Learning Attributed Subgraph Matching

Yanxi Lu

A thesis submitted for the degree of
Bachelor of Advanced Computing(Honours) at
The Australian National University

June 2020

Preliminary version – 18 June 2020

Preliminary version – 18 June 2020

Except where otherwise indicated, this thesis is my own original work.

Yanxi Lu
18 June 2020

Acknowledgments

I would like to thank my supervisor, Dr. Qing Wang, for the amazing support she gave me throughout this project. From guiding my research, organizing weekly meetings, to editing my draft thesis, she has helped me more than I could ever imagine. I also wish to thank Asiri and Fangbing for spending their valuable time and effort offering ideas to help me come up with the model design.

Abstract

The attributed subgraph matching problem is the task of establishing a meaningful correspondence of nodes between a query graph and another graph which maximize the node and structure similarity. It is widely applicable in areas such as pattern recognition in computer vision, and biometric information retrieval. Over the past decades, many algorithms have been proposed for this NP-hard problem, including theoretical ones and practical ones. Recently, the rapid development of machine learning techniques has brought a whole class of new methods to this problem. In this work, we give a comprehensive review of existing methods and show that there is still a big gap in current researches. We propose a novel neural network model which attempts to bridge this gap. In this model, we have tried two previously unseen approaches, including modeling graphs as node sequences and use an NLP-inspired method, as well as training on pairs of subgraphs directly. We evaluate the model performance on a synthetic dataset and two famous real-world datasets: Cora and MovieLens. Though the result is still far from optimal, our innovative model has outperformed some existing work. We analyze the reasons and perform further experiments to investigate factors affecting the matching performance, which could help us make the model better.

Contents

Chapter 1

Introduction

Graph is a data structure which is capable of encoding structural information compactly. Over the last decades, it has been widely used in areas such as computer vision, biology, internet and finance. The accelerated growth of these domains has sparked the research interest in graphs. In particular, modern applications are transitioning to use the so-called attributed a graph to store large amount of information and complex structure simultaneous on this data structure.

Research on the problem of graph matching, which refers to establishing a meaningful correspondence of nodes between two or more graphs[Fey et al., 2020], can be dated back to centuries ago. However, the increasing interest of applying it on attributed graphs, especially from the computer vision society has led to a frenzy in related research efforts.

The attributed graph matching problem has been heavily investigated, both theoretically and in practice, using well-established theories and methods such as graph edit distance(GED) and graph kernels. There are still new methods coming out occasionally, to better approximate this NP-hard problem.

After performing extensive research on related methods, we find there is still a big gap in existing methods for the attributed subgraph matching problem. In this thesis, we propose a novel neural network-based method which trains the attribute learning and matching processes jointly.

1.1 Research Objectives

The objective of this research project is investigating the attributed subgraph matching problem, and come up with a way to solve this traditional problem with modern machine learning techniques. To this end, we will:

- Investigate the problem in detail and analyze the difficulty in modelling this problem.
- Develop a machine learning model for attributed graph matching which finds the solution with reasonably high accuracy.
- Perform experiments on a synthetic dataset and real-world data, study how the choice of parameters affect the matching results.

1.2 Contributions

The main contributions of this work are:

- For the attributed subgraph matching problem, we introduce the novel idea of treating graphs into a sequence of data to study the graph structure by investigating the correspondence between points efficiently and effectively. We train on pairs of subgraphs, which has never been used in this domain.
- We attempt to bridge the gap between node-level similarity which is used by most existing work and the final optimal subgraph.

1.3 Thesis Outline

The thesis is organized as followed: In chapter 2, we look at the graph matching problem in detail and review related works. We then explain our model architecture in detail in chapter 3. Chapter 4 presents the model performance on both synthetic and real-world datasets. Finally, we discuss directions for future work in chapter 5.

Chapter 2

Background and Related Work

In this chapter, we introduce the notations and provide definitions necessary for understanding the rest of the thesis. We then discuss the graph matching problem in detail and review existing methods in this domain. We also briefly introduce the graph neural network(GNN), which is a powerful tool that plays an important role in our model design. Before proceeding to different sections, we list the notations that will be used throughout the thesis in Table 2.1.

2.1 Basic Definitions on Graph

A graph G is defined by a tuple (V, E) where V is set of nodes and $E \in V * V$ is the set of edges. We restrict our attention to undirected graphs, where an edge denotes the existence of relationship between the nodes it connects. We use $N(v_i)$ to denote the neighbourhood of a node v_i in G, i.e. $N(v_i) = \{v_j \in G \mid (v_i, v_j) \in E\}$. Graph data is usually represented in matrix form, and perhaps the most commonly used one is the adjacency matrix. Denoted by a function adj in this thesis, it maps G to a $|V| * |V|$ matrix a with binary elements $a_{i,j} = 1$ if $(v_i, v_j) \in E$ and 0 otherwise. A path between two nodes v_i and v_j is a sequence of nodes which starts with v_i and ends with v_j such that there exists an edge between any two subsequent nodes in the sequence. G is catorized as a connected graph if for any two nodes in V, there exists a path between them. We define the shortest distance between two nodes v_i and v_j in G to the length of the shortest path between them. Finally, a graph $g = (V_g, E_g$ is a subgraph of G iff $V_g \subseteq V$ and $E_g \subseteq E$.

2.2 Attributed Graph

While each node in the above definition is uniquely identifiable by an index, modern applications of graphs usually require storing more information. The attributed graph is a more general way to define graphs which are capable of capturing rich information on nodes and edges. The definitions we use here also appear in many other works [Bollobas, 1998] [Livi, 2013]. Let AN be the set of node attributes and AE be the set of edge attributes. An attributed graph G is defined by a tuple (V, E, σ, μ)

where V and E are defined similarly as before. $\sigma : V \rightarrow AN$ is a function which maps each $v_i \in V$ to a set of attribute values in AN. Similarly, $\mu : E \rightarrow AE$ is a function which maps each edge to a set of edge attribute values. In the following work, we will use attributes and features interchangeably. Fig 2.1 gives an illustration of how attributed graphs may be used.

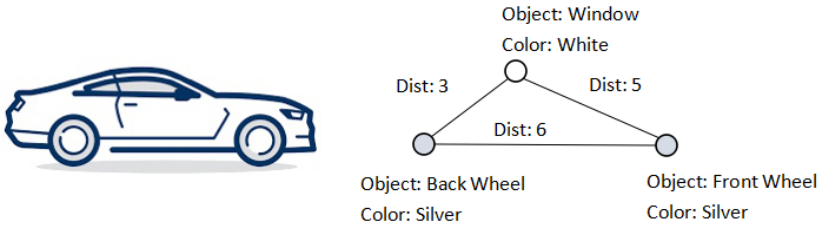

Figure 2.1: Example usage of attributed graph

In image processing tasks, we often extract high-level features from detected objects. In Fig 2.1, we have extracted three features from the car image on the left, namely the window and two wheels. We can then store the information compactly with an attributed graph on the right. For simplicity, we use a fully connected model. The three nodes correspond to the three features. For each node, there are two attributes, include the object identity and color. For each edge, there is an attribute indicating the distance between the two features it connects. The additional information on node and edges makes this graph a useful representation of the original image.

2.3 Attributed Graph Matching

Graph matching is a class of problems which, given two graphs as inputs, aims to find a mapping between all or some nodes in each graph such that, when the nodes are labelled according to such mapping, the two graphs are most similar. [Caetano, 2009] Generally, there are two classes of graph matching problems, depending on the form of node mapping we aim to find.

2.3.1 Exact Matching

The exact graph matching problem is characterized by the fact that the mapping between nodes in the two graphs must be edge-preserving, in other word, the mapping of any two nodes which are edge-connected in the first graph are connected in the second graph. We are normally more interested in the most strict version, where the mapping must be bijective, i.e.$(v1, v2) \in E \; iff \; (f(v1), f(v2)) \in E'$.

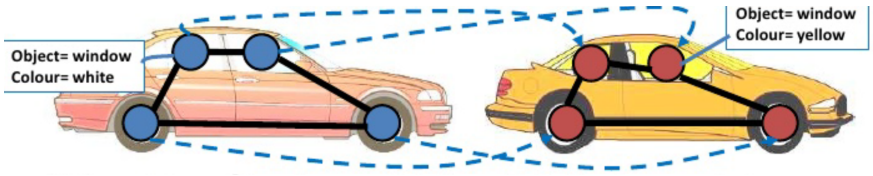

Figure 2.2: Example of matching attributed graphs

Fig 2.2 gives an example of two graphs which satisfies our previous constraint. If we omit the attributes, the dotted lines denote the mapping between nodes from the two graphs representing two cars. This problem is also called the graph isomorphism [Bengoetxea, 2002].

