Master Thesis

Federico Calabrò 247976

Department of Mathematics and Computer Science
2024

Contents

1 Introduction

Graphs are fundamental structures in computer science and mathematics that model relationships between entities. They consist of vertices (or nodes) and edges (or links) that connect pairs of vertices. This simple yet powerful representation can capture a wide variety of real-world scenarios, providing a versatile tool for analyzing complex systems. For instance, social networks can be represented as graphs where nodes denote individuals, and edges represent their connections or interactions, enabling the study of social dynamics, influence, and community formation. In biology, protein-protein interaction networks, neural networks of the brain, and ecological networks can all be modeled as graphs, facilitating the understanding of biological processes, brain functionality, and ecosystem interdependencies. Similarly, in transportation, cities can be nodes and roads or flights can be edges, creating a network that facilitates route optimization, urban planning, and logistical efficiency. Graphs can also model communication networks, where devices are nodes and connections are edges, allowing for analysis of data flow, network robustness, and optimization of resource allocation.

Graph theory provides a rich framework for analyzing these structures, with properties such as connectivity, centrality, and clustering coefficient helping to understand the underlying patterns and behaviors within the network. Connectivity measures how well the nodes are connected, centrality identifies the most important nodes within the graph, and clustering coefficient gives insight into the degree to which nodes tend to cluster together. Additionally, other important graph properties include graph diameter, which measures the longest shortest path between any two nodes, and graph density, which indicates the level of interconnectedness in the network. These properties help in uncovering critical information about the structure and function of the graph, enabling more effective analysis and decision-making.

The Graph Edit Distance (GED) problem is a critical measure in graph theory, providing a similarity metric between two graphs. GED quantifies how many operations (such as insertions, deletions, and substitutions of nodes and edges) are required to transform one graph into another. This measure is invaluable for various applications, including bioinformatics, where it can compare molecular structures to identify potential drug candidates or understand evolutionary relationships. In computer vision, GED is crucial for object recognition, where the structural similarity between graphical representations of different objects must be assessed to identify and classify them accurately. Other graph similarity measures include graph isomorphism, which checks for exact structural similarity, and subgraph isomorphism, which identifies if one graph is a subgraph of another, useful for pattern matching and searching within larger networks.

Knowing the exact GED between two graphs can provide profound insights. For example, in bioinformatics, understanding the similarity between different molecular structures can lead to the discovery of new drugs and therapeutic targets by revealing structural patterns that correlate with biological activity.

In social network analysis, GED can help detect communities or clusters of users with similar interaction patterns, aiding in the identification of influential individuals, the spread of information, or the formation of social groups. Moreover, in pattern recognition and image analysis, GED can assist in identifying objects and understanding their structural relationships, enhancing the accuracy and reliability of automated systems. The ability to quantify the similarity between graphs allows for more precise and meaningful comparisons, driving innovations and improvements across these fields.

However, computing the exact GED is notoriously difficult due to its high computational complexity. The problem is NP-hard, meaning that the time required to solve it grows exponentially with the size of the graphs, making it computationally prohibitive for large graphs. This involves an exhaustive search over all possible edit paths, which is impractical for real-world applications. Various heuristics and approximation algorithms have been proposed, but they often struggle to balance accuracy and computational efficiency, leading to trade-offs that can impact the reliability of the results. The NP-hard class encompasses problems that are at least as hard as the hardest problems in NP, and no known polynomial-time algorithm can solve them. This inherent difficulty underscores the challenge of computing GED and the necessity for developing efficient approximation methods that can provide accurate results within reasonable timeframes.

Neural networks, a cornerstone of modern machine learning, have revolutionized numerous fields by providing robust methods for handling complex, high-dimensional data. A neural network is a series of algorithms that attempt to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. They consist of interconnected layers of nodes, or neurons, each capable of processing inputs and producing outputs. Neural networks are trained using large datasets where they adjust their internal parameters based on the error between the predicted outputs and the actual outputs. This training process involves forward propagation, where inputs are passed through the network to generate outputs, and backpropagation, where the error is propagated back through the network to update the weights, thereby improving the model's accuracy. Neural networks have been successfully applied to a wide range of tasks, including image and speech recognition, natural language processing, and more recently, graph-structured data analysis, demonstrating their versatility and effectiveness.

