Graph node matching for edit distance

Aldo Moscatelli, Jason Piquenot, Maxime Bérar, Pierre Héroux, Sébastien Adam

Graph Node Matching for Edit Distance

Aldo Moscatelli*,[1], Jason Piquenot[1], Maxime Bérar[1], Pierre Héroux[1], and Sébastien Adam[1]

[1]LITIS Lab, University of Rouen Normandy, Avenue de l'université, Saint-Étienne-du-Rouvray, 76800, Normandie, France.

Abstract

Graphs are commonly used to model interactions between elements of a set, but computing the Graph Edit Distance between two graphs is an NP-complete problem that is particularly challenging for large graphs. To address this problem, we propose a supervised metric learning approach that combines Graph Neural Networks and optimal transport to learn an approximation of the GED in an end-to-end fashion. Our model consists of two siamese GNNs and a comparison block. Each graph pair's nodes are augmented by positional encoding and embedded by multiple Graph Isomorphism Network layers. The obtained embeddings are then compared through a Multi-Layer Perceptron and Linear Sum Assignement Problem solver applied on a node-wise Euclidean metric defined in the embedding space. We show that our approach achieves state-of-the-art results on benchmark datasets and outperforms other similar works in the domain. Our approach also provides explainability through the extraction of an edit path from one graph to another and guarantees metric properties conservation during training and inference.

1 Introduction

Many real-world situations can be described by interactions between entities within a structure. Typical examples include interactions between individuals in a social network, between atoms in a molecule, or between pages on a website. Graphs are a versatile and powerful formalism for encoding such interactions through a set of entities called nodes (or vertices) connected by edges (or arcs) that represent the interactions.

From a Pattern Recognition perspective, a key but computationally challenging task is to define a distance between a pair of graph instances. Applications involving such a distance include the task of retrieving a set of similar graphs

*Corresponding author: aldo.moscatelli@gmail.com

from a database given a user query, or classification/regression tasks on graphs using a distance-based machine learning algorithm (e.g. k-Nearest Neighbors).

The most commonly used measure of dissimilarity between graphs in the literature is the Graph Edit Distance (GED). In general, GED is defined as the minimum "amount of distortion" required to transform one graph into another using substitutions, insertions, and deletions of nodes/edges. More formally, the graph edit distance $d(.,.)$ is a function given by :

$$\begin{aligned} d \quad : \quad & \mathcal{G} \times \mathcal{G} \to \mathbb{R}^+ \\ & (G_1, G_2) \mapsto d(G_1, G_2) = \min_{(o_1,\ldots,o_k)\in\Gamma(G_1,G_2)} \sum_{i=1}^{k} c(o_i) \end{aligned}$$

where $\Gamma(G_1, G_2)$ is the set of all edit paths $o = (o_1, \ldots, o_k)$ that allow transforming G_1 to G_2. An elementary edit operation o_i is one of vertex substitution $(v_1 \to v_2)$, edge substitution $(e_1 \to e_2)$, vertex deletion $(v_1 \to \varepsilon)$, edge deletion $(e_1 \to \varepsilon)$, vertex insertion $(\varepsilon \to v_2)$ and edge insertion $(\varepsilon \to e_2)$. ε is a dummy vertex or edge used to model insertions or deletions. $c(.)$ is a function that associates a cost to each elementary edit operation o_i. These edit operations define an edit path, which can be useful in explaining the distance value from a practitioner's point of view. If each elementary operation satisfies the criteria of a distance (separability, symmetry, and triangular inequality), the GED defines a metric between graphs [1].

Despite these important properties, it is well known that computing a GED is an instance of the NP-hard Quadratic Assignment Problem (QAP)[2]. As a consequence, it is not possible to compute a GED for a pair of graphs whose size is greater than a few tens of vertices. To overcome this problem, many approximations of the GED have been proposed in the literature by transforming the initial problem into a relaxed one [3, 4] or by introducing heuristics based on deep learning into a combinatorial algorithm [5, 6]. These approaches speed up the computation at the cost of a loss of precision, as shown in [7, 8].

Very recently, the GED computation has been tackled using strategies based entirely on deep learning [9, 10, 11, 12, 13, 14, 15, 16, 17, 6, 18] to overcome the computational burden of algorithmic approaches. Such approaches aim to learn a statistical model from a dataset of graph pairs labeled with their GED. Once learned, the model is able to predict the distance for unseen graph pairs in the inference phase. Existing models are generally based on two main steps. The first takes advantage of recent advances in Graph Neural Networks (GNNs), which have been shown to be an efficient class of models for learning graph representations. Through a Siamese GNN architecture, this step embeds each graph of the pair into vector data, either at the node level or at the graph level. The second step infers graph similarity by comparing the obtained embeddings.

In this paper, we follow the same general architecture and present GNOME (Graph NOde Matching for Edit distance), a new Siamese architecture that targets the learning of a graph similarity function. Our proposal differs from existing ones by simultaneously reaching three main goals. The first is to address the necessary trade-off between the expressiveness of the model and its

efficiency. We achieve this objective through a linear-time GNN, taking as input the node features enriched with positional encodings to improve the model expressivity. The second objective is the explicability of the obtained distance, i.e. the ability to provide an edit path associated with the distance value. We achieve this goal by using a node-level dissimilarity measure provided by a Linear Sum Assignment Problem (LSAP) between node embeddings, rather than computing a metric between graph embeddings. The third objective is to ensure that the theoretical properties of a distance are preserved by the model. We prove that the GNOME output respects the properties of a distance and we show the impact on a query-answering downstream task.

The paper is organized as follows. Section 2 reviews recent learning-based approaches for graph similarity computation. Section 3 describes the proposed GNOME model, with an emphasis on the three objectives mentioned below. Finally, section 4 presents experimental results obtained on reference datasets, showing that GNOME equals or even outperforms existing models w.r.t. the aforementioned objectives.

2 Related work

As mentioned above, deep learning strategies for computing distances between graphs have recently been investigated. To the best of our knowledge, a pioneering paper under this paradigm is the one presenting the SimGNN architecture[16]. SimGNN computes graph similarity by combining two strategies. The first one relies on a Neural Tensor Network (NTN) that takes as input an embedding of each graph, obtained through siamese Graph Convolutional Network (GCN) layers [19] and an attention-based readout function. The output of the NTN is a similarity vector. The second strategy operates at the node level, by computing a histogram of the pairwise dot product similarities between GCN node embeddings. The outputs of each strategy, used alone or concatenated, are fed as input of an MLP to produce the similarity. SimGNN obtains very competitive results w.r.t. non-learning-based models and is frequently used as a comparison baseline. However, it does not provide any matching between the nodes (i.e. the explicability target) nor theoretical metric properties. Nevertheless, SimGNN and more particularly its Siamese-based architecture inspire many subsequent works that can be distinguished according to the embedding level used for matching.