Graph isomorphism on attributed graphs is more complicated since we need to consider the matching of attributes as well. Given two attributed graphs $G = (V, E, \sigma, \mu)$ and $G' = (V', E', \sigma', \mu')$, with $|V| = |V'|$, we want to find a bijection $f : V \mapsto V'$ which satisfy the following constraints [Caetano, 2009]:

- $\sigma(v) = \sigma'(f(v)), \forall v \in V$
- $\forall e_{i,j} \in E$ there exists an edge $e'_{f(i),f(j)} \in E'$ such that $\mu(e_{i,j}) = \mu'(e'_{f(i),f(j)})$
- $\forall e'_{i,j} \in E'$ there exists an edge $e_{f^{-1}(i),f^{-1}(j)} \in E$ such that $\mu(e_{f^{-1}(i),f^{-1}(j)}) = \mu'(e'_{i,j})$

Using the above definition, the two graphs in Fig 2.1 don't exactly match because any node mapping won't satisfy the attribute constraint. This leads to our following discussion about inexact graph matching.

2.3.2 Inexact Matching

In real scenarios, due to external noise or other causes, two graphs may be very similar in both structure and attributes, but with small differences. Under the constraints of exact matching, it is impossible to find a mapping between the two graphs. In real-world applications such as pattern recognition, the procedure often include performing a query with the detected pattern and finding the matching one in an existing pattern database. The two attributed graphs in Fig 2.1 could be an example query graph and stored pattern for a car recognition task. While they do not match exactly due to the small difference in node attributes, they would be considered as a reasonably good match which serves the recognition purpose. Therefore, when matching attributed graphs, it is often more interesting to consider the attribute and structural similarity between them. This need comes from the fact that graphs from the same class may differ in small parts. [Livi, 2013] In general, given two graphs $G = (V, E, \sigma, \mu)$ and $G' = (V', E', \sigma', \mu')$, the class of inexact matching problems wants to find a mapping $f : V \mapsto V'$ which maximizes some matching goodness(graph similarity) measure $sim(G, G')$.

2.3.2.1 Graph Similarity Measure

In general, the function *sim* measuring the similarity between two graphs G and G' should satisfy the following properties:

- **symmetry** $sim(G, G') = sim(G', G)$
- **triangular inequality** $sim(G, G') + sim(G, G'') \geq sim(G', G'')$

There are three mainstream approaches to define this similarity measure[Livi, 2013]. We provide a brief description of each. They are essential in the understanding of the inexact problem and subsequent discussions of existing methods. Also, we will be using some of them in the design of our model.

- **Graph edit distance** The graph edit distance(GED) measure originates from the well-known edit distance for strings. It is a nonnegative measure of dissimilarity between two graphs which computes the total cost to transform one graph into another. With this cost, we can know the amount of structural and attribute distortion required to make the two graphs the same.[Livi, 2013] Consider the situation where we transform attribute graphs G into G', the transformation consists of three elementary operations[Carletti, 2016], including:

  1. node/edge removal which removes a node from V or an edge from E
  2. node/edge insertion which inserts a node into V or an edge into E
  3. node/edge substitution which changes the attributes of a node in V or an edge in E.

  Each elementary operation is associated with a predefined nonnegative cost. A number of such elementary operations applied in a sequence constitutes an edit path. In transform from G to G', we define an edit path as a bijection $f : v_i \to v'_j$ where $v_i \in V, v'_j \in V'$. We define a complete edit path as a path that transforms G into G'. There may be many such paths, the one with the shortest cost is the graph edit distance we are interested in.

- **Graph kernels** Kernel methods are classical machine learning methods which learn by comparing pairs of data points using a particular similarity measure - kernels.[Borgwardt, 2020]. When applied to graphs, it is a useful method to compute graph similarity. We will provide some basic definitions first.
  Let X be a generic input space, and $k : X * X \mapsto R$ be a function. Then, k is a kernel on X if there is a Hilbert space H_k and a feature map $\phi : X \mapsto H_k$ such that $K(x, y) = \langle \phi(x), \phi(y) \rangle$, where $\langle\ ,\ \rangle$ denotes the inner product. We also require k to be symmetric,i.e. $k(G, G') = k(G'G)$ and positive semidefinite. We can obtain the similarity between the data points from the kernel values of the pair.
  Another important concept with kernel method is the Gram Matrix K, Given $x_1, ..., x_n \in X$ and a kernel function k, $K_{i,j}$ stores the pairwise kernel value for

$(i, j \in [1, n])$. K is commonly used in graph kernel methods where the graph-level similarity is estimated by combining local kernel values.
There are many different graph kernels methods to estimate similarity between graphs. We will introduce twos famous one, the shortest-path kernel, to provide some intuitions.

1. The shortest-path kernel[Borgwardt, 2005] is one of the first graph kernels proposed. It uses a kernel function which, for any pairs of nodes in the two graphs, compare their attribute difference and length of the shortest graph distance between them. Let G and G' be the two attributed graphs, and $d(v_i, v_j)$ denote the shortest distance between two nodes in the same graph, then the kernel function is defined as:

$$k_{SP}(G, G') = \Sigma_{(v_i,v_j)\in V^2} \Sigma_{(v'_i,v'_j)\in V'^2} k((v_i, v_j), (v'_i, v'_j)) \tag{2.1}$$

   where $k((v_i, v_j), (v'_i, v'_j)) = k_L(v_i, v_j).k_L(v'_i, v'_j).k_d(d(v_i, v_j), d(v'_i, v'_j).k_L$ is kernel function measuring the attribute similarity between two nodes and K_D is a kernel function which measures the shortest-distance similarity between one pair of nodes from each graph. This is a classical example of first applying a kernel to subgraph structures of graph and then combining the result. These type of graph kernel methods are called the convolutional kernel.
2. [Gartner et al., 2003] and [Kashima et al., 2003] proposed the random walk kernel simultaneously. A random walk in a graph G is, given a starting node v and a pointer originally located at v. For each iteration, move the pointer to one of its neighbours with some probability based on the weight of incident edges. We can derive a node sequence after a set number of iterations. The originally proposed random walk kernel are derived from performing random walks in each graph, and count the number of walks that two graphs have in common. The intuition is that the two graphs are similar if they exhibit similar patterns during random walks. Similarly as in shortest-path kernels, most random walk kernels and convolutional kernels where kernel functions are also defined for substructures such as individual nodes.

- **Graph embedding** The last common approach to measure graph similarity graph embedding. The idea is more recent compared to the previous two approaches. To explain this idea, we first introduce the concept of embedding between two spaces[Livi, 2013].
  Let $(X, d), (Y, e)$ be two metric spaces where $d : X * X \mapsto R$ and $e : Y * Y \mapsto R$ are the distance measure in space X and Y respectively, X is embeddable into Y if there exists a function $\phi : X \mapsto Y$ such that $d(x, y) = e(\phi(x), \phi(y)).\forall x, y \in X$
  Generally speaking, graph embedding allows associating the generic graph space X which in the context of graph matching, the space of graphs to a known space such as the euclidean space, where known methods and established al-

gorithms can be readily applied. Based on the definition of the embedding, the distances measures in the two spaces are positively correlated. Therefore, we can estimate the similarity in the original graph space using the distance measure in our embedded known space.
There are several ways to compute the embedding function. Here, we will explain some of them. GED embedding[Riesen and Bunke, 2010] uses graph edit distance as a basic dissimilarity measure and maps each graph in the graph space into a P-d vector, where each vector element is the graph edit distance between the input graph and a selected distinct graph in the same graph space. The set of P selected graphs are called the prototype set. This approach, though said to be applicable to any labeled graph, suffers from the difficulty in selecting the prototype set P. [Robles-Kelly and Hancock, 2005] proposed two seriation based embedding methods. It consists of finding an ordering of the nodes and form a sequence. The sequences can then be processed using methods such as string kernels. Note that these methods currently only apply to graphs where node attributes do not affect the seriation. Lastly, modern neural network approaches such as the graph neural network(GNN)[Scarselli, 2008] can obtain graph embedding through learning the individual node embeddings. Extending on this, the graph matching network model[Li, 2019] learns the embeddings of the two input graphs jointly using a cross-graph attention mechanism.

To summarize, we have looked at the three general approaches which can be applied to generating a similarity measure for most graphs. The graph edit distance and graph kernel approaches have been widely used in many inexact graph matching tasks. Graph embedding, in particular the ones based on machine learning have also attracted attention recently.

2.3.3 Attributed Subgraph Matching

In this thesis, we focus on one instance of the inexact matching problems, the attributed subgraph matching problem. Given two graphs $G = (V, E, \sigma, \mu)$ and $G' = (V', E', \sigma', \mu')$ with $|V| \leq |V'|$, an intuition of this problem is to find an injective mapping $f : V \mapsto V'$ which yields a better matching result than any other mapping. Note that the range of the mapping is a subset of V', hence forms a subgraph of G'. Generally speaking, we want the subgraph found to maximize some matching goodness measure. We will give a formal problem definition in the next chapter. The problem is particularly interesting and widely applicable in areas such as recommender systems and web information retrieval. Take an example of a movie recommender system, one of its main purposes is to recommend movies to users based on user attributes such as demographics, viewing history and habits. In addition, the system usually needs to take into account the relationship between different users such as friendship and liked comments. A common way for such systems to model user is using attributed graphs to model users as nodes and relationships between users as edges. When performing recommendations to a new user with limited information, it is impossible to obtain an exact matching in the system's existing database. In-

stead, the recommender system can look for the most similar user stored and make recommendations based on the obtained information. This recommendation process is an instance of the attributed subgraph matching problem as it involves searching for a subgraph of a large graph which best matches the query graph.