Graph Neural Networks (GNNs) are a specialized type of neural network designed to work directly with graph-structured data. GNNs aim to leverage the graph's inherent structure by performing convolution operations over the nodes and edges, capturing both local and global graph properties. This makes them well-suited for various tasks, including node classification, link prediction, and graph classification. Given their ability to learn complex patterns and representations, GNNs hold promise for approximating the GED. GNNs operate by iteratively updating the representation of each node based on its neighbors, effectively capturing the dependencies and relationships within the graph. This iterative process enables GNNs to learn hierarchical representations that are cru-

cial for understanding and analyzing graph-structured data, allowing for more accurate and meaningful predictions in various applications.

This thesis reviews a range of key articles to explore the current state-of-the-art methods in GED computation, encompassing both neural network-based approaches and traditional methods. The reviewed works span various innovative strategies, each attempting to tackle the challenges of GED computation from different angles. By critically analyzing these methods, this review aims to identify their strengths and limitations, offering insights into potential improvements. The seminal paper on SimGNN [1] serves as a foundation for many subsequent works, introducing a neural network-based approach to GED computation that has inspired numerous advancements. More recent works, such as GedGNN [10], continue to push the boundaries of what is possible, integrating novel techniques and improving upon previous methods.

Improving GED computation methods is crucial for enhancing the performance of numerous applications that rely on graph similarity measures. For instance, more efficient and accurate GED computation can lead to breakthroughs in drug discovery by enabling faster and more precise comparison of molecular structures, facilitating the identification of new compounds with therapeutic potential. In social network analysis, it can facilitate the detection of more accurate community structures, leading to better understanding and management of social dynamics, improving the effectiveness of interventions and policy decisions. In computer vision, improved methods can enhance object recognition systems, making them more reliable and efficient, which is vital for applications ranging from autonomous vehicles to security systems. The implications of better GED computation extend to numerous domains, highlighting the importance of continued research and development in this area to unlock new possibilities and advancements.

As we delve into the review of these articles, the goal is to provide a comprehensive overview of the advancements in GED computation. By highlighting innovative strategies and pinpointing areas for further research, this thesis aims to contribute to the ongoing efforts to refine and enhance GED computation methods. We will reproduce the results of key recent papers, such as the one proposing GedGNN, to validate their findings. Additionally, this thesis will offer critical and constructive advice on aspects such as code quality, the fairness of presented results, and the limitations of the datasets used. We will discuss issues like poor dataset quality and propose solutions, including artificial dataset generation and the development of neural networks that can be tested on any dataset, ensuring a fairer evaluation. This comprehensive review aims to bridge the gap between existing methods and the potential for new, more effective techniques, ultimately contributing to the broader field of graph theory and its myriad applications. By providing a thorough analysis and constructive feedback, this thesis seeks to guide future research and development efforts, paving the way for advancements that will enhance the accuracy, efficiency, and applicability of GED computation methods in various fields.

2 Graph Data Structure

A *graph* G [Figure 1] is a non-linear data structure consisting of vertices and edges. The vertices are sometimes also referred to as nodes and the edges are arcs that connect any two nodes in the graph. Graphs are used to represent relationships between different entities and have applications in many fields including Computer Science, Physics, Biology, Chemistry, Optimization Theory and many more.

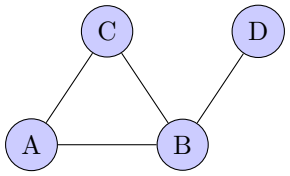


Figure 1: A simple graph

A graph is formally defined as a tuple $G = (V, E)$, where:

- V is a finite set of vertices (or nodes).
- E is a set of edges, where each edge is an unordered pair of distinct vertices from V. Thus, $E \subseteq \{\{u, v\} \mid u, v \in V \text{ and } u \neq v\}$.

2.1 Types of Graphs

Graphs can be classified into various types based on their properties, including:

- **Directed Graph** [Figure 2]: A graph in which the edges have a direction, i.e., each edge is an ordered pair of vertices.

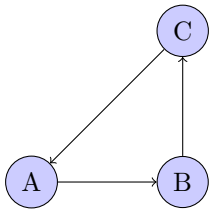


Figure 2: Directed graph example.

- **Undirected Graph** [Figure 3]: A graph in which the edges do not have a direction, i.e., each edge is an unordered pair of vertices.

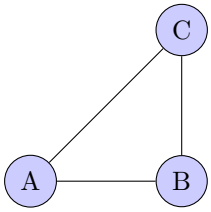


Figure 3: Undirected graph example.

- **Weighted Graph** [Figure 4]: A graph in which a weight (or cost) is associated with each edge.