Graph-level matching This category corresponds to the first strategy of SimGNN. It consists of using a siamese graph encoder that combines GNN layers and an invariant readout function to extract a graph embedding. The reference method for this technique is GREED [9] which embeds graphs into vectors through GIN layers. GREED provides good metrics properties conservation but has no explicability about the matching since it does not rely on the GED definition. MGMN [14] uses the same kind of encoding block but with a siamese GCN. It also computes a node-graph interaction with an attention mechanism

and Bi-LSTM layers in order to obtain a multi-level description of the graph pair. In the same spirit, GMN [13] improves the classical siamese network by adding a cross-graph matching vector in the message propagation. Such models are particularly efficient for retrieval tasks thanks to their metric properties that enable linear time query-answering.

Node-level matching As for the second strategy of SimGNN, another way to compute the similarity between two graphs is to perform matching at the node level. Such architectures are still based on a GNN encoder block but do not include readout functions. The encoder produces a set of node representations of the graph which can be seen as a distribution of the graph. Then, different strategies take place to infer the similarity between these distributions. For example, GraphSim [11] computes a pairwise matching node cost matrix with a dot product between all nodes' representation vectors obtained through a siamese GCN pipeline. Those cost matrices are processed as images by learning the similarity between graph pairs with a Convolutional Neural Network(CNN). Although the CNN allows inferring similarity, there is no explainability in the matching performed. Moreover, there is no metrics properties conservation during training. GMN [13] tackles the problem differently by modifying the internal aggregation function of the GNN. The node matching is included directly through a cross-graph aggregation which considers the neighborhood of a node but also a cross-graph matching vector quantifying how well a node in one graph can be matched to nodes in the other. More recently GotSIM [10] uses multiple discrete optimal transport blocks at each layer to match nodes distribution with cosine similarity. Each matching result is contained in a vector given as input to an MLP to learn the GED. This latter doesn't grant any of the basic metric properties during training. Very recently, GEDGNN[18] proposes an original strategy to learn the optimal matching using the GED permutation matrix as learning data in addition to the GED value. To obtain a valid edit path, they perform a costly k-best matching on the learned matrix.

Subgraph-level matching These techniques are at an intermediary level. They aim to match sets of nodes or subgraphs with other sets or sub-graphs. For example, PSimGNN [12] uses matching between a partition of the nodes and pooling of the graphs. More recently, H2MN [15] integrates subgraph matching in the similarity computation through an innovative hypergraph-based strategy. Each graph of a pair is firstly transformed into hypergraphs, before being transformed through dedicated convolution, pooling, and subgraph matching layers. Then the similarity is inferred from the hypergraphs with a readout function and an MLP. Being at an intermediate level, such approaches have interesting properties but suffer from many choices to be made, such as the choice of the pooling technique or the decomposition of the graph into sub-graphs. Moreover, these methods are generally more expensive in terms of processing time and memory usage.

3 GNOME architecture presentation

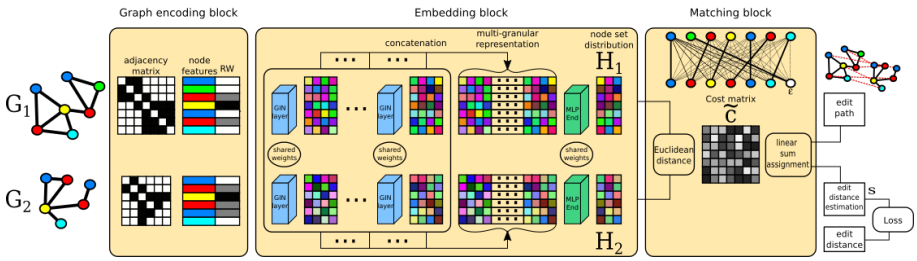

Figure 1: Overview of the GNOME architecture. The Graph encoding block preprocesses the data by adding RW positional encoding (3.1). The Embedding block computes the set of node representations through GIN layers (3.2). The Matching block infers GED values and outputs the corresponding graph matching (3.3).

In this section, we describe the proposed GNOME architecture. As shown by Figure 1, GNOME is composed of three main blocks. The first block consists in enriching the input nodes' representation (features) by a positional encoding, in order to improve the structural expressive power of the model. Then, following the same paradigm as approaches described in section 2, the second block relies on Siamese GNNs. It performs an embedding at the node level, in order to better fit the GED problem definition and to allow explicability of the output. The third block is a distance computing one that processes a matching between the node representation sets of both graphs. This matching is performed by a LSAP solver, in order to keep distance theoretical properties.

3.1 Inputs and positional encoding

In the models reviewed in section 2, Siamese GNNs are classically fed by both the adjacency matrices and the node/edge feature matrices of the graphs. In GNOME-RW, node features are augmented by a vector that encodes the position of the node in the graph. This positional encoding is implemented with the Random-Walk (RW) descriptor [20], defined for a node i by :

$$\mathbf{RW}_i^{(k)} = [(\mathbf{D}^{-1}\mathbf{A})_{(i,i)}, (\mathbf{D}^{-1}\mathbf{A})^2_{(i,i)}, ..., (\mathbf{D}^{-1}\mathbf{A})^{k-1}_{(i,i)}, (\mathbf{D}^{-1}\mathbf{A})^k_{(i,i)}] \quad (1)$$

where $\mathbf{A}$ (resp. $\mathbf{D}$) is the adjacency (resp. degree) matrix. $(\mathbf{D}^{-1}\mathbf{A})^j_{(i,i)}$ gives the probability of a random walk of length j from i to itself. $\mathbf{RW}_i^{(k)}$ gathers all these probabilities for random walks of length j with $j \in \{1, \dots, k\}$. Depending on the value of j, $(\mathbf{D}^{-1}\mathbf{A})^j_{(i,i)}$ gives some structural information related to node i. For example, $(\mathbf{D}^{-1}\mathbf{A})^2_{(i,i)}$ is related to the degree of i. For greater values of j, $(\mathbf{D}^{-1}\mathbf{A})^j_{(i,i)}$ is related to the number of walks of length j (including cycles) starting and ending on node i. Finally, $\mathbf{RW}_i^{(k)}$ brings a k-hop structural