2.4 Existing Methods

The inexact graph matching problems on attributed graphs are known to be NP-complete.[Bengoetxea, 2002] The attributed subgraph matching problem is even harder because it involves searching for an optimal mapping for a subset of nodes. Therefore, while it is possible to enumerate all possible mappings for nodes across the input graphs,it is computationally prohibitive.

There is a long history of research on methods for graph matching problems. While most early works are focused on exact matching problems. [Tian, 2008], recently there is an increase in interest on the inexact attributed subgraph matching problem due to its expanding applicability. We will discuss two main approaches adopted by researches on the attributed subgraph matching problem. In all occasions, these approaches rely on first coming up with a similarity measure, for example the ones discussed before for evaluating the matching quality, then find the optimal node mapping which maximizes this similarity measure.

2.4.1 Index-based Methods

The first approach is index-based. It involves a two-step process: Indexing some graph entity(node, edge, subgraph or their hybrid) within each graph, then finding the best matching subgraph according to some graph similarity measure using the indices. [Shearer et al., 2000] tackles the problem in the context of image processing, he indexes the objects detected in the image based on their spatial relationships and uses the largest common subgraph as a graph similarity measure. [Tian, 2008] proposed TALE which uses a Neighbourhood Index to incorporate neighbourhood information into node indices. For each node, its neighbourhood index captures the local structural information including the size of immediate neighbourhood and connectivity among neighbours. For the matching process, TALE assumes that the overall matching quality is maximized if the indices of individual nodes are best-matched. Hence, it adopts a greedy approach which finds the matching nodes sequentially.[Zhang, 2009] improves upon TALE and considers large neighbourhoods. For each node, its neighbourhood consists of all nodes within a predefined graph distance. The method then generates indices for every pair of nodes using the information about the intersection of their neighbourhood. It also uses a node-wise matching paradigm to find the matching nodes sequentially. [Zhu, 2010] proposes incorporating both attribute and structural information in indexing. The method cluster highly-connected nodes with similar attributes using some predefined cluster connectivity and attribute similarity measure. This methods focus more on the importance of similarity in structure on the overall matching quality and matches edges sequentially. The main difficulty

with index-based methods is the necessity to handcraft an indexing method. There lacks a universally acknowledged indexing method.Therefore, these methods may not generalize well when applied on different instances of the attributed subgraph matching problem. Furthermore, the greedy approach adopted by these methods is susceptible to incorrect starting points. The second problem is dependent on the first one, hence the effectiveness of index-based methods relies on choosing a good indexing method.

2.4.2 Substructure Similarity-based Methods

Methods along this approach usually have a two-step process[Jouili, 2009]. Firstly, they construct a graph similarity measure. Then they compute a pair-wise distance(similarity) matrix between graph substructures such as node, edge, subgraph across the input graphs. The selected substructure is closely related to the graph similarity measure. As discussed before, the attributed subgraph matching problem is NP-hard. The use of substructure information is usually for selecting candidate matching subgraphs since exhaustively searching the space of all possible matching subgraphs is prohibitive. Most existing work is developed along this direction. To obtain the substructure similarity, some methods use handcraft similarity measures which rely on expert knowledge, while others such as machine learning techniques learn the similarity automatically.

2.4.2.1 Handcraft Substructure Similarity

Jouili [2009] proposes a Local description(LG) architecture which models the graphs as a collection of local information for each node. The method generates a node signature by encoding local features for each node. The features include the node attributes, degree and attributes on the incident edges to the node. The model then computes the distance between each pair of nodes across the input graphs from the similarity between their node signatures. They model the matching step as an assignment problem and uses the Hungarian method to solve it. Zhang [2016] proposes a family of algorithms called FINAL utilizing the alignment(matching) consistency principle, which determines the node similarity by stating that the for each pair of matching nodes across graphs, their close neighbour with similar attributes have a high chance to be matched as well. After computing the similarity matrix(alignment matrix) in the context, the problem is formatted as a quadratic assignment problem and approximation methods are used to estimate the optimal assignment.[Du, 2017], they proposed FIRST algorithm computes the node pairwise similarity by combining the principle of the two methods discussed before.

While these similarity measures are sound and often innovative, they are designed for specific types of graphs and may have limited applicability in general.

2.4.2.2 Learned Substructure Similarity

The recent development in machine learning methods provides alternative approaches to a lot of complicated tasks, including the attributed subgraph matching problem. In Comparison to using handcraft node-similarity measures which use expert knowledge, machine learning methods learns the similarity measure from the data. Here, we introduce several machine learning methods closely related to the problem.

Inference Model [Khan et al., 2013] models the attributed subgraph matching problem as an inference model in graphical models. The model iteratively computes an inference cost for each query node and its candidate nodes. The optimal inference cost, denoting the dissimilarity between nodes across graphs are updated each iteration. The algorithm terminates when reaching a stationary point and the optimal matching for every node in the query graph has been found.

Graph Kernel As discussed before, graphs kernels are commonly used ways to measure graph similarity. Some are applicable in solving the attributed subgraph matching problem as well. Kriege [2012] proposes the subgraph matching kernel which considers all possible bijections between all subgraphs constituting at most n nodes, where n is dependent on the size of substructure considered. For each bijection, a kernel value is evaluated based on the similarity in attributes of constituent nodes and edges. While the author did not explicitly state the use of subgraph matching kernel in the attributed subgraph matching problem, one way to extend it could be setting n to be equal to the size of intended matching subgraph and find the mapping with the highest kernel value. [Frohlich et al., 2005] proposes the optimal assignment kernel. When applied in this context, it involves checking all possible mappings between components(such as nodes) of the two input graphs and find the one which maximizes the sum of the kernel value for pairs of components. The component kernel value could be derived using node embedding or other node similarity measures. This kernel is substructure-similarity based because the derived optimal assignment is dependent on the kernel value of components.

Though graph kernels are widely used to measure graph similarity, it can be observed that those applicable to the attributed subgraph matching problem, for example, the two methods described, both involve some tedious enumeration processes.

Neural Network Approaches Recently, neural networks have been gaining popularity in many areas. A neural network of a certain complexity is proven to be capable of approximate any function. Therefore, it certainly can approximate the substructure similarity measure in the attributed subgraph matching problem. [Subramaniam, 2016] propose a neural network model which uses a correlation layer to learn about the similarity between every pair of nodes across the two input graphs. They then use convolution neural networks to aggregate the node level correlation information and obtain the matching result. Recently, the GMNwVGG model proposed by Zanfir [2018] uses deep learning to train feature learning and structure matching jointly. This model learns an edge similarity matrix(denoted as the affinity matrix in the text), where the similarity between every pair of edges across the two input graphs are computed. The model then approximates the leading eigenvector

for the matrix using power iterations, implemented with multiple feedforward layers. The eigenvector is then used to compute a confidence matrix which records the pairwise confidence score which denotes the matching confidence between nodes. Finally, the algorithm uses a special voting layer which uses a softmax-based method to return the matching result.

All these recent approaches have achieved remarkable results for the attributed subgraph matching problem. However, most neural network approaches try to dissect the problem into matching the substructures. The matching process is prone to finding false correspondences which are only similar locally.[Fey et al., 2020] It is hard to reason about questions such as how exactly are substructures put together to constitute the optimal matching subgraph. Moreover, we feel that current neural network methods do not sufficiently exploit the rich structure of graphs.

2.4.3 Summary

In this chapter, we have reviewed existing methods for the attributed subgraph matching problem and analyzed their pros and cons. We feel that there is still a large gap, particularly in neural network-based methods. In this work, we want to come up with a model that processes the graphs and performs the matching intuitively. We will be using graph neural network(GNN)Scarselli [2008], which is a class of neural network models specialized in processing graphs. Surprisingly, except [Baskararaja and Manickavasagam, 2012] which only targets the exact subgraph matching problem. we cannot find any existing work applying GNN for the attributed subgraph matching problem.

2.5 Graph Neural Network

Graph Neural Network (GNN), as suggested by its name, is a class of machine learning models dedicated to processing graphs. Models in this class find a mapping $f : X \mapsto R^m$, where X denotes the original graph space. In another word, it transforms graphs in the generic graph space into known euclidean space representations. The results are m-dimensional vector representations for each node, called node embeddings. GNN includes a rich family of models, in this section, we focus on the original message passing architecture.

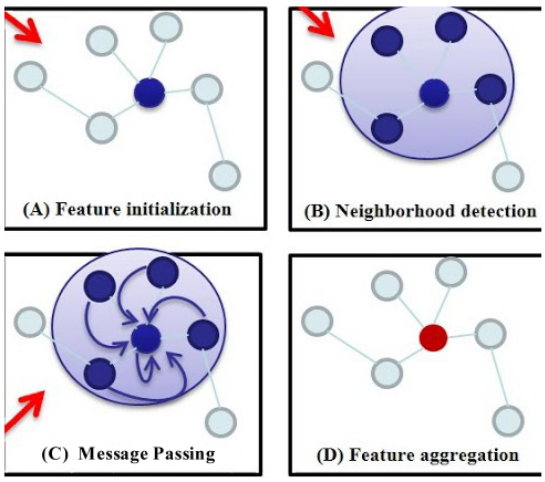

Figure 2.3: An illustration of 1-hop) graph neural network. The model works based on the principle of message passing.