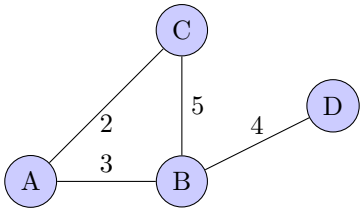


Figure 4: Weighted graph example.

- **Simple Graph** [Figure 5]: A graph with no loops (edges connecting a vertex to itself) and no multiple edges (more than one edge connecting the same pair of vertices).

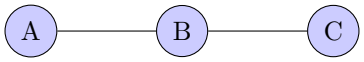


Figure 5: Simple graph example.

- **Complete Graph** [Figure 6]: A graph in which there is exactly one edge between each pair of distinct vertices.

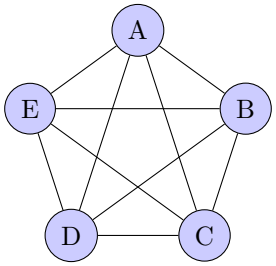


Figure 6: Complete graph example.

- **Bipartite Graph** [Figure 7]: A graph whose vertices can be divided into two disjoint sets U and W such that every edge connects a vertex in U to a vertex in W.

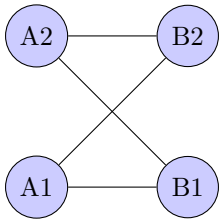


Figure 7: Bipartite graph example.

2.2 Graph Representation

Graphs can be represented in various ways, including:

- **Adjacency Matrix** [Figure 8]: A square matrix A of size $|V| \times |V|$ where $A_{ij} = 1$ if there is an edge between vertices v_i and v_j, and $A_{ij} = 0$ otherwise.

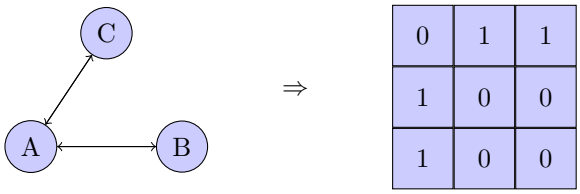
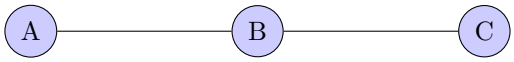


Figure 8: Adjacency Matrix representation.

- **Adjacency List** [Figure 9]: An array of lists. The array contains a list for each vertex, and each list contains the vertices that are adjacent to the corresponding vertex.



$$[\ Adj(A) = [B], \quad Adj(B) = [A, C], \quad Adj(C) = [B]\ ]$$

Figure 9: Adjacency List representation for an undirected graph.

3 Neural Networks

A *neural network* is a computational model that is inspired by the way biological neural networks in the human brain process information. It consists of intercon-

nected units called neurons that work together to solve specific problems. The basic building block of a neural network is the perceptron, which computes a weighted sum of its inputs and passes the result through an activation function.

3.1 Basic Structure of a Neural Network

A simple neural network [Figure 10] consists of three types of layers:

- **Input Layer**: The layer that receives the input data.
- **Hidden Layers**: One or more intermediate layers that process the inputs received from the input layer.
- **Output Layer**: The layer that produces the final output.

The following figure illustrates a basic neural network with one hidden layer:

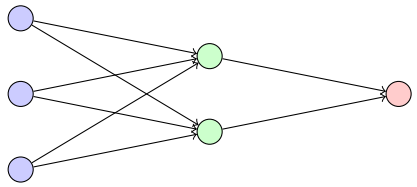


Figure 10: A simple neural network with one hidden layer.

3.2 Training Neural Networks

Training a neural network involves adjusting the weights of the connections to minimize the error between the predicted output and the actual output. This is typically done using a method called *backpropagation* along with an optimization algorithm like *gradient descent*. The training process involves multiple iterations, where in each iteration, the network processes a batch of input data, calculates the output, computes the error, and updates the weights to reduce this error.

3.2.1 Backpropagation

Backpropagation [Figure 11] is an algorithm used to compute the gradient of the loss function with respect to each weight by the chain rule, iterating backward from the last layer to the first layer. During the forward pass, the input data propagates through the network layer by layer until the final output is obtained. The error is then calculated using a loss function, which measures the difference between the predicted output and the actual output. In the backward pass, this error is propagated back through the network, allowing the algorithm to

compute the gradients of the loss function with respect to each weight. These gradients indicate how much the loss would change with a small change in the weights, guiding the weight update process.