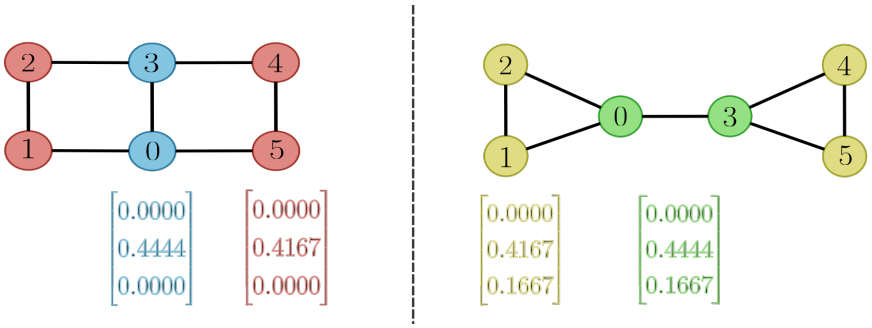

Figure 2: Illustration of RW descriptor (see Eq.1) of length 3 applied on two graphs. The blue vector corresponds to the RW descriptor of blue nodes (the same occurs for other colors). These descriptors allow to distinguish the two graphs while 1-WL test cannot.

description of the neighborhood of i. As illustrated in Figure 2, the proposed Random-Walk positional encoding is able to distinguish nodes that can not be discriminated by the first-order Weisfler-Lehman test [21]. Hence, adding RW to the nodes features helps the matching by improving the ability of the GNN to output different node embeddings despite symmetries in the graph. The dimension k of **RW** is a hyper-parameter of the model.

The impact of this positional encoding on the performance of the model is assessed in the experimental evaluation (4.3.).

3.2 The embedding block: Siamese Graph Isomorphism Networks

The embedding block aims to provide a node representation set for each graph before these representations are matched to compute the graph distance. Both graphs must be processed in a similar way in order to ensure symmetry. For this purpose, the graphs are submitted to Siamese GNNs. The Siamese mechanism is implemented by using two GNNs whose learned weights are shared. We chose the Graph Isomorphism Network(GIN) model[**?**] since GIN is one of the most expressive Messages Passing Neural Networks (MPNNs) with a linear time complexity[22]. The GIN model uses the following equation:

$$\mathbf{h}_v^{(l+1)} = MLP^{(l)}\left(\left(1+\epsilon^{(l)}\right).\mathbf{h}_v^{(l)} + \sum_{u\in\mathcal{N}(v)} \mathbf{h}_u^{(l)}\right) \tag{2}$$

where $\mathbf{h}_v^{(l)}$ is the node representation of node v at layer l, $MLP^{(l)}$ is a Multi-Layer Perceptron (MLP) with at least two layers and ϵ is a parameter which is either fixed or learned.

As shown in eq.2, each layer of GIN updates the representation of node v by aggregating the hidden representation of its neighborhood $\mathcal{N}(v)$ with a non-linear weighting through MLP. That allows, at layer l, to aggregate a structural l-hop information into the node representation. To have better stability and less training time, skip connections are added between layers of the model, if the number of layers exceeds three. The final embedding of each node is then obtained by applying $MLP^{(End)}$ to the concatenation of the node representations at each layer:

$$\mathbf{H}_v = MLP^{(End)}\left(\mathbf{h}_v^{(1)}||\mathbf{h}_v^{(2)}||...||\mathbf{h}_v^{(L-1)}||\mathbf{h}_v^{(L)}\right) \tag{3}$$

where $||$ is the concatenation operator. This corresponds to a multi-granular view of the evolution of the node representation following the 1-WL procedure. At this stage, each graph is embedded as a set (or distribution) $\mathbf{H}$ of node representations in an Euclidean space.

When graph edges are attributed, the GINE architecture[23] (see. Eq. 4) is used instead of GIN, in order to integrate edge features in the computation of node representations.

$$\mathbf{h}_v^{(l+1)} = MLP^{(l)}\left(\left(1+\epsilon^{(l)}\right).\mathbf{h}_v^{(l)} + \sum_{u\in\mathcal{N}(v)} ReLU(\mathbf{h}_u^{(l)} + \mathbf{E}_{u,v})\right) \tag{4}$$

In this equation, $\mathbf{E}_{u,v}$ is an embedding of the feature vector associated with edge (u, v) denoted $e_{(u,v)}$. $\mathbf{E}_{u,v}$ is obtained with a linear layer applied to $e_{(u,v)}$ whose output has the same dimension as the one of $\mathbf{h}_v^{(l)}$.

$$\mathbf{E}_{u,v} = \mathbf{linear}(e_{(u,v)}) \tag{5}$$

3.3 The matching block: Linear Sum Assignment Problem Solver

At the output of the embedding block, each graph is encoded by a set of node vectors. A matching between the two node sets is performed using Linear Sum Assignment Problem (LSAP) solver in order to infer the distance between the two graphs. It is a node-based mapping method that has commonly been used in several works such as [24, 25, 26, 27, 28, 29]. This ensures injective matching and also provides explainability. Moreover, LSAP can be used with a differentiable loss function and has $\mathcal{O}(N^3)$ complexity where N is the number of matching performed.

For graphs with the same size, LSAP formulation is equivalent to Monge's problem of discrete optimal transport theory:

$$\mathbf{LSAP}(\mathbf{C}) = \min_{\mathbf{P}\in\mathcal{P}_\mathcal{N}} \sum_{i=1}^{N}\sum_{j=1}^{N} c_{i,j}p_{i,j} \tag{6}$$

where $\mathbf{C}$ is a square matrix of size N filled with the costs $c_{i,j}$ of matching node i of the first set to node j of the second one and $\mathbf{P} = (p_{i,j})$ is a permutation matrix of size N . $\mathcal{P}_\mathcal{N}$ is the set of all possible permutations of N elements. To compute the $c_{i,j}$, one can use any distance function or dissimilarity. However, the choice of this comparison function is important regarding global metrics properties conservation.

To adapt LSAP to different sizes of graphs, we use the same strategy than [24] and [30]. For two graphs G_1, G_2 of size N_1, N_2, a square cost matrix $\mathbf{C}_{(G_1,G_2)}$ of size $N_1 + N_2$ is defined, by adding dummy nodes ε to guarantee that the matching is still injective, i.e. that an element is sent to only one element. This extension is meaningful as it corresponds to insertion and deletion operations of the GED.

