**Introduction**

1. The interest of inexact graph matching has been recently increased in the last years due to the application of computer vision to areas such as cartography, character recognition, and medicine. In these areas, automatic segmentation of images results in an over-segmentation and therefore in the data graph containing more vertices than the model graph. That is why applications on these areas do usually require inexact graph matching techniques. In cartography, the typical example is when a graph is used to represent the knowledge extracted from of a map storing all the features. The matching with an image consists in identifying structures in the image with the help of the map. In character recognition, a model in the form of a graph is generated for each character and the objective is to find which is the model that best suits the analyzed image of a character. In medical images, graphs can be used to represent an anatomical atlas. As a concrete case of the latter, in brain imaging internal brain structures can be recognized with the help of a graph where each vertex represents a brain structure in the atlas, and edges represent spatial relationships between these structures. In the case of the recognition of human facial features from images the regions in the model represent each of the features to be recognized such as mouth, eyes and eyebrows

**General points for introduction:**

1. Graph is a commonly used data structures to encode structural information compactly. Over the last decade, the endeavour of brilliant researchers have helped usunderstand subject better. Nowadays, graphs are widely used in computer sciencerelated areas such as computer vision, navigation systems and database operations[].
2. Graph matching is the problem of finding the similarity between graphs. It is known to be NP-complete[]. Researchers have developed many algorithms to approximate the optimal solution. The rapid development in machine learning gives rise to more robust approximation methods to this problem[][][]. In this work, we focus on the attributed subgraph matching problem, which is the problem of finding the best matching subgraph in a graph *G* for a query graph *G’*.
3. We propose an end-to-end neural network model called ..., which trains the feature learning and matching process jointly. (results).

**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 a reasonably high accuracy.
* Perform experiments on a synthetic dataset and real world data to validate our model.

**1.2 Contributions**

The main contribution of this work are:

* We show that current methods require handcraft graph similarity measures, which may not generalize well to all attributed subgraph matching tasks. We propose a new model, called ...., which uses a neural network method to learn the similarity measure automatically.
* We introduce the novel idea of treating graphs into sequence of data to study the graph structure by investigating the correspondence between points efficiently and effectively.

**1.3 Thesis Outline**

In chapter 2, we look at the graph matching problem in detail and review related works. We then give explain our model architecture in detail in chapter 3. Chapter 4 presents the model performance on both synthetic and real world datasets. Lastly, we discuss directions of future work in chapter 5.

**Background and Related Work**

This chapter gives an overview of graph matching. In particular, we discuss the attributed subgraph matching problem and review existing theoretical and machine learning methods for this problem. This chapter also briefly describes related general graph processing models.

**2.1 Graph Matching**

Generally speaking, the graph matching problem is about studying the similarity between two graphs. We can state the problem as followed: Given two graphs *G=(V, E) and G’=(V’, E’)*, where *|V| = |V’|*, we want to find a bijection *f : V → V’* such that adjacency is preserved, i.e. *(v1, v2)∈E* iff *(f(v1), (v2))∈E.* This problem for matching two graphs of the same size is also called the graph isomorphism problem[1]. One common variant to this is the subgraph matching/isomorphism problem, where we have *|V| < |V’|* and aim to find a subgraph of *G’* which is isomorphic to *G.* Both the graph and subgraph isomorphism problem are NP-complete[2], researchers have proposed approximation methods[3][4][5] for this problems.

**2.1.1 Attributed Graph**

In modern applications such as computer vision and data mining, graphs are usually used as a structured tool to store rich information. These information are stored as node and edge attributes.