The original formalization of GNN by Scarselli [2008] uses message passing to learn neighbourhood information for every node. Fig 2.3 shows the standard 4-stage process for a GNN. The model takes in a graph $G = (V, E, \sigma, \mu)$ and its adjacency matrix A as input. In stage A, we initialize the node and edge features(attributes) using the functions σ and μ. We then find the immediate neighbours for each node in stage B from the adjacency matrix. Stage C shows the message passing process for the central node. Each arrow indicates one message from one neighbour to the central node. Each message contains information about the neighbour and the edge. In the last stage, the central node aggregates messages it just received and updates its information. The last two stages together are often referred to as a message propagation step. We require the step to be performed on every node in parallel. Usually, in real graph processing tasks, we want to repeat the message passing multiple times so that we can capture information for larger neighbourhoods. We use the name N-hop GNN to denote performing the propagation step N times. In an N-hop GNN, each node can receive information all nodes within distance k. To record the updated node information after each propagation, we use m-dimensional hidden vectors denoted by h_i^t, where $i \in \{1, ..., |V|\}$ and $t \in \{0, ..., k\}$. h_i^t records the state of node v_i after the tth propagation. h_i^0 is the initial vector representation of v_i. Depending on the task and G, the initial representation may be the raw or preprocessed node feature vectors. h_i^k is the node embedding for v_i after the last propagation step, which is also the model output for the node. Throughout the propagation iterations, the dimension of h_i^t is constant. After obtaining the final node embeddings, we can optionally obtain a graph-level embedding H by aggregating them.

There are many other GNN models such as Graph Convolution Network(GCN) [Kipf, 2016] which performs graph convolution at each step, Gated Graph Neural Networks [Li et al., 2015] which uses gate mechanisms such as GRU[Chung, 2014] or LSTM[Gers, 1999] in the propagation step to better utilize the history of hidden vec-

tors for nodes.

2.5.1 Analysis of GNN

The discussed GNN model captured both feature and structural information of graphs, its affinity towards the data structure makes it a natural choice for graph learning. It has achieved great success in graph-related tasks such as image classification[Garcia, 2017] and protein interface prediction[Fout]. We feel that it is possible to extend it to the attributed subgraph matching problem. To make the original GNN model applicable, we have to note some limitations when using it to solve our problem.

One limitation is the lack of intuition in structural match. Though GNN captures some structural information on node embeddings by considering neighbourhood information, simply finding the matching nodes based on similarity in embedding does not explicitly constrain structural similarity with the query graph.

The other limitation comes from its propagation step. For an N-hop GNN, each node receives information from its neighbours N times. This iterative update process for fixed nodes may cause the distribution of node embeddings to be smoothed and become less informative for distinguishing each node [Bengoetxea, 2002]. Therefore, when using the node embeddings to find the best matches, there will be more close candidates due to the smoothed embedding distribution, hence increasing the difficulty to find the optimal match.

2.6 Chapter Summary

In this chapter, we introduced the concept of attributed graphs and discussed the exact and inexact graph matching problems. We reviewed in detail the graph similarity measures for inexact matching problems and introduced the mainstream approaches for the attributed subgraph matching problem. Finally, we described and analyzed the message passing graph neural network model, which forms a key component in our design.

Table 2.1: notations used in this thesis

Notation	**Description**
R^m	m-dimensional euclidean space
$G = (V, E, \sigma, \mu)$	an attributed graph
$G' = (V', E', \sigma', \mu')$	another attributed graph
$V = (v_i)$	the set of nodes in G $(i = 1, ..., n)$
$V' = (v'_i)$	the set of nodes in G' $(i = 1, ..., n')$
$\mid . \mid$	the size of a discrete set
n	$\|V\|$
n'	$\|V'\|$
$E = \{(v_i, v_j)\}$	the set of edges in G
$E' = \{(v'_i, v'_j)\}$	the set of edges in G'
$N(v_i)$	the neighbourhood of v_i in G
$N'(v'_i)$	the neighbourhood of v'_i in G'
adj	a function which returns the adjacency matrix of a graph
AN	the set of node attributes
AE	the set of edge attributes
$\sigma : V \mapsto AN$	a function which maps each node to a set of node attribute values
$\mu : E \mapsto AE$	a function which maps each edge to a set of edge attribute values
$d : V * V \mapsto R$	measure the shortest distance between two nodes in a graph, the function assumes the graph definition is known
$g = (V_g, E_g, \sigma_g, \mu_g)$	a subgraph of G
$g' = (V_{g'}, E_{g'}, \sigma_{g'}, \mu_{g'})$	the ground truth matching subgraph of g in G'
g''	the found matching subgraph of g in G'
k	the graph kernel function
X	a generic space
N	the number of propagation step in a GNN
h_i^t	the hidden representation of node v_i at timestep t $(t = 1, ..., k)$
H_i	the node embedding for node i
e	the dimension of a node embedding
$H[g]$	the concatenation of node embeddings in graph g
$H(g)$	the graph embedding for graph g
sim	the graph similarity measure, return a similarity score $\in$ [0,1] for two input graphs
$M = \{m_{i,j}\}$	the $n * n'$ node similarity matrix$(i = 1, ..., n), (j = 1, ..., n')$, measures the pairwise node-similarity for nodes in G and G'
$[.\|.]$	concatenation of two matrices along the first dimension
L_e	the embedding loss
L_m	the matching loss
D	the node pairwise distance measure for a graph
λ	a hyperparameter in the loss function

Chapter 3

Methodology

This chapter presents the design of our model to solve the attributed subgraph matching problem and the rationale behind this design.

3.1 Problem Formulation

We have briefly introduced the attributed subgraph matching problem. In this section, we give a formal definition for it. We first define a subgraph in the context of attributed graphs. Let $G = (V, E, \sigma, \mu)$ and $g = (V_g, E_g, \sigma_g, \mu_g)$ be two undirected attributed graphs, g is a subgraph of G iff the following constraints are satisfied:

- $V_g \subseteq V$
- $E_g \subseteq E$
- $\sigma_g(v_i) = \sigma(v_i),\ \forall v_i \in V_g$
- $\mu_g(v_i, v_j) = \mu(v_i, v_j),\ \forall (v_i, v_j) \in E_g$

Let $G = (V, E, \sigma, \mu)$ and $G' = (V', E', \sigma', \mu')$ be two attributed graphs. The attributed subgraph matching problem is to, given a subgraph $g \subseteq G$ where $g = (V_g, E_g, \sigma_g, \mu_g)$, find a subgraph $g' \subseteq G'$ where $g' = (V'_g, E'_g, \sigma'_g, \mu'_g)$ such that $|V_g| = |V'_g|$ and $sim(g'', g) < sim(g', g)$ for any other subgraph $g'' = (V''_g, E''_g, \sigma''_g, \mu''_g) \subseteq G'$ with $|V''_g| = |V_g|$. We denote $|V_g|$ as n_g.sim is a the graph similarity measure which, given two graphs g and g', gives their similarity score.. It should satisfy the following properties:

- **symmetry** $sim(g, g') = sim(g', g)$
- **triangular inequality** $sim(g, g') + sim(g, g'') \geq sim(g', g'')$

We restrain g and g' to be connected because connected graphs are usually more meaningful in real-life tasks. Furthermore, to simplify the problem we set the set of edge attributes to **empty**, only the existence of edges is considered.

Graph Similarity Measure. For the attributed subgraph matching problem, we aim to maximize the graph similarity measure between the g' and g''. In this context,

we use it as a guideline to evaluate the performance of our model. In chapter 2, we have reviewed three popular graph similarity measures, we will be utilizing the graph embedding-based measure in designing the objective function in our model. It is worth noting that an objective function in neural network methods is a measure which we aim to minimize. Therefore, we use the counterpart of graph similarity measure - the graph dissimilarity measure. A similarity measure can be easily transformed into an equivalent dissimilarity measure, and vice versa.

3.2 Model Overview

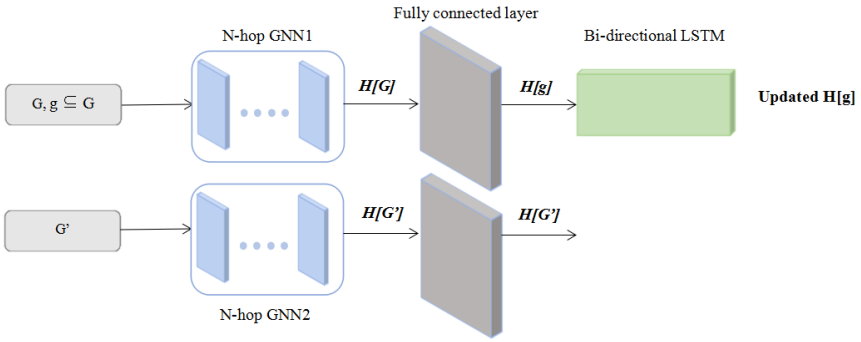

Figure 3.1: Overview of the model. Bold font denotes the output of a previous layer, italic font denotes input to the next layer. A bold and italic font denote both. For separate layers,i.e. the two GNNs, use different names. The fully connected layer is shared.