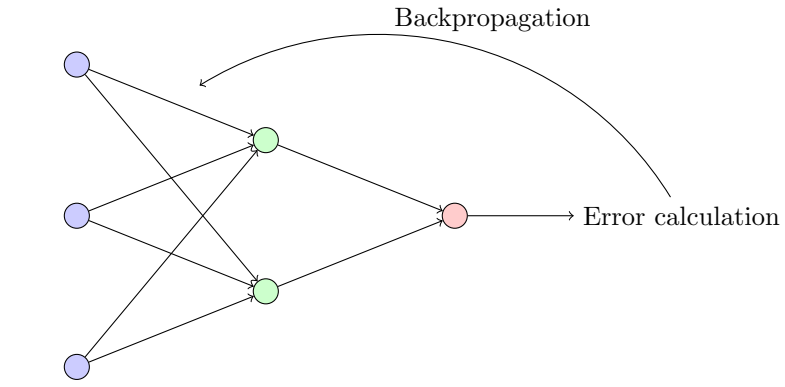


Figure 11: A simple neural network with one hidden layer.

3.2.2 Gradient Descent

Gradient descent [Figure 12] is an optimization algorithm used to minimize the loss function by iteratively moving towards the steepest descent, based on the computed gradients. At each iteration, the algorithm updates the weights by moving them in the opposite direction of the gradient of the loss function with respect to the weights. The size of these steps is determined by the learning rate, a hyperparameter that needs to be carefully chosen. A too large learning rate can cause the algorithm to overshoot the minimum, while a too small learning rate can make the convergence excessively slow.

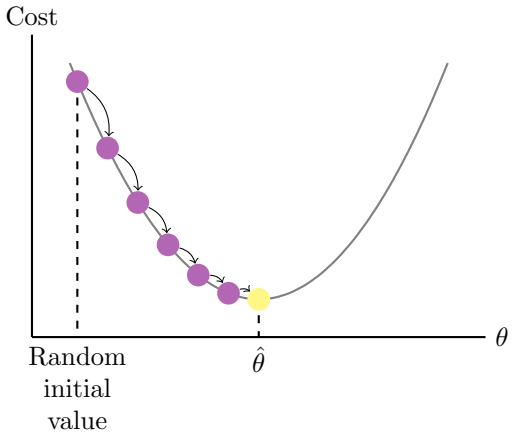


Figure 12: Gradient descent method applied on a simple function.

3.3 Advanced Topics in Neural Networks

3.3.1 Deep Neural Networks

A *deep neural network* (DNN) [Figure 13] is an artificial neural network with multiple hidden layers between the input and output layers. DNNs can model complex patterns and relationships in data, making them powerful tools for tasks such as image and speech recognition. The increased depth allows these networks to learn hierarchical representations of the data, capturing low-level features in the early layers and high-level features in the deeper layers.

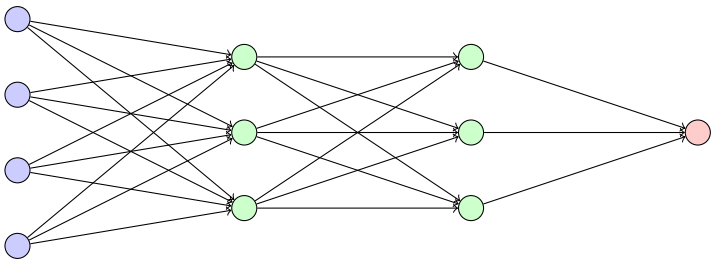


Figure 13: A Deep neural network with two hidden layers.

3.3.2 Convolutional Neural Networks

A *convolutional neural network* (CNN) [Figure 14] is a specialized type of neural network designed for processing structured grid data, like images. CNNs use convolutional layers that apply a series of filters to the input data to extract

features. These filters slide over the input data, performing element-wise multiplications and summing the results, which helps in detecting patterns such as edges, textures, and shapes.

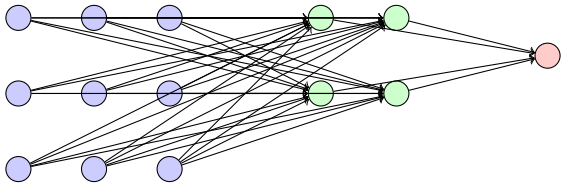


Figure 14: A simple convolutional neural network architecture.