$\mathbf{C}$ can be written as the concatenation of four matrices as follows:

$$\mathbf{C} = \begin{pmatrix} \mathbf{S} & \mathbf{D} \\ \mathbf{I} & 0 \end{pmatrix}, \tag{7}$$

where $\mathbf{S}$ matrix of size $N_1 \times N_2$ corresponds to substitution costs, $\mathbf{D}$ matrix of size $N_1 \times N_1$ to the deletion costs and $\mathbf{I}$ matrix of size $N_2 \times N_2$ to the insertion costs.

For substitution cost inferior to the sum of insertion and deletion costs, one can observe that if $N_1 \geq N_2$ there will never be an insertion in N_1 since that will necessarily result in a supplementary deletion. In this case, substitutions are always favored in the matching provided by the LSAP. As a consequence, $\mathbf{C}$ can be reduced to a square matrix $\tilde{\mathbf{C}}$ of size $\mathbf{max}(N_1, N_2)$. In this new matrix, the substitution block remains unchanged, but either the insertion or deletion block is kept and adapted. That reflects the fact we can order the matching problem by going from the larger set to the smaller (or the opposite). An illustration of the matching, in this case, is illustrated in Figure 3.

The cost matrix $\tilde{\mathbf{C}}$ finally becomes:

$$\tilde{\mathbf{C}} = \begin{pmatrix} \mathbf{S} & \tilde{\mathbf{D}} \end{pmatrix}, \tag{8}$$

with $\tilde{\mathbf{D}}$ the deletion cost matrix of size $N_1 \times (N_1 - N_2)$. This simplification allows for reducing space and time complexity with the same level of performance as shown and proved in [30].

For substitution, we define the $\mathbf{S}$ matrix by:

$$\mathbf{S} = \begin{pmatrix} s_{1,1} & \cdots & s_{1,N_2} \\ \vdots & \ddots & \vdots \\ s_{N_1,1} & \cdots & s_{N_1,N_2} \end{pmatrix}, s_{i,j} = \|\mathbf{H}_{1,i} - \mathbf{H}_{2,j}\|_2, \tag{9}$$

where each $s_{i,j}$ represents the cost to match the embedding $\mathbf{H}_{1,i}$ of the i-th node of G_1 with the embedding $\mathbf{H}_{2,j}$ of the j-th node of G_2 using the Euclidean distance. Note that the MLP layers of the embedding block play a metric learning role, thus mitigating the effects of the norm choice. The choice of a norm-induced distance is important compared to similarities, for which metric

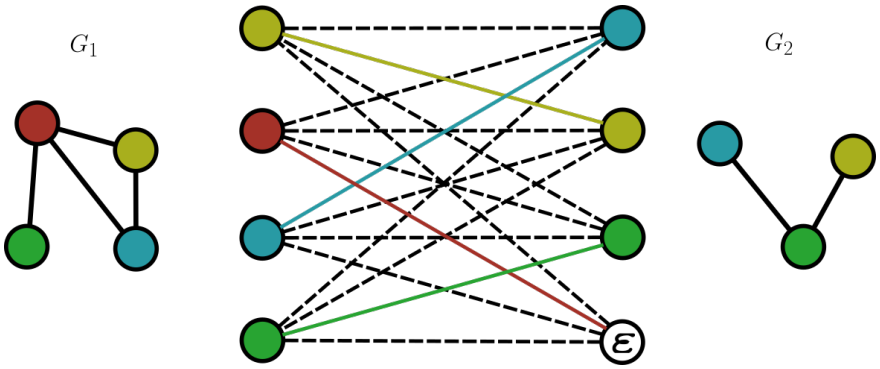

Figure 3: LSAP Bipartite matching applied on two graphs G_1, G_2 with the cost matrix reduction scheme $\tilde{\mathbf{C}}$ (see Eq.8).

properties are not guaranteed.

In order to keep metric properties in the reduced formulation, $\tilde{\mathbf{D}}$ is defined by:

$$\tilde{\mathbf{D}} = \begin{pmatrix} \tilde{d}_{1,1} & \cdots & \tilde{d}_{1,N_1-N_2} \\ \vdots & \ddots & \vdots \\ \tilde{d}_{N_1,1} & \cdots & \tilde{d}_{N_1,N_1-N_2} \end{pmatrix}, \tilde{d}_{i,j} = \|\mathbf{H}_{1,i}\|_2 , \tag{10}$$

The triangle inequality of the Euclidean distance:

$$\|\mathbf{H}_{1,i} - \mathbf{H}_{2,j}\|_2 \leq \|\mathbf{H}_{1,i}\|_2 + \|\mathbf{H}_{2,j}\|_2 \tag{11}$$

allows us to be consistent with the simplification performed.

3.4 Output and Loss

Given the $\tilde{\mathbf{C}}$ cost matrix describing all possible ordered edit operations between both graphs distributions, LSAP is solved to obtain a similarity value $\mathbf{s}$:

$$\mathbf{s}(G_1, G_2) = \mathbf{LSAP}(\tilde{\mathbf{C}}) \tag{12}$$

The whole architecture is fully differentiable and allows end-to-end learning. Mean Squared Error (MSE) is the loss function since the approximation of the GED is a regression task.

$$\mathcal{MSE}_{Loss} = \frac{1}{|\mathcal{T}|} \sum_{(G_1,G_2)\in\mathcal{T}} \|\mathbf{s}(G_1, G_2) - GED(G_1, G_2)\|_2^2 \tag{13}$$

with $\mathcal{T}$ the set of all training pairs of graphs and $\mathbf{s}(G_1, G_2)$ the output cost prediction of eq.12.

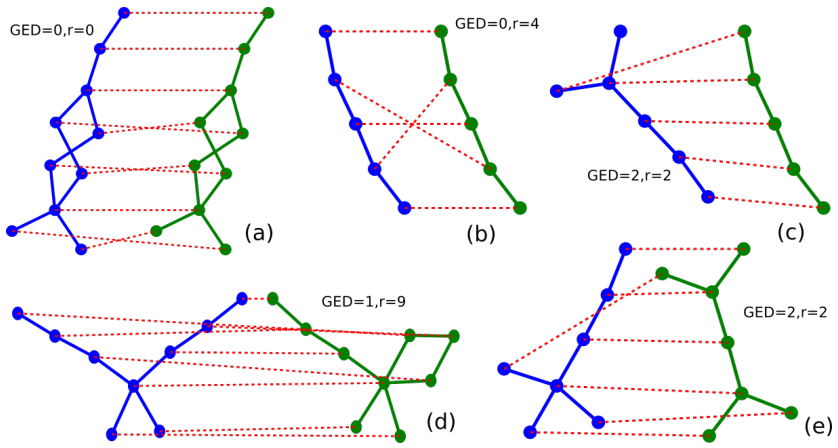

Figure 4: Illustration of graph matching obtained on LINUX dataset. The r value is the cost of the reconstructed edit path. Cases (a) and (b) correspond to isomorphic graphs. In (b) LSAP matching can not deal with symmetric nodes because their embeddings are the same. In (d) only two nodes are mismatched. For (b) and (d), this leads to an extra cost due to edge mismatches, and r largely overestimates the GED. (c) and (e) are cases of node deletion and edge deletion.