Fig 3.1 gives an overview of our proposed model. In general, it takes in the graphs G and G', as well as the subgraph $g \subseteq G$ to be matched as inputs. The input also includes the adjacency matrix for each graph. The model first uses separate GNNs to learn about the attributes for both G and G'. With the node embeddings $H[G]$ and $H[G']$. The model uses a matching process which involves putting $H[g]$ into a bi-directional LSTM[Sak et al., 1953]. We use the output of the bidirectional LSTM to obtain an updated $H[G]$ and use this, as well as $H[G']$ to find the optimal match.

The overview gives a general picture of our model. Our training stage and matching stages use slightly different versions from this one.

3.3 Training Stage Model Architecture

We discuss in detail the methodology for the training stage of our model. For this stage, we use ground truth matching subgraph pairs $g \subseteq G$ and $g' \subseteq G'$ as inputs. For the attributed subgraph matching problem, we thought it would be a natural choice to use such subgraph pairs as training data. However, our research only finds [Caetano, 2009] which uses a similar form of data. However, their method is not

targeted at matching subgraphs and uses pairs of whole graphs. For our g and g', we assume that the nodes are organized in sequences which makes the matching consistent, i.e. matching nodes appear at the same slot.

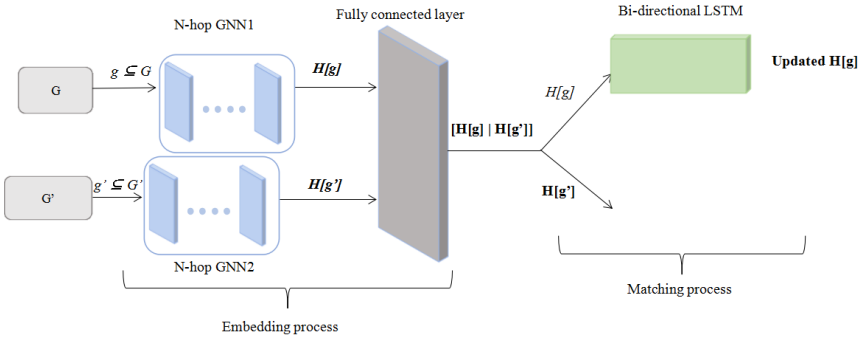

Figure 3.2: Overview of the training stage model. The notation format is the same as Fig 3.1

As shown in Fig 3.2, the training model has two components, an embedding process which learns node attributes and a matching process to find the optimal match for g.

3.3.1 Embedding Process

The embedding process consists of two steps: using two separate GNNs to learn the node features in g and g' separately, then aggregate the results.

3.3.1.1 GNN for Attributed Learning

To learn the features of each training pair, we use separate GNNs for each input. We do this because the inputs belong to separate graphs and GNNs are designed to learn information about a single graph. We define the two GNNs similarly, each containing the following steps:

- **Attribute initialization** Given an input graph $G = (V, E, \sigma)$, the first step is to initialize the attributes. We obtain the attribute of each node using the function σ and construct a $n * d$ matrix, where $n = |V|$ and d is the dimension of attribute vector for each node. Depending on the task, we may want the dimension of node embeddings e to be different from d(for example, when the dimension of attribute vector is too large). To do this, we can apply additional functions on the raw attributes. The output of this step are the initial hidden node representations h_i^0. $\forall v_i \in V$.

$$h_i^0 = f_1(\sigma(v_i)) \quad \forall v_i \in V \tag{3.1}$$

Formula 3.1 gives a formal definition of this step. We use a single fully connected layer with input dimension d and output dimension e. The output of this step is a $n * e$ matrix which records the initial hidden representation of all nodes in V.

- **Propagation Layer** For a N-hop GNN, there are N propagation layers. The input to each layer in our setting is a $n * e$ matrix representing the node hidden representations from the previous iteration. Each propagation layer performs one round of message passing and map h_i^t to h_i^{t+1} $\forall v_i \in V$.

$$h_i^{t+1} = f_a(h_i^t, \Sigma_j(f_m(h_i^t, h_j^t))) \quad \forall i \in V, (i,j) \in E \tag{3.2}$$

Formula 3.2 defines this procedure. f_m is a function which takes in the current hidden representations of a pair of edge-connected nodes (v_i, v_j) and computes the message passed from v_j to v_i in the current propagation iteration. Again, we use a fully connected layer whose output dimension is e, the embedding dimension. The input hidden representations are concatenated to form a $2 * e$ dimensional vector, therefore, the input dimension to f_m is $2 * e$. We then aggregate the messages using a simple summation. There are other operators such as mean, max or the attention-based weight sum[Velickovic et al., 2017]. f_a is a function which generates a new hidden representation for v_i, using its previous hidden representation and the summation of all messages it receives. We use the gated recurrent unit(GRU)[Li et al., 2015] for this. GRU is a memory-incorporated architecture which can store the node hidden representations across different iterations. It mitigates GNN's embedding smoothing scenario noted in chapter 2. Intuitively, it is a good fit for this problem for being capable of capturing some long-term dependencies across nodes. The output of a propagation layer is the updated hidden representation matrix of shape $n * e$.

- **Output** After all N propagation iterations are finished, we obtain $\{h_i^n\}$ the final node embedding for v_i. The output is a $n * e$ matrix where each row represents the learned embedding for the corresponding node.

3.3.1.2 Combine the Embeddings

After obtaining the node embeddings $H[g]$ and $H[g']$ for the pair of input subgraphs, we join the embedding outputs from the two GNNs. The reason is that the purpose of the model is to perform matching across two graphs. Having separate embeddings does not help understand how they are matched. Formula 3.3 describes this step.

$$out = f([H[g] \mid H[g']]) \tag{3.3}$$

We use a single fully connected layer for f. Denote $|V_g| = n_g$, the inputs to this layer are two $n * e$ node embeddings matrices. We concatenating the inputs along the first

dimension to form a $2n_g * e$ matrix. The output dimension of the layer is the same as the input. This is the final output of the attribute learning section.

3.3.1.3 Embedding Loss

In the output $2n * e$ matrix, the first n rows are the embedding $H[g]$ of g and the rest are $E[g']$ for g'. We define the embedding loss L_e for the training pair.

$$L_e(g, g') = \Sigma_{i \in [1, n_g]} ||H_i, H'_i||_2 \tag{3.4}$$

In formula 3.4, we compute the difference between the node embedding matrices, then find the total loss by computing the sum of the norm of each row. This loss function is design along the graph embedding-based approach, though we use node embeddings, the embedding matrix can be used to readily generate graph embedding. In fact, some literature such as [Li, 2019] uses concatenation of node embeddings as graph embeddings.

3.3.2 Structure Matching

After obtaining the node embeddings of g and g', the next process is using these embeddings to finding the matching subgraph. The second part of the model takes in only $H[g]$, the embedding of g and attempts to find its best matching subgraph g'' from G'. From our discussion in chapter 2, though GNN is capable of capturing some degree of structural information, it is not enough for good matches. We will study the result that directly finding the nearest neighbour for $v_i \in V_g$ from G' does not give a satisfactory result.

The inspiration for the matching process comes from approaches in natural language processing(NLP). Sentences in languages are a type of sequential data. Such data takes the form of ordered data points. Data points are highly correlated in these scenarios and looking at them independently will lose information about the context. In meaningful sentences, the words are closely correlated. These correlations may come from the relationship between their meanings and the context of the sentence. Moreover, such correlation may appear between words which are very close or distant away. Analyzing the correlations can help us understand the structure and meaning of the sentence better. In graph matching, it is unintuitive how graphs can be sequential because of the lack of natural order between nodes. However, similar to words in sentences, nodes in attributed graphs are often meaningfully connected. Consider the modern usage of attributed graphs discussed in chapter 1, nodes usually represent entities, they are connected only if there is some relationship between entities. For example, in a graphical representation of the car components, we may have nodes representing the engine, axle and wheels. Obviously, the engine and axle are connected and so for axle and wheels. Moreover, though the engine may not be directly connected to the wheels, they are correlated and there should be a path between them. These observations give rise to the possibility of modelling graphs as sequential data. If this is successful, we can use the rich resort of methods for

sequential data, such as those used for NLP to learn about the correlation between nodes. In our problem, we are interested in undirected graphs, hence there is no restriction on the node sequence. We can exhaustively consider dependencies between every pair of nodes in both directions. Depending on the distance between nodes, we need to consider short-term or long-term correlations. Luckily, there is a well-established method, the Long Short-Term Memory(LSTM) model[Sak et al., 1953], which satisfy all the requirements. It uses memory units with gating mechanism which can control things to read from, add to or delete from the memory, hence provides a flexible framework capable of at capturing both long-term and short-term dependencies. The LSTM model has been proven effective in many sequential data processing tasks.[Murugan, 2018][Le et al., 2019]

The input to the matching process is the embedding $H[g]$ of shape $n * e$. We now serialize the nodes. As discussed before, the order of serialization does not matter, but for each pair of nodes, we need to consider their correlation from both directions. The problem with ordinary LSTM is that it only considers the dependencies of later data points on the previous ones. Luckily, bi-directional LSTM(bi-LSTM) provides a suitable solution for the problem. The intuition with bi-LSTM is shown in Fig 4.2.