3.3.3 Recurrent Neural Networks

A *recurrent neural network* (RNN) [Figure 15] is a type of neural network that is well-suited for sequential data, such as time series or natural language. RNNs have connections that form directed cycles, allowing information to persist. This cyclic structure enables RNNs to maintain a hidden state that captures information from previous time steps, making them effective for tasks where the context is important, such as language modeling and machine translation.

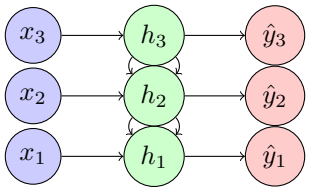


Figure 15: Recurrent Neural Network (RNN) unrolled through time.

3.4 Graph Neural Networks and Attention Mechanisms

3.4.1 Graph Neural Networks

A *graph neural network* (GNN) is a type of neural network designed to handle graph-structured data. GNNs generalize neural networks to work directly on the graph domain by incorporating the graph's structure into the learning process. Nodes in a GNN aggregate feature information from their neighbors through multiple layers, enabling the network to capture the dependencies and relationships inherent in the graph. This message-passing mechanism [Figure 16] allows GNNs to learn rich node representations and can be used for tasks such as node classification, link prediction, and graph classification.

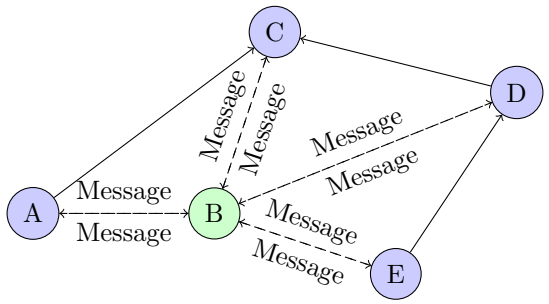


Figure 16: Message passing between node B and its neighbors in a GNN.

3.4.2 Attention Mechanisms

Attention mechanisms are techniques used to enhance the performance of neural networks by dynamically focusing on relevant parts of the input data while processing. The most famous use of attention mechanisms is in the *Transformer* model, which has revolutionized natural language processing tasks. The attention mechanism [Figure 17] computes a weighted sum of values, where the weights are derived from the compatibility of the query with the corresponding keys, allowing the model to selectively focus on important parts of the input. This selective focus helps improve the efficiency and accuracy of the model, especially in tasks involving sequences, such as machine translation and text summarization. The combination of GNNs and attention mechanisms has shown significant promise in various applications.

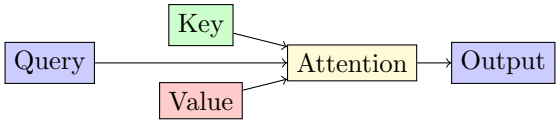


Figure 17: Attention mechanism in neural networks.

4 Graph Similarity Problem

The *graph similarity problem* involves determining the degree of similarity between two graphs. This problem has numerous applications in pattern recognition, computer vision, bioinformatics, and other fields. One common method to quantify graph similarity is through the *graph edit distance* (GED).

4.1 Graph Edit Distance (GED)

The *graph edit distance* between two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is defined as the minimum cost required to transform G_1 into G_2 using a sequence of atomic operations. Formally, let Σ be the set of all possible edit operations,

and let $c : \Sigma \to \mathbb{R}^+$ be a cost function that assigns a positive real number to each operation. The GED is then given by:

$$\text{GED}(G_1, G_2) = \min_{\sigma \in \Sigma^*} \sum_{o \in \sigma} c(o)$$

where Σ^* denotes the set of all finite sequences of operations from Σ, and o represents an individual operation within a sequence σ.

The computation of GED is known to be **NP-HARD**, indicating that finding the exact minimum edit distance between two graphs is computationally intensive.

4.2 Atomic Operations

The basic atomic operations typically includes:

- **Vertex Insertion**: Inserting a new vertex v into the graph.
- **Vertex Deletion**: Deleting an existing vertex v from the graph.
- **Edge Insertion**: Inserting a new edge $e = \{u, v\}$ into the graph.
- **Edge Deletion**: Deleting an existing edge $e = \{u, v\}$ from the graph.

To demonstrate the Graph Edit Distance (GED), consider pair of graphs represented in [Figure 18]:

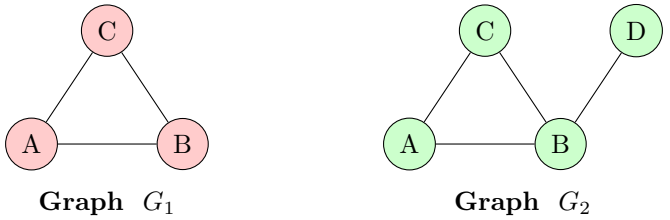


Figure 18: Pair of graphs to demonstre Graph Edit Distance.