The model also outputs the matching obtained by LSAP minimization as presented in Figure 4. This matching is given by the indices of the 1 values in the permutation matrix solution of the LSAP. The edge matching can then be deduced from the node matching to recover an edit path [3]. When computing the cost of this recovered edit path (called r on figure 4), its precision w.r.t. the true GED depends on properties such as node features, graph densities and symmetries. Even a small number of node matching errors will result in a large number of incorrect arc matches, resulting in a large gap between r and the true GED (please see examples (b) and (d) on figure 4).

3.5 Metrics property conservation

We prove in this section that the LSAP has good metrics property conservation when used with a distance function. Formally, GED is a distance thus satisfying the following axioms for all graphs $x, y, z \in G$.

1. (Positivity) If $x \neq y$, then $ged(x, y) > 0$.
2. $ged(x, x) = 0$.
3. (Symmetry) $ged(x, y) = ged(y, x)$. This property is true if and only if the costs of deletion and insertion are equal for nodes and for edges.
4. (Triangle inequality) $ged(x, z) \leq ged(x, y) + ged(y, z)$.

The approximation of GED by a deep neural architecture does not guarantee positivity since the embedding block can generate similar node embedding sets for different graphs, resulting in a null comparison. However, all the other properties hold.

For the second property, the two sets of embedded nodes to compare are the same. In this case, the cost matrix will have at least a zero value in each row and column. Since LSAP is permutation invariant, the matching chosen by the LSAP will have a null cost. Concerning the third property, the LSAP formulation is symmetric by construction. For the triangle inequality, the property holds if a distance is used for the cost matrix computation.

Demonstration of triangle inequality Let $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$ be three graphs. Without loss of generality let's assume $|\mathcal{X}| > |\mathcal{Y}| > |\mathcal{Z}|$. There exist two permutation matrices $P^*_{\mathcal{X},\mathcal{Y}}$ and $P^*_{\mathcal{Y},\mathcal{Z}}$ solutions of LSAP $\left(\tilde{C}_{\mathcal{X},\mathcal{Y}}\right)$ and LSAP $\left(\tilde{C}_{\mathcal{Y},\mathcal{Z}}\right)$. Like in eq. 8 both matrices can be decomposed as:

$$P^*_{\mathcal{X},\mathcal{Y}} = \begin{pmatrix} S_{\mathcal{X},\mathcal{Y}} & \tilde{D}_{\mathcal{X},\mathcal{Y}} \end{pmatrix} \ , \quad P^*_{\mathcal{Y},\mathcal{Z}} = \begin{pmatrix} S_{\mathcal{Y},\mathcal{Z}} & \tilde{D}_{\mathcal{Y},\mathcal{Z}} \end{pmatrix}.$$

Let's construct $\tilde{P}$, the following permutation matrix

$$\tilde{P} = \begin{pmatrix} S_{\mathcal{X},\mathcal{Y}} S_{\mathcal{Y},\mathcal{Z}} & \tilde{D}_{\mathcal{X},\mathcal{Y}} || S_{\mathcal{X},\mathcal{Y}} \tilde{D}_{\mathcal{Y},\mathcal{Z}} \end{pmatrix}$$

We will now show that

$$\sum_{i=1}^{|\mathcal{X}|} \sum_{j=1}^{|\mathcal{X}|} \left(\tilde{C}_{\mathcal{X},\mathcal{Z}}\right)_{i,j} \left(\tilde{P}\right)_{i,j} \leqslant \text{LSAP}\left(\tilde{C}_{\mathcal{X},\mathcal{Y}}\right) + \text{LSAP}\left(\tilde{C}_{\mathcal{Y},\mathcal{Z}}\right).$$

By using a distance for the cost matrices construction, for all k in $|\mathcal{Y}|$, we have: $\left(\tilde{C}_{\mathcal{X},\mathcal{Z}}\right)_{i,j} \leq \left(\tilde{C}_{\mathcal{X},\mathcal{Y}}\right)_{i,k} + \left(\tilde{C}_{\mathcal{Y},\mathcal{Z}}\right)_{k,j}$.

As $\tilde{P}$ is a permutation matrix, if $\left(\tilde{P}\right)_{i,j} = 0$ then for all k in $|\mathcal{Y}|$

$$\left(\tilde{C}_{\mathcal{X},\mathcal{Z}}\right)_{i,j} \left(\tilde{P}\right)_{i,j} \leq \left(\tilde{C}_{\mathcal{X},\mathcal{Y}}\right)_{i,k} \left(P^*_{\mathcal{X},\mathcal{Y}}\right)_{i,k} + \left(\tilde{C}_{\mathcal{Y},\mathcal{Z}}\right)_{k,j} \left(P^*_{\mathcal{Y},\mathcal{Z}}\right)_{k,j}$$

If $\left(\tilde{P}\right)_{i,j} = 1$, by construction of $\tilde{P}$, it means that a unique index k in $|\mathcal{Y}|$ exists such that

$$\left(\tilde{C}_{\mathcal{X},\mathcal{Z}}\right)_{i,j} \left(\tilde{P}\right)_{i,j} \leq \left(\tilde{C}_{\mathcal{X},\mathcal{Y}}\right)_{i,k} \left(P^*_{\mathcal{X},\mathcal{Y}}\right)_{i,k} + \left(\tilde{C}_{\mathcal{Y},\mathcal{Z}}\right)_{k,j} \left(P^*_{\mathcal{Y},\mathcal{Z}}\right)_{k,j}$$

where k corresponds to $\left(P^*_{\mathcal{X},\mathcal{Y}}\right)_{i,k} = 1$ and $\left(P^*_{\mathcal{Y},\mathcal{Z}}\right)_{k,j} = 1$ for the substitution cases. By construction of $\tilde{P}$, deletion cases from $\mathcal{X}$ to $\mathcal{Z}$ correspond also to a deletion either from $\mathcal{X}$ to $\mathcal{Y}$ or from $\mathcal{Y}$ to $\mathcal{Z}$ so the inequality also stands. Doing