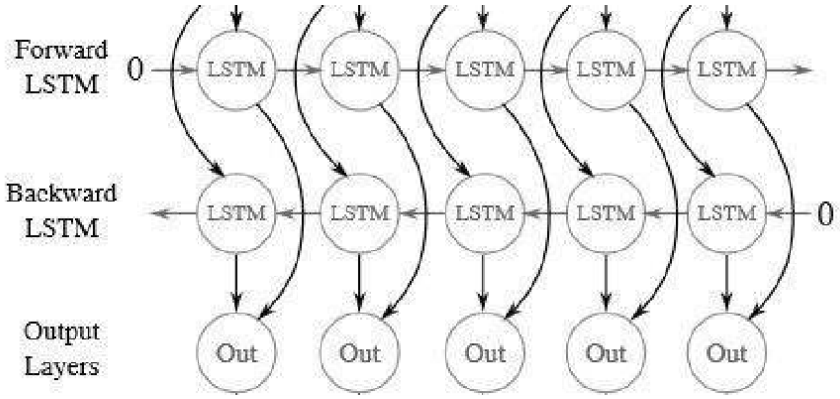

Figure 3.3: 1 layer Bi-directional LSTM used in the model.

Fig 4 shows the unfolding of operations of the same LSTM unit at different time. The arrows coming in from the top are the input sequential data points and the number of timesteps in this setting is equal to n_g, the number of nodes in g. A bi-LSTM comprises a forward LSTM which runs on the input sequence and a backward LSTM which runs on the reverse of the sequence. For any node sequence, the forward LSTM captures dependencies from later nodes on early nodes, while the backward LSTM deals with the other direction. With this architecture, we can learn all the correlation using any node sequence of V_g. The number of output units of the bi-LSTM is set to be e, the same as the node embeddings. Hence, the output of the model is a $n_g * 2e$ matrix(omitting the batch size). We reshape the matrix to $n_g * 2 * e$ $n_g * e$ to separate the forward and backward results. We then take the average along the second dimension to get the average for each node and obtain a new $n_g * 2e$ embedding matrix as output.

3.3.2.1 Matching Loss

For each output produced, we use formula 5.6 to find the top matching subgraph g'' in G'.

$$g'' = \{n' \in \ | \ \forall i \in g_k.\ argmin(d(n', O(i)))\} \tag{3.5}$$

g'' constitutes the nodes in G' which are the closest to each node embeddings to $H[G]$. In the case where there are duplicates, the next closest node is selected instead.

We can then compute the matching loss between the ground truth g' and found g''. The loss function is defined in formula

$$L_m = \Sigma_{i \in [1, n'_g]} ||H'_i, H''_i||_2 + \ \lambda(|(D_{g''} - D_{g'})|) \tag{3.6}$$

In the loss function, the first term defines similarity as the embedding loss. It accounts for the difference in the node embeddings between g' and g''. In the second term, D is the pairwise distance matrix for a graph. Since g' has n_g nodes, both $D_{g'}$ and $D_{g''}$ are n_g*n_g, where position (i,j) denotes the distance between the ith and jth nodes in the graph. Hence, the second term computes the sum of difference of distances between all pairs of nodes across the graphs. Intuitively, for the structure of two graphs to match, this sum should be minimized. λ is a hyperparameter between 0 and 1, we will investigate its impact on the model performance in detail in the next chapter. The overall loss function for each training pair g and g' is hence $L_e + L_m$.

3.4 Testing Stage Model Architecture

The general architecture for testing out model is similar to the model overview. Recall our problem definition is given G, G' and a subgraph $g \subseteq G$, find the best matching subgraph $g' \subseteq G'$. Therefore, the general usage of our model takes in only one subgraph. Similarly, as in the training stage, we pass the three inputs through the GNN and fully connected layer to obtain $H[G]$, $H[G']$ and $H[g]$ the node embedding for the three inputs. H[g] is then fed into the bi-LSTM. The weights of the entire model are already trained, so we would get a good node embedding matrix for g. The best matching is found using formula 3.5.

3.5 Chapter Summary

This chapter gives a formulation of the attributed subgraph matching problem. We then provide a detailed description of our model architecture. In the next chapter, we proceed to test and evaluate our model on both synthetic and real-world datasets.

Chapter 4

Experiments

This chapter describes the implementation detail of our model. We conduct experiments on three datasets and compare the performance measure of our model with an architecture proposed by Jouili [2009] which approximates node similarity using local descriptions of nodes, the optimal assignment(OA) kernel proposed by[Frohlich et al., 2005] and an index-based method TALE proposed by [Tian, 2008]. These methods may not the latest but cover a wide background and serve a good comparison for our new method. We also study how different settings impact the matching results. We evaluate the following research questions: 1.How our proposed model matches subgraphs of different sizes and feature complexity 2.How the choice of hyperparameters, particularly λ in the matching loss function, affect the matching result.

4.1 Datasets

We validate the performance of our model on a synthetic dataset and two real datasets: Cora and MovieLens.

4.1.1 Synthetic Dataset

The attributed subgraph matching problem of interest takes in pairs of attributed graphs G and G' as inputs whereas out model uses pairs of matching subgraphs as input. The synthetic dataset is cleverly constructed to meet both demands. The generation process is defined in Fig 4.1

```
1  def dataset_generation(n, k, f):
2      ""n: number of pair of subgraphs to be included
3        k: number of nodes in each subgraph
4        f: map <fname:domain>, for node features and their domain""
5
6      eps = 0.1         //probability of making changes to attributes and structure
7      modn_cap = 20%    //maximum ratio of modifications can be done
8      G = G' = empty
9      generate a node with random features, add to G and G'
10
11     for i from 1 to n:
12         select a random node n from G //it corresponds to n' in G'
13         generate a random connected g with k nodes containing n
14         randomly set node features for g except node n
15         g' = g.copy
16
17         // alter the graph with some small possibility
18         for node in g':
19             for feature in node:
20                 the feature value changes with a probability eps
21
22         for any two nodes in g':
23             an edge is added/deleted with a probability eps
24
25         // if no modification was made or the number of modifications is too
26         // large, reproduce g'.
27         if g == g' or g' is not connected or difference(g, g') > mod_cap:
28             regenerate g'
29
30         // the other k-1 nodes in g are connected to n in G,
31         // the other k-1 nodes in g' are connected to n' in G'
32         put g into G, g' into G'
33
```

Figure 4.1: Pseudocode for generating the synthetic dataset

Fig 4.1 shows the algorithm for dataset generation. The generator takes in 3 parameters. n is the number of pairs of subgraphs to be generated, k is the number of nodes in each subgraph, and f stores the node features and their domain.

We start with G and G' both empty, then perform a recurrent process of two steps:
Step 1: Generate one pair of matching subgraphs (g, g')
Step 2: Aggregate g into G, g' into G'

4.1.1.1 Pair Generation

Line 12 to 14 in Fig 4.1 generates a random connected graph g of k nodes. Using g as the reference graph, line 15 to 23 then construct a matching subgraph g' by first duplicating g. Small modifications are then done on g' to modify the node features and connectivity with a small probability. Either when $g' = g$, or the number of modifications on g' from g is too small or too large, g' is regenerated.For this experiment, I generated 200 pairs of matching subgraphs with 3, 4 and 5 nodes respectively. Each node has one feature Color with domain "R","G","B". In addition, each node has a unique node id which is shared between the matching nodes across G and G'. However, the id must not be visible to the model. The probabilities for either edge or features being edited are set to be 0.1. The modification cap is 20 per cent.

4.1.1.2 Graph Aggregation

Note that before we generate a pair of subgraphs, a random existing node is selected from G and reused in the generation of g and g'. This ensures that we can put the generated subgraphs into the large graphs G and G'. I generated 200 pairs of matching subgraphs of 3, 4 and 5 nodes each.