In this example, graph G_1 has vertex set $V_1 = \{A, B, C\}$ and edge set $E_1 = \{\{A, B\}, \{A, C\}, \{B, C\}\}$, while graph G_2 has vertex set $V_2 = \{A, B, C, D\}$ and edge set $E_2 = \{\{A, B\}, \{A, C\}, \{B, C\}, \{B, D\}\}$. The transformation (which cost is lowest) from G_1 to G_2 involves: Inserting the vertex D and Inserting the edge $\{B, D\}$. If we assign a cost of 1 to each operation the total cost (GED) is: $1 + 1 = 2$.

5 State of the Art Review

Traditional imperative solutions for solving the graph edit distance (GED) problem do exist and can be quite effective for graphs of modest size. These methods typically involve explicit algorithms that compare graphs node by node and

edge by edge, often using combinatorial techniques. However, as the number of nodes in the graph increases, the complexity of these methods grows exponentially, leading to scalability issues. In such cases, imperative solutions often become impractical, taking an inordinate amount of time to compute the GED.

To address these scalability challenges, recent research has focused on leveraging artificial intelligence (AI) models. These AI-based approaches offer more robust and scalable solutions by learning patterns and features from graph data, significantly reducing computation times and improving accuracy. In this section, we review some of the prominent AI-based models that have advanced the state of the art in GED computation, including SimGNN, GPN, TaGSim, and GedGNN.

5.1 SimGNN

The first innovative model that significantly outperformed the competition is SimGNN [1], introduced in 2019. SimGNN serves as a foundational model in the field of graph similarity computation, and subsequent models often inherit its core concepts, making it the starting point of reference for anyone working in this niche field.

SimGNN is an end-to-end neural network-based approach designed to learn a function that maps a pair of graphs to a similarity score. An overview of SimGNN is illustrated in Figure 19. The architecture of SimGNN involves several key stages:

- **Node Embedding Stage**: Each node in the graph is transformed into a vector that encodes its features and structural properties.
- **Graph-Level Embedding Stage**: The node embeddings are aggregated using an attention mechanism to produce a single embedding for each graph.
- **Graph-Graph Interaction Stage**: The graph-level embeddings of the two graphs are interacted to produce interaction scores that represent the similarity between the graphs.
- **Final Similarity Score Computation Stage**: The interaction scores are further processed to compute the final similarity score, which is compared against the ground-truth similarity score for parameter updates.

In addition to the graph-level embedding interaction strategy, SimGNN incorporates a pairwise node comparison strategy:

- **Pairwise Node Comparison**: This strategy involves computing pairwise interaction scores between the node embeddings of the two graphs. For graphs of different sizes, fake nodes with zero embeddings are added to the smaller graph to ensure compatibility. The resulting similarity matrix is used to extract histogram features, which are then combined with graph-level interaction scores to provide a comprehensive view of graph similarity.

The combination of these two strategies allows SimGNN to capture both coarse global comparison information and fine-grained node-level comparison information, resulting in a robust and thorough approach to graph similarity computation.

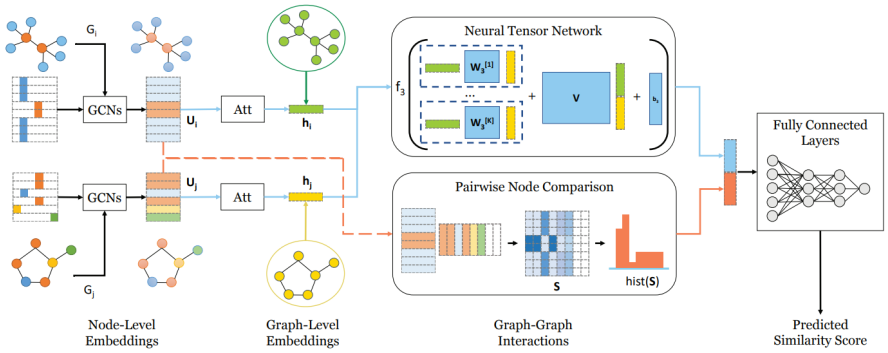


Figure 19: SimGNN architecture overview.