Table 1: Main features of the datasets used in the experimental evaluation

Dataset	#graphs	#features	#nodes	#edges
AIDS	700	29	∼8.9	∼17.6
LINUX	1000	0	∼7.6	∼13.9
IMDB	1500	0	∼13.0	∼131.9

the sum on all indexes we obtain:

$$\sum_{i,j=1}^{|\mathcal{X}|} \left(\tilde{C}_{\mathcal{X},\mathcal{Z}}\right)_{i,j} \left(\tilde{P}\right)_{i,j} \leqslant \sum_{i=1}^{|\mathcal{X}|+|\mathcal{Y}|} \left(\tilde{C}_{\mathcal{X},\mathcal{Y}}\right)_{i,k_i} \left(P^*_{\mathcal{X},\mathcal{Y}}\right)_{i,k_i} + \sum_{j=1}^{|\mathcal{Y}|+|\mathcal{Z}|} \left(\tilde{C}_{\mathcal{Y},\mathcal{Z}}\right)_{k_j,j} \left(P^*_{\mathcal{Y},\mathcal{Z}}\right)_{k_j,j}.$$

By definition of the LSAP, as the left term is greater than the LSAP solution, we can then conclude that $\text{LSAP}\left(\tilde{C}_{\mathcal{X},\mathcal{Z}}\right) \leqslant \text{LSAP}\left(\tilde{C}_{\mathcal{X},\mathcal{Y}}\right) + \text{LSAP}\left(\tilde{C}_{\mathcal{Y},\mathcal{Z}}\right)$.

4 Experimental Results

In this section, we describe the datasets and the experimental protocol that has been used to assess GNOME and we discuss the results obtained in comparison to state-of-the-art methods.

4.1 Dataset presentation

Three datasets are used in our main experiments. Their description is given in Table 1. AIDS and IMDB-MULTI come from TUdataset [31]. AIDS consists of graphs representing molecular compounds that are constructed from the AIDS Antiviral Screen Database of Active Compounds. IMDB-MULTI is a relational dataset that consists of ego networks of actors or actresses in IMDB movies. A node represents an actor, and an edge connects two nodes if the corresponding actors appear in the same movie. The LINUX dataset comes from [32]. The three datasets are all freely available on Pytorch Geometric[33] and were first adapted to the GED problem by [16].

4.2 Experimental protocol

The experimental study consists in evaluating the GNOME architecture on a regression task where the value to be predicted is the GED between a pair of graphs provided as input. GNOME-RW is the same model as GNOME but with an additional RW feature vector concatenated to graph node features. Ground-truth values are computed thanks to the ILP F2 method [34] for LINUX and AIDS. For IMDB, the size of the graphs makes the GED costly and even untractable. The ground-truth value associated with each pair is the minimum

between three well-known upper bounds of the GED: Beam, Hungarian, and VJ as described in [16]. Elementary edition costs are set to 1. To assess an information retrieval downstream task and evaluate GNOME ranking abilities, we also compute the Spearman's rank correlation coefficient ρ and the Kendall's rank correlation coefficient τ which evaluate the global ranking preservation of GNOME output compared to the GED. The precision p@10 (resp. p@20) evaluates for each "query" graph in the test database the rate of relevant graphs among the 10 (resp. 20) first predictions given by GNOME.

For both the validation and test sets, we use 200 different graphs for LINUX, 140 for AIDS, and 300 for IMDB. The remaining graphs are used for training. Graph pairs are generated following the protocol defined in [16][33]. Independently of the dataset, the depth of GNOME architecture is fixed to eight GIN layers. A starting learning rate of 10^{-4} is used with a cosine adaptive strategy. The random-walk feature vector results from 12 aggregation steps. The loss function used for training is the Mean Square Error (MSE) and the data are processed in batches of 200.

4.3 Experimental results and discussion

Table 2 gives the results obtained by GNOME and several methods from the literature [10, 5, 9, 15, 16, 18, 17, 6] on the three datasets according to numerous error metrics (the Mean Absolute Error (MAE), Mean Squared Error (MSE), normalized MSE (MSE(10^{-3}))) and ranking metrics ($\rho, \tau, p@10, p@20$)). Among all the evaluated methods, only GENN-A^* [5] and Noah [6] are hybrid methods that use deep learning to drive the A^* search. These methods face scaling issues, resulting in prohibitive execution time (Noah) or even failure (GENN-A^*) on IMDB datasets even though they are effective on datasets with smaller graphs. Other methods use deep learning strategies described in section 2. The results obtained by TaGSim are far worse than for other methods. This can be explained by the fact that this method learns a vector of numbers of GED operation instead of approximating the GED value. GEDGNN [18] did not provide the results for the whole IMDB dataset, so the corresponding entry is empty in the table.

As shown in Table 2, GNOME or GNOME-RW results are always in the top-3 methods and achieve good performance among all deep neural architectures. Note that for the IMDB dataset, the task of predicting the GED is difficult since all graphs are cliques or unions of cliques via a central node. Cliques are strongly regular graphs and GNNs have difficulties in embedding and discriminating nodes of such graphs. This impacts the performance of GNN-based prediction methods but also the matching choice of the LSAP because of the symmetries. Thanks to their ability to preserve the triangle inequality, GNOME and GNOME-RW show good performance on the ranking metrics. The use of GNOME is then relevant for a search in a graph database or in a work like Nass [35].

The effect of the Random-Walk descriptor is not the same for all datasets. Results on AIDS and LINUX corroborate our choice to include RW

Table 2: Results obtained by several methods on benchmark datasets. The results on GED regression are expressed in terms of MSE, MAE, and MSE(10^{-3}) as in [16] (lower is better). Spearman's (ρ) and Kendall's (τ) rank correlation, and precision (p@10 and p@20) are given to measure the ranking preservation (higher is better). For each metric the best result appears in bold. The experimental protocol is the one followed in [9]. Inference times for GNOME and other models are provided in the last column. Results for other models are reported from original papers.