4.1.2 Cora Dataset

Cora is a well-known node classification dataset. It contains 2708 machine learning papers each classified into one of seven categories. The dataset uses a dictionary of 1433 unique keywords as paper features. Hence, each paper can be viewed as a node represented by its id and 1433 binary values each denoting the presence of corresponding keywords. The dataset also records the citation relation among papers, which forms the edges. Note that our model works on undirected graphs, therefore, though the citation relationship is directed, we consider two papers related if one cites another. To get the pairs of subgraphs, we need to extract from the original dataset. In this case, we do not have two large graphs. However, we can duplicate the datasets for our large graphs G and G'. This won't be lead to the model getting the same result for both graphs, because we only train on the subgraphs and the node ids are not used as a feature. Hence, as long as the pairs of matching subgraphs are not the same ones, the model will treat them as separate graphs. Therefore, it suffices to find pairs of independently matching subgraphs. Same as before, we first generate a random subgraph g of G of k nodes. To find g', it is very time-consuming to look for the global optimal match, hence, we adopt a greedy approach to get a good enough estimate. For each node in g, we find the closest 10 nodes which are in the same class, sorted by the difference in features. We then find all the connected graphs using the top 10 candidates for each node. We then perform a breadth-first search starting from the top candidate for each node, and find the connected subgraph which is closest to g in structure. The structural similarity is computed using the sum of the difference between the distance matrices, similarly as in the matching loss. For this experiment, I generated 200 pairs of matching subgraphs of 3, 4 and 5 nodes each.

4.1.3 MovieLens Dataset

MovieLens is a dataset which describes the rating and free-text tagging activity from movielens.org, a movie recommendation service. It is a widely used dataset to evaluate the performance of recommender systems. We use the version from September 2018, containing 100836 ratings and 3683 tag applications across 9742 movies from 610 users. The dataset contains two types of entities, movie and user, each uniquely identifiable by an id. Each movie may belong to one or more genres and user rating is on a scale from 1 to 5. The dataset also provides the timestamp when a rating is posted. Since the model is not targeted at solving heterogeneous graph matching problems which can contain different types of nodes, we construct nodes for movies only. For each movie node, we use a 22-dimensional feature vector. The first feature

is the year of the movie and the second is the number of ratings it reviewed, the third feature records the average rating the movie receives, and the fourth one records the frequency where the movie is rated based on the timestamps. The rest 18 dimensions correspond to the movie genres, sorted in alphabetical order, the corresponding feature is set to 1 if the movie belongs to the genre. Note that some movies belong to multiple genres so this is not a one-hot encoding. Two movie nodes are connected if they are rated by the same user. After constructing the derived graph, we remove a few nodes which are not connected. The process of generating pairs of matching subgraphs is similar to that in Cora processing.

4.2 Experimental Setup

We use the Pytorch library to implement our model and uses the default random initialization method for all neural network layers. We use 2-hop GNNs because the testing subgraphs are quite small. For all identified subgraph pairs in the three datasets, we divide them into a 3:1:1 ratio for training, validation and testing. After each training epoch, we obtain the validation accuracy by applying our model on the validation set. We terminate the training process if the validation accuracy decreases for two epochs and revert the network setting to two epochs before. When constructing the node attribute matrix, we consider both categorical and continuous attributes. For categorical node attributes such as color, we record them using one-hot encodings. For continuous attributes such as movie ratings for MovieLens, we normalize the values using the formula k / (max - min) where max and min are the largest and smallest value of that attribute and k is the current attribute value. Hence, we are able to construct node vectors by concatenating all the attributes and build the attribute matrix. We set the learning rate to 0.01 and use the Adam optimizer [Kingma, 2014] for error backpropagation. For the three baselines, i.e., the Local Description(LD) architecture, OA kernel and the index-based TALE model. For the LG architecture, we implement it in Python and use the same definitions in the paper. For OA kernel, given the input graphs G, G' and $g \subseteq G$ we use the GNN models to generate the node embeddings and use it as the kernel function for nodes. For each node in g, We then find the top 3 candidates in the G' using similarity in node embedding and put all the candidates together to form a candidate graph. Note this step is not included in the original OA kernel, however, trying with every possible assignment between g and G' is not feasible. Finally, all mappings between g and the candidate graph are compared to find the optimal one. For TALE,I use the same setting in the paper which generates a neighbourhood index for each node, and progressive match the nodes.

We have conducted three sets of experiments. Firstly, evaluating how well does our model perform on the attributed subgraph matching problem and compare with the other three baseline methods. Secondly, examine how the subgraph size and node feature complexity affect the matching result. Lastly, what is the significance of the hyperparameter λ in the matching loss function affect the result.

For the first set of experiments, we note one important difference of our model from the baselines, which is, our model trains on pairs of subgraphs, but the baselines use whole graphs directly. Therefore, if the baselines are trained on the Cora and MovieLens dataset where G and G' are identical, it is obvious that our result would be much worse. Therefore, we only tested on the synthetic dataset. For this experiment, we set λ to be 0.5.

The second set of experiments compares the performance of our model with subgraphs of varying complexity. Obviously, the three datasets have different node feature complexity: The synthetic dataset has only one node feature, the MovieLens dataset has 22 while the Cora dataset has 1433. For each dataset, we also study the matching result for different sizes of subgraphs. For this experiment, we set λ to be 0.5.

In the last set of experiments, we compare the performance of our model with different λ. This parameter decides the significance of structural similarity in our design. For each dataset, we choose three λ values 0, 0.5 and 1 and keep all other settings constant. We use subgraphs of three nodes.

4.2.1 Performance Measure

We use two popular performance measures, including accuracy and precision. Accuracy is defined as the number of correctly-identified matching subgraph over the total number of test instances. Precision is the measure of correctly identified matching nodes over the total number of test instances times the subgraph attribute size. Note that the precision, in this case, is greater than or equal to the accuracy in this problem.

4.3 Results

4.3.1 Performance Comparison with the Three Baselines on the Synthetic Dataset

We use the same training set for our method and the three baseline methods. For our method and the OA kernel, we take the average experimental results over 10 runs. The other two methods are deterministic hence only run once. We record the accuracy of the four methods.

Table 4.1: Accuracy comparison on the four models on the 40 instances in the synthetic dataset

Subgraph size	Our model	LG	OA kernel	TALE
3	0.61	0.52	0.49	**0.75**
4	0.57	0.43	0.45	**0.70**
5	0.50	0.29	0.38	**0.63**

Table 4.1 shows the accuracy comparison between our model and the three exist-

ing baselines. We observe that while our model outperforms the LG and OA kernel methods by a significant margin, the index-based TALE model has higher accuracy on all instances. It is also quite stable, mainly attributed to the fact that nodes are added to the mapping sequentially. Surprisingly, the OA kernel which extensively searches for all mappings between the query and candidate graph performs very poorly. It may confirm my early experimental result that node embeddings generated by GNN do not capture enough structural information for the matching task. Though our model does not have the best performance, it is understandable because the design idea, especially sequencing the data is quite new and there is great room for future improvement. It can also be observed that our model is stable. It may suggest that the node sequencing method is learning correlations quite well.

4.3.2 Experiment on All Datasets with Different Subgraph Size

For each instance of this experiment, we present the testing accuracy and precision. An matching subgraph found is said to be accurate if it matches the ground truth. We also record the precision, which is the percentage of correctly identified nodes which are in both the found subgraph and the ground truth. We can use the discrepancy between these two values to analyze the reason behind the matching performance of our model. Table 4.2 lists the experimental results. From table 4.2, we observe

Table 4.2: Performance comparison on subgraphs of varying complexity, each taking the average of 10 runs

Dataset	Subgraph size	Accuracy	Precision
Synthetic	3	0.61	0.70
Synthetic	4	0.57	0.70
Synthetic	5	0.50	0.62
Cora	3	0.62	0.72
Cora	4	0.52	0.68
Cora	5	0.48	0.66
MovieLens	3	0.59	0.71
MovieLens	4	0.52	0.64
MovieLens	5	0.40	0.59

that generally, the model performance is reasonable but not robust. We continue to analyze the impact of node attribute similarity and subgraphs size.

4.3.2.1 Analysis of the impact of node attribute complexity on the matching outcome

From the results, it is surprising to observe that more complexed attributes do not necessarily lead to worse matching outcomes. The reason behind may be more complexed attributes makes the nodes more easily separable by their embeddings. It is intuitive that two nodes in a synthetic dataset may not differ by much because there

is only one attribute, while even similar nodes in the Cora dataset may differ by a much larger margin. However, we also observe that more complicated attributes tend to lead to larger embedding loss and outweighs the structural similarity. This is evident from the huge drop in accuracy when the number of nodes in the subgraph increases from 3 to 4. The precision is very high for the Cora dataset even though the accuracy is low. It may show that the node embeddings from GNN indeed learn the node attributes extensively.

4.3.2.2 Analysis of the impact of subgraph size on the matching outcome

While it is intuitive that larger graphs lead to worse matching outcomes, we are interested in the margin of performance drop when increasing the size of subgraphs. We observe that the drop in accuracy is greater in all instances when the subgraph size increases from 4 to 5 than that from 3 to 4. This is rather counter-intuitive since if we consider the probability of correctly matching a node to be p, under the independence assumption, the rate of decrease in the value p^n slows down as n increases. A possibility is that the independence assumption is very flawed here, the strong correlation between the nodes learned during the structural matching process may impose strong correlation between the nodes found hence any mistake will lead to an inaccurate instance.