5.2 GedGNN

Very recently (2023), the GedGNN [10] model introduced significant advancements in the field of graph similarity computation, demonstrating substantial performance improvements over previous models like SimGNN. An overview of GedGNN is illustrated in Figure 20. The architecture of GedGNN comprises several key components:

- **Node Embedding Generation**: GedGNN employs a Graph Neural Network (GNN) to generate node embeddings for the input graphs. Specifically, it utilizes a three-layer Graph Isomorphism Network (GIN), which is known for its exceptional ability to capture intricate graph structures.
- **Cross Matrix Modules**: GedGNN incorporates two distinct cross matrix modules to capture node-to-node correspondences between the embeddings of the two graphs:
  - One module produces a matching matrix (A_{match}), which predicts the ground-truth matching matrix.
  - The other module generates a cost matrix (A_{cost}), where each element represents the cost of edit operations required to align nodes between the two graphs.
- **Similarity Score Prediction**: The final similarity score, representing the graph edit distance (GED) value, is computed by calculating the weighted sum of costs in the cost matrix, with an additional bias value.

Compared to SimGNN, GedGNN offers several key advancements, including enhanced efficiency, increased accuracy, and greater robustness. Additionally, GedGNN is capable of retrieving an edit path between the two graphs in input. A post-processing algorithm based on k-best matching is employed to derive k potential node matchings from the matching matrix produced by GedGNN, with the best matching leading to a high-quality edit path.

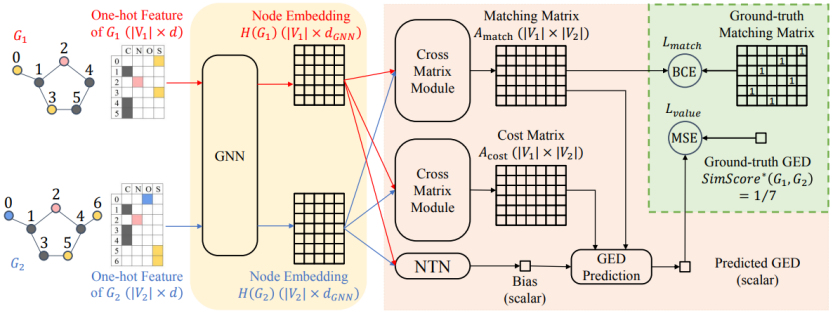


Figure 20: GedGNN architecture overview.

5.3 Other Relevant Mentions

This section highlights additional notable contributions in the field of graph similarity and edit distance computation, showcasing a range of innovative approaches and models.

- Graph Partitioning and Graph Neural Network Based Hierarchical Graph Matching for Graph Similarity Computation [15] proposes **PSimGNN**, which partitions graphs and uses a graph neural network for efficient similarity computation, combining coarse and fine-grained approaches to outperform existing methods.
- NOAH: Neural Optimized A* Search Algorithm for Graph Edit Distance Computation [16] proposes **Noah**, which integrates A* search with Graph Path Networks for approximate GED computation, optimizing search complexity and demonstrating practical effectiveness.
- Learning Efficient Hash Codes for Fast Graph Based Data Similarity Retrieval [13] proposes **HGNN**, combining GNNs with hash learning for efficient graph-based data retrieval, addressing irregular structures and high computational complexity in graph operations.
- More Interpretable Graph Similarity Computation via Maximum Common Subgraph Inference [6] proposes **INFMCS**, an interpretable model inferring Maximum Common Subgraph during graph similarity learning, combining transformers and graph convolution for improved performance.

- H2MN: Graph Similarity Learning with Hierarchical Hypergraph Matching Networks [17] proposes **H2MN**, leveraging hierarchical hypergraph matching for graph similarity learning, employing hyperedge pooling and multi-perspective matching to achieve superior results.

- Efficient Graph Edit Distance Computation using Isomorphic Vertices [5] proposes **Isomorphic Vertex Reduction**, a method to reduce GED computation costs by addressing redundant mappings from isomorphic vertices, aiming for optimization in exact and approximate GED computations.

- Exploring Attention Mechanism for Graph Similarity Learning [12] proposes **NA-GSL**, a unified framework incorporating attention mechanisms for graph similarity estimation, achieving improved performance through graph convolution and cross-graph co-attention.

- Graph Edit Distance Learning via Different Attention [9] proposes **DiffAtt**, a graph-level fusion module leveraging attention mechanisms to compute GED efficiently, outperforming traditional methods in accuracy and speed.

