilers as similar, and different functions as not similar. They didn't share their dataset of FFmpeg functions with us, but referred us to a GitHub repository, which I then forked (`https://github.com/johannespitz/functionsimsearch/tree/master`) and used to generate the UnRAR datasets, which (at the time of writing) can be found at '/nfs/students/pitz/cfgs' on the file server. To generate the graphcollections I used the notebook 'cfgs.ipynb' in the graph-distance repository. Note: use the .pickle version because it is much faster than .npz.

Currently we have a few different options for train/val/test splits. The simplest split is to use train_all/val/test, which is an 80/10/10 split in terms of functions (not graphs). The attraction list contains all pairs of graphs that are from the same function, and the repulsion list contains as many negative samples. The train_all split is then further split in: Option1, train_across/val_across/test_acorss. Here out of the n graphs of any function we pick one to be the test graph, and all pairs between it and the remaining n-1 graphs are placed in the attraction list. Of the n-1 we then draw a validation graph and again all pairs between it and the remaining n-2 graphs are placed in the attraction list. The attraction list of the train_across split contains all possible pairs of the remaining n-2 graphs. Option2, train12/val1/val2/test1/test2. Here we split train_all as chunk/val2/test2 (70/15/15) in terms of graphs (not functions), while ensuring that at least one graph of each function is contained the chunk. We then generate all positive pairs, but for the chunk we split those again into train12/val1/test1 (80/10/10) ensuring that every graph is in train12 at least once. All repulsion lists for both options contain as many negative samples as the corresponding attraction list. Triplets are only available with the simple train_all/val/test (80/10/10) split.

Our model can be trained with pairs or triplets as it is. I did not include any results with the dataset in this report because the baselines cannot be trained on it (without putting in additional work).

C Yeast Dataset

The Yeast Dataset (`https://www3.nd.edu/~cone/MAGNA++/`), used by Xu et al. (2019a), contains the protein-protein interaction (PPI) network of yeast and noisy versions thereof. The base network contains 1,004 proteins and 4,920 high-confidence interactions. The noisy versions contain additional low-confidence interactions.

The first obstacle with this dataset would have been the generation of train/val/test splits. Moreover, we could not have compared our results directly with Xu et al. (2019a) because they learn unsupervised. The second obstacle is that the dataset contains no node or edge features. That is mostly a problem because GNNs are particularly good at using node, and also edge features. Therefore, it is questionable if a model based on GNNs would perform impressive enough. In the end we decided not to use this dataset, also because we discovered the paper by Fey et al. (2020) who use plenty of graph matching datasets that fit our model much better.

D Consensus Step

The main difference between our model and Fey et al. (2020) is that they normalize the cost matrix C instead of kernel matrix $e^{-\frac{C}{\lambda}}$. Consequently, they report bad gradients with Sinkhorn normalization and use row-wise softmax instead. For their cost matrix, they only use the pairwise distances between nodes without padding with norms, which should not be a problem for tasks where each node in the first graph has a match in the second graph but could be problematic in other settings. In Table 5 we collected a complete list of the differences between both models:

	DGC	GDN
$C^0_{i,j}$	$h^0_i \cdot h^0_j$	$\begin{cases} \left\lVert h^0_i - h^0_j \right\rVert_2 & \text{if } 1 \leq i \leq N_1,\ 1 \leq j \leq N_2 \\ \left\lVert \alpha h^0_i \right\rVert_2 & \text{if } N_1 < i \leq N_2,\ 1 \leq j \leq N_2 \end{cases}$
$C^l_{i,j}$	$C^{l-1}_{i,j} + \text{MLP}(h^l_i - h^l_j)$	$\begin{cases} \beta^l C^0_{i,j} + (1-\beta^l)\text{MLP}(h^l_i - h^l_j) & \text{if } 1 \leq i \leq N_1,\ 1 \leq j \leq N_2 \\ \gamma^l \gamma^l C^{l-1}_{i,j} & \text{if } N_1 < i \leq N_2,\ 1 \leq j \leq N_2 \end{cases}$
M^l	$\text{softmax}(C^l)$	$\text{sinkhorn}\left(e^{-\frac{C^l * \text{reg}^l}{\lambda}}\right)$

Table 5: Differences of the two models

Note1: All superscripts indicate the layer of the consensus step. Note2: $\gamma^l \in \mathbb{R}$ is a trainable parameter initialized with 1. Note3: Fey et al. (2020) use only those pairs where every node in the source graph finds a match in the target graph (therefore: $N_1 < N_2$). And since we use the no-BP cost matrix we have $1 \leq i, j \leq N_2$