		GED			Ranking				Time
Dataset	Metrics	MAE	MSE	MSE(10^{-3})	ρ	τ	p@10	p@20	(s/100 p)
Linux	SimGNN	0.489	0.443	2.172	0.939	0.830	94.2%	93.3%	0.244
	GREED	0.318	0.172	0.914	-	-	-	-	0.007
	GotSIM	-	0.329	4.25	0.92	0.89	86%	-	-
	H^2MN	0.534	0.538	1.630	0.984	-	95.3%	-	0.087
	GEDGNN	**0.094**	-	-	0.963	0.903	96.2%	97.6%	0.380
	TaGSim	0.391	-	5.278	0.941	0.834	81.6%	86.7%	0.117
	Noah	1.747	-	-	0.874	0.802	90.9%	93.6%	77.237
	GENN-A^*	-	**0.071**	**0.324**	**0.991**	-	96.2%	-	217.7
	GNOME	0.214±0.011	0.104±0.015	0.572±0.075	0.987	0.932	**98.3%**	99.1%	0.2653
	GNOME-RW	0.195±0.006	0.077±0.004	0.423±0.089	0.989	**0.936**	98.1%	**99.3%**	0.2653
AIDS	SimGNN	0.816	1.075	2.238	0.843	0.690	42.1%	51.4%	0.283
	GREED	0.629	0.634	1.334	-	-	-	-	0.005
	GotSIM	-	0.992	2.36	0.86	0.72	87%	-	-
	H^2MN	0.777	0.988	1.913	0.877	-	51.7%	-	0.095
	GEDGNN	0.773	-	-	0.876	0.751	71.6%	77.9%	0.408
	TaGSim	0.841	-	9.827	0.688	0.527	64.6%	32.2%	0.123
	Noah	1.542	4.675	-	0.734	0.560	80.9%	81.2%	168.390
	GENN-A^*	-	0.823	**0.635**	0.959	-	**87.1%**	-	1332.3
	GNOME	0.555±0.009	0.490±0.017	0.990±0.023	0.949	0.846	80.4%	84.5%	0.265
	GNOME-RW	**0.508±0.008**	**0.407±0.014**	0.849±0.032	**0.959**	**0.863**	81.3%	**86.7%**	0.265
IMDB	SimGNN	28.082	4389.06	5.618	0.878	0.770	75.9%	77.7%	0.258
	GREED	3.612	45.347	1.243	-	-	-	-	0.006
	GotSIM	-	4424.9	5.92	0.85	0.80	73%	-	-
	H^2MN	28.486	7409.25	0.744	0.912	-	86.1%	-	0.083
	GEDGNN	-	-	-	-	-	-	-	-
	TaGSim	-	-	35.69	0.958	0.926	-	**98.6%**	0.113
	Noah	3.755	55.636	-	0.810	0.716	35.4%	40.5%	5201.6
	GENN-A^*	-	-	-	-	-	-	-	-
	GNOME	2.976±0.004	**33.433±0.054**	0.831±0.021	0.994	0.953	87.7%	87.8%	0.272
	GNOME-RW	**2.920±0.007**	33.769±0.072	**0.722±0.053**	**0.995**	**0.955**	**88.2%**	88.6%	0.272

positional encoding to improve the expressivity of the embedding block. The GIN layers can produce different embeddings for symmetric nodes. For the IMDB dataset, since every node in a clique has the same RW descriptor, RW has no impact, and so, GNOME-RW does not outperform GNOME. The choice of the descriptor length depends on the data, but a common observation is that beyond a moderate value (11 for AIDS and Linux), increasing the descriptor length does not lead to better performance.

Runtime : GNOME has the same runtime order of magnitude as other deep approaches, except for GREED, which is linear since it does not perform matching at the node level, and A^*-based methods (Noah, GENN-A^*) which are much slower since they depend on tree exploration.

4.4 Ablative study on MAO dataset

This ablative study aims to measure the influence of edge encoding and RW positional encoding on the performance of GNOME. In such a context, experiments

Table 3: Ablative study on MAO dataset, results expressed in MSE.

Cost	vanilla	RW	Edge	Edge + RW
(4,2,1,1)	1.12 ± .11	0.72 ± .10	0.27 ± .08	**0.11 ± .03**
(4,2,2,1)	2.39 ± .23	1.09 ± .19	0.85 ± .16	**0.45 ± .14**
(2,6,1,3)	2.37 ± .29	1.81 ± .25	0.24 ± .06	**0.12 ± .03**

Cost = c_{n_ins/n_del}, c_{n_sub}, c_{e_del/e_ins}, c_{e_sub}

are conducted on the Mono-amide oxidase dataset (MAO) from the GREYC challenge [7]. MAO is a small dataset composed of 67 molecule graphs of mean size 17, with nodes and edges features. The task consists of predicting the GED of the 4489 pairs. The study is performed with a 5-fold cross-validation.

Table 3 presents the results obtained for different cost settings configurations and how the model performs with such costs. As one can see, GNOME adapts well to different costs even with the third ones which are defined by inverted triangle inequality between operations. In this case, the inference of the cost is good but the matching should not be as accurate. As expected, using GINE to take edge features into account improves performance. The results also show that the RW positional encoding provides better node representations and thus better matching.

5 Conclusion

The paper presents the GNOME architecture which learns an approximation of the GED. The architecture uses GIN layers to embed graphs and LSAP on node distribution to operate matching. This linear relaxation of QAP is compensated by the representation power of GNNs. GNOME keeps the explainability of GED since node embedding is closer to the initial problem than graph-level representation. Moreover GNOME conserves the GED metric properties. Experimental results show that GNOME outperforms all existing machine learning-based models. GNOME is not dedicated to the GED problem. It could also learn any other distance between graphs for example task-dependent metrics. In this case, the distance will be task-driven but the graph embedding will also benefit from GNOME's good metrics properties and explainability. Another direction is toward the expressivity of the embedding block and the possibility to take into account edges and edges matching to have a finer prediction and better edit path output.

Acknowledgments and reproducibility: This work has been supported by the ANR-20-THIA-0021 and ANR-21-CE23-0025 grants. The source code of this work is available at `https://github.com/aldomos/GNOME`.

References

[1] Michel Neuhaus and Horst Bunke. *Bridging the Gap Between Graph Edit Distance and Kernel Machines*. World Scientific Publishing Co., Inc., USA, 2007.

[2] Michel Neuhaus and Horst Bunke. A quadratic programming approach to the graph edit distance problem. In *Graph-Based Representations in Pattern Recognition*, pages 92–102. Springer, Berlin, Heidelberg, 2007.

[3] Kaspar Riesen and Horst Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision Computing*, 27(7):950–959, 2009.

[4] 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, feb 2015.