- Graph-Graph Context Dependency Attention for Graph Edit Distance [4] proposes **GED-CDA**, incorporating context dependency attention modules for GED computation, capturing inter- and intra-graph dependencies effectively, and demonstrating high efficiency.

- GREED: A Neural Framework for Learning Graph Distance Functions [11] proposes **GREED**, a siamese GNN model for GED and Subgraph Edit Distance, preserving metric properties and achieving high accuracy and efficiency for large-scale retrieval tasks.

- MATA: Combining Learnable Node Matching with A* Algorithm for Approximate Graph Edit Distance [8] proposes **MATA***, a hybrid model combining GNNs and A* algorithms for approximate GED computation, addressing scalability and efficiency issues with promising results.

- Multilevel Graph Matching Networks for Deep Graph Similarity Learning [7] proposes **MGMN**, a multilevel graph matching network capturing cross-level interactions for graph similarity, demonstrating superior performance and robustness with increasing graph size.

- Wasserstein Graph Distance Based on L1-Approximated Tree Edit Distance Between Weisfeiler-Lehman Subtrees [3] proposes **WWLS**, combining Wasserstein distance with L1-approximated tree edit distance to detect subtle structural differences, showing superior performance in graph classification tasks.

- EUGENE: Explainable Unsupervised Approximation of Graph Edit Distance [2] proposes **EUGENE**, an unsupervised method for GED estimation using algebraic representation and rounding, demonstrating competitive performance and potential for broader applications.

6 Methodology and Experimentation

7 Discussion and Conclusions

References

[1] Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. Simgnn: A neural network approach to fast graph similarity computation. page 384–392, 2019.

[2] Aditya Bommakanti, Harshith Reddy Vonteri, Sayan Ranu, and Panagiotis Karras. Eugene: Explainable unsupervised approximation of graph edit distance. 2024.

[3] Zhongxi Fang, Jianming Huang, Xun Su, and Hiroyuki Kasai. Wasserstein graph distance based on l1–approximated tree edit distance between weisfeiler–lehman subtrees. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(6):7539–7549, Jun. 2023.

[4] Ruiqi Jia, Xianbing Feng, Xiaoqing Lyu, and Zhi Tang. Graph-graph context dependency attention for graph edit distance. pages 1–5, 2023.

[5] Jongik Kim. Efficient graph edit distance computation using isomorphic vertices. *Pattern Recognition Letters*, 168:71–78, 2023.

[6] Z. Lan, B. Hong, Y. Ma, and F. Ma. More interpretable graph similarity computation via maximum common subgraph inference. *IEEE Transactions on Knowledge; Data Engineering*, (01):1–12, apr 2021.

[7] 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, February 2023.

[8] Junfeng Liu, Min Zhou, Shuai Ma, and Lujia Pan. Mata*: Combining learnable node matching with a* algorithm for approximate graph edit distance computation. page 1503–1512, 2023.

[9] Jiaxi Lv, Liang Zhang, Yi Huang, Jiancheng Huang, and Shifeng Chen. Graph edit distance learning via different attention. 2023.

[10] 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.

[11] Rishabh Ranjan, Siddharth Grover, Sourav Medya, Venkatesan Chakaravarthy, Yogish Sabharwal, and Sayan Ranu. Greed: A neural framework for learning graph distance functions. 2023.

[12] Wenhui Tan, Xin Gao, Yiyang Li, Guangqi Wen, Peng Cao, Jinzhu Yang, Weiping Li, and Osmar R. Zaiane. Exploring attention mechanism for graph similarity learning. *Know.-Based Syst.*, 276(C), sep 2023.

[13] Jinbao Wang, Shuo Xu, Feng Zheng, Ke Lu, Jingkuan Song, and Ling Shao. Learning efficient hash codes for fast graph-based data similarity retrieval. *IEEE Transactions on Image Processing*, 30:6321–6334, 2021.

[14] Runzhong Wang, Tianqi Zhang, Tianshu Yu, Junchi Yan, and Xiaokang Yang. Combinatorial Learning of Graph Edit Distance via Dynamic Embedding. *arXiv e-prints*, page arXiv:2011.15039, November 2020.

[15] 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.

[16] Lei Yang and Lei Zou. Noah: Neural-optimized a* search algorithm for graph edit distance computation. pages 576–587, 2021.

[17] Zhen Zhang, Jiajun Bu, Martin Ester, Zhao Li, Chengwei Yao, Zhi Yu, and Can Wang. H2mn: Graph similarity learning with hierarchical hypergraph matching networks. page 2274–2284, 2021.