4.3.3 Experiment on λ

Similarly as before, we present the precision and accuracy of our modeling. We evaluate the performance on all three datasets using three different values of λ. Table 4.3 lists the results. From Fig 4.3, we can observe the significance of having the

Table 4.3: Performance comparison using different λ on subgraphs of 3 nodes

Dataset	λ	Accuracy	Precision
Synthetic	0	0.48	0.57
Synthetic	0.5	0.61	0.70
Synthetic	1	0.58	0.66
Cora	0	0.59	0.64
Cora	0.5	0.62	0.68
Cora	1	0.69	0.73
MovieLens	0	0.48	0.55
MovieLens	0.5	0.59	0.71
MovieLens	1	0.62	0.72

structural matching term in the loss function. For all three datasets, the result of having $\lambda = 0$ is much lower than when using the other 2 values.

4.3.3.1 Analysis of the impact of lambda on the matching outcome

For datasets with complicated node attributes such as Cora, setting λ to 0 only detrimentally affects the result by a small margin, very likely due to the fact that node embeddings alone can differentiate most nodes. The impact of increasing *lambda* from 0.5 to 1 also differs for the three datasets. This can also be attributed to the significance of attributes in the two real-world dataset. In general, we observed that a non-zero λ is necessary, but the actual value may be problem-dependent. The current structural matching loss may be too small in comparison with the embedding loss for complicated datasets. Developing a dynamic matching loss measure depending on the context of graphs may be a better way to solve this problem.

4.4 Chapter Summary

In this chapter, we evaluated our models on the three datasets and compared the performance against three existing measure. We also observed and analyzed the reason behind the change in matching performance caused by different settings. The performance of the model is not great, hence in the next chapter, we discuss possible future research directions to improve our model.

Chapter 5

Conclusion and Future Work

In this work, we have proposed a novel neural network architecture for the attributed subgraph matching problem. Our experiments showed reasonable but not excellent results. In this chapter, we look at potential future research directions to improve the performance of our model and explore possibilities to expand our model to solve problems beyond attributed subgraph matching.

5.1 Future Work

5.1.1 Alternative to GNN

Though GNN is shown to be very successful in learning node embeddings, there exist related alternative architectures which are more suited for the matching problem. These models may give us better node embeddings for the matching purpose. Recently, there have been efforts such as the Graph Matching Network[Li, 2019] to learn the node embedding of two graphs jointly with some cross-graph attention mechanism. We could develop a variation of our model by using the GMN for the feature learning section.

5.1.2 Alternative to Euclidean Space Embedding

Almost all current machine learning effort to solve graph matching problems use similarity measures and node embeddings in the euclidean space. However, graph is a rich structure and the euclidean space may be unable to fully represent the data structure. Recent researches such as [Frogner, 2019] propose using the Wasserstein space to represent graphs, where each node is mapped into a distribution. Hence, the nodes constituting a graph can be modelled as a combine distribution. This leads to increased flexibility and stronger representation power. Distribution matching could also be an interesting way to model graph matching problems.

5.1.3 Extend to Involve edge attributes

Our current model assumes no edge attributes. However, in real-life applications of attributed subgraph matching, edges usually carry semantic information as well.

Considering the edge attributes also helps us take into account the type of correspondence between nodes in the graph. To achieve this, we would need to come up with more sensible ways to serialize the nodes.

5.1.4 Extend to directed graphs

Our current model assumes undirected graphs. In real life attributed graphs, the edges are often directional, i.e. indicate relationship from one node to another. It would not be a difficult task since we have established serialization methods for directed graphs such as topological sort.

5.1.5 Heterogeneous graph matching

Currently, our model matches graphs where all nodes are of the same type and we tune datasets to meet this criterion. Heterogeneous graphs, which can contain different types of nodes and edges, are also common in real-life applications. It is challenging but possible to incorporate heterogeneous graphs into our model, using latest architectures such as the Heterogeneous Graph Attention Netowrk[Xiao, 2019].

Bibliography

Baskararaja, R., G. and Manickavasagam, S., M., 2012. Subgraph matching using graph neural network. (2012). (cited on page 12)

Bengoetxea, E., 2002. The graph matching problem. (2002). (cited on pages 5, 9, and 14)

Bollobas, B., 1998. Modern graph theory. *"Graduate texts in mathematics"*, (1998). (cited on page 3)

Borgwardt, K. M., 2005. Shortest-path kernels on graphs. (2005). (cited on page 7)

Borgwardt, K. M., 2020. A survey on graph kernels. (2020). (cited on page 6)

Caetano, S., T., 2009. Learning graph matching. *"IEEE T"*, (2009). (cited on pages 4, 5, and 18)

Carletti, C., 2016. Exact and inexact methods for graph similarity in structural pattern recognition phd thesis of vincenzo carletti. (2016). (cited on page 6)

Chung, J., 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. (Dec. 2014). (cited on page 13)

Du, B., 2017. First: Fast interactive attributed subgraph matching. *"KDD'17"*, (Aug. 2017). (cited on page 10)

Fey, M.; Lenssen, E. M. C., J.; and Masci, J., 2020. Deep graph matching consensus. (2020). (cited on pages 1 and 12)

Fout, t. . P. y. . . p. . N., A. (cited on page 14)

Frogner, C., 2019. Learning embedding into entropic wasserstein spaces. *"Proceeding of the International Conference on Learning Representations"*, (2019). (cited on page 33)

Frohlich, H.; Wegner, K., J.; and Sieker, F., 2005. Optimal assignment kernels for attributed molecular graphs. *"International Conference on Machine Learning"*, (2005). (cited on pages 11 and 25)

Garcia, V., 2017. Few-shot learning with graph neural networks. (2017). (cited on page 14)

Gartner, T.; Flach, P.; and Wrobel, S., 2003. On graph kernels: hardness results and efficient alternatives. (2003). (cited on page 7)

Gers, F., 1999. Learning to forgetl continual prediction with lstm. (1999). (cited on page 13)

Jouili, S., 2009. Attributed graph matching using local descriptions. (2009). (cited on pages 10 and 25)

Kashima, H.; Tsuda, K.; and Inokuchi, A., 2003. Marginalized kernels between labeled graphs. *"International Conference on Machine Learning"*, (2003). (cited on page 7)

Khan, A.; Wu, Y.; and Aggarwal, C., 2013. Nema: fast graph search with label similarity. (2013). (cited on page 11)

Kingma, D., 2014. Adam: A method for stochastic optimization. (Dec. 2014). (cited on page 28)

Kipf, T., 2016. Semi-supervised classification with graph convolutional networks. (Sep. 2016). (cited on page 13)

Kriege, N., 2012. Subgraph matching kernels for attributed graphs. *"Proceeding of the 29th International Conference on Machine Learning"*, (2012). (cited on page 11)

Le, X.; Ho, T., H.; Lee, G.; and Jung, S., 2019. Application of long short-term memory(lstm) neural networks for flood forecasting. (2019). (cited on page 22)

Li, Y., 2019. Graph matching network for learning the similarity of graph structured objects. *"Proceeding of the 36th International Conference on Machine Learning"*, (2019). (cited on pages 8, 21, and 33)

Li, Y.; Tarlow, D.; Brockschimdt, M.; and Zemel, R., 2015. Gated graph sequence neural networks. (2015). (cited on pages 13 and 20)

Livi, L., 2013. The graph matching problem. *"Pattern analysis and applications"*, (2013). (cited on pages 3, 5, 6, and 7)

Murugan, P., 2018. Learning the sequential temporal information with recurrent neural networks. (2018). (cited on page 22)

Riesen, K. and Bunke, H., 2010. Graph classification and clustering based on vector space embedding. (2010). (cited on page 8)

Robles-Kelly, A. and Hancock, E., 2005. Graph edit distance from spectral seriation. (2005). (cited on page 8)

Sak, H.; Senior, A.; and Beaufays, F., 1953. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. (1953). (cited on pages 18 and 22)

Scarselli, F., 2008. The graph neural network model. *"IEEE transaction on Neural Networks 20.1"*, (Dec. 2008). (cited on pages 8, 12, and 13)

Shearer, K.; Bunke, H.; and Venkatesh, S., 2000. Video indexing and similarity retrieval by largest common subgraph detection using decision trees. (2000). (cited on page 9)

Subramaniam, A., 2016. Deep neural networks with inexact matching for person re-identification. (2016). (cited on page 11)

Tian, Y., 2008. Tale: A tool for approximate large graph matching. *"IEEE 24th International Conference on Data Engineering"*, (2008). (cited on pages 9 and 25)

Velickovic, C.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y., 2017. Graph attention networks. (2017). (cited on page 20)

Xiao, X., 2019. Heterogeneous graph attention network. *"The World Wide Web Conference"*, (2019). (cited on page 34)

Zanfir, A., 2018. Deep learning of graph matching. *"Proceeding of the IEEE Conference on Computer Vision and Pattern Recognition"*, (2018), 2684–2693. (cited on page 11)

Zhang, S., 2009. Gaddi: Distnace index based subgraph matching in biological networks. (2009). (cited on page 9)

Zhang, S., 2016. Final: Fast attributed network alignment. *"Proceeding of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining"*, (Aug. 2016), 1345–1354. (cited on page 10)

Zhu, L., 2010. Structure and attribute index for approximate graph matching in large graphs. *"Information Systems"*, (2010). (cited on page 9)