[5] Runzhong Wang, Tianqi Zhang, Tianshu Yu, Junchi Yan, and Xiaokang Yang. Combinatorial learning of graph edit distance via dynamic embedding. In *2021 IEEE/CVF Conf. on Computer Vision and Pattern Recognition*, pages 5237–5246, Los Alamitos, CA, USA, jun 2021.

[6] Lei Yang and Lei Zou. Noah: Neural-optimized a* search algorithm for graph edit distance computation. In *2021 IEEE 37th Int. Conf. on Data Engineering*, pages 576–587, 2021.

[7] Zeina Abu-Aisheh, Benoit Gaüzère, Sébastien Bougleux, Jean-Yves Ramel, Luc Brun, Romain Raveaux, Pierre Héroux, and Sébastien Adam. Graph edit distance contest: Results and future challenges. *Pattern Recognition Letters*, 100:96–103, 2017.

[8] David B. Blumenthal, Nicolas Boria, Johann Gamper, Sébastien Bougleux, and Luc Brun. Comparing heuristics for graph edit distance computation. *The VLDB Journal*, 29(1):419–458, jul 2019.

[9] Rishabh Ranjan, Siddharth Grover, Sourav Medya, Venkatesan Chakaravarthy, Yogish Sabharwal, and Sayan Ranu. GREED: A neural framework for learning graph distance functions. In *Advances in Neural Information Processing Systems*, 2022.

[10] Khoa D. Doan, Saurav Manchanda, Suchismit Mahapatra, and Chandan K. Reddy. Interpretable graph similarity computation via differentiable optimal alignment of node embeddings. In *Proc. of the 44th Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*, SIGIR '21, page 665–674, New York, NY, USA, 2021. ACM.

[11] Yunsheng Bai, Hao Ding, Ken Gu, Yizhou Sun, and Wei Wang. Learning-based efficient graph similarity computation via multi-scale convolutional set matching. *Proc. of the AAAI Conf. on Artificial Intelligence*, 34(04):3219–3226, Apr. 2020.

[12] Haoyan Xu, Ziheng Duan, Yueyang Wang, Jie Feng, Runjian Chen, Qianru Zhang, and Zhongbin Xu. Graph partitioning and graph neural network based hierarchical graph matching for graph similarity computation. *Neurocomputing*, 439:348–362, 2021.

[13] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. In *Proc. of the 36th Int. Conf. on Machine Learning*, volume 97, pages 3835–3845. PMLR, 09–15 Jun 2019.

[14] Xiang Ling, Lingfei Wu, Saizhuo Wang, Tengfei Ma, Fangli Xu, Alex X. Liu, Chunming Wu, and Shouling Ji. Multilevel graph matching networks for deep graph similarity learning. *IEEE Transactions on Neural Networks and Learning Systems*, 34(2):799–813, feb 2023.

[15] Zhen Zhang, Jiajun Bu, Martin Ester, Zhao Li, Chengwei Yao, Zhi Yu, and Can Wang. H2mn: Graph similarity learning with hierarchical hypergraph matching networks. In *Proc. of the 27th ACM SIGKDD Conf. on Knowledge Discovery and Data Mining*, KDD '21, page 2274–2284, New York, NY, USA, 2021. ACM.

[16] Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. Simgnn: A neural network approach to fast graph similarity computation. In *Proc. of the Twelfth ACM Int. Conf. on Web Search and Data Mining*, WSDM '19, page 384–392, New York, NY, USA, 2019. ACM.

[17] Jiyang Bai and Peixiang Zhao. Tagsim: Type-aware graph similarity learning and computation. *Proc. VLDB Endow.*, 15(2):335–347, oct 2021.

[18] Chengzhi Piao, Tingyang Xu, Xiangguo Sun, Yu Rong, Kangfei Zhao, and Hong Cheng. Computing graph edit distance via neural graph matching. *Proc. VLDB Endow.*, 16(8):1817–1829, apr 2023.

[19] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th Int. Conf. on Learning Representations, ICLR 2017, Conference Track Proceedings*, 2017.

[20] Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. In *The Tenth Int. Conf. on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*, 2022.

[21] Boris Weisfeiler and Andrei Lehman. The reduction of a graph to canonical form and the algebra which appears therein. *Nauchno-Technicheskaya Informatsia, Seriya 22 9. In Russian*, page 12–16, 1968.

[22] Muhammet Balcilar, Pierre Héroux, Benoit Gaüzère, Pascal Vasseur, Sébastien Adam, and Paul Honeine. Breaking the limits of message passing graph neural networks. In *Proc. of the 38th Int. Conf. on Machine Learning*, June 2021.

[23] Weihua Hu*, Bowen Liu*, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *Int. Conf. on Learning Representations*, 2020.

[24] Kaspar Riesen, Michel Neuhaus, and Horst Bunke. Bipartite graph matching for computing the edit distance of graphs. In *Graph-Based Representations in Pattern Recognition*, pages 1–12, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[25] Xiaoyang Chen, Hongwei Huo, Jun Huan, and Jeffrey Scott Vitter. An efficient algorithm for graph edit distance computation. *Knowledge-Based Systems*, 163:762–775, 2019.

[26] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[27] Roy Jonker and Ton Volgenant. Improving the hungarian assignment algorithm. *Operations Research Letters*, 5(4):171–175, 1986.

[28] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[29] Stefan Fankhauser, Kaspar Riesen, and Horst Bunke. Speeding up graph edit distance computation through fast bipartite matching. In *Graph-Based Representations in Pattern Recognition*, pages 102–111.

[30] Lijun Chang, Xing Feng, Xuemin Lin, Lu Qin, Wenjie Zhang, and Dian Ouyang. Speeding up ged verification for graph similarity search. In *2020 IEEE 36th Int. Conf. on Data Engineering*, pages 793–804, 2020.

[31] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020.

[32] Xiaoli Wang, Xiaofeng Ding, Anthony K.H. Tung, Shanshan Ying, and Hai Jin. An efficient graph indexing method. In *2012 IEEE 28th Int. Conf. on Data Engineering*, pages 210–221, 2012.

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

[34] Julien Lerouge, Zeina Abu-Aisheh, Romain Raveaux, Pierre Héroux, and Sébastien Adam. Exact graph edit distance computation using a binary linear program. In *Structural, Syntactic, and Statistical Pattern Recognition*, pages 485–495. Springer International Publishing, Cham, 2016.

[35] Jongik Kim. Boosting graph similarity search through pre-computation. In *Proc. of the 2021 Int. Conf. on Management of Data*. ACM, jun 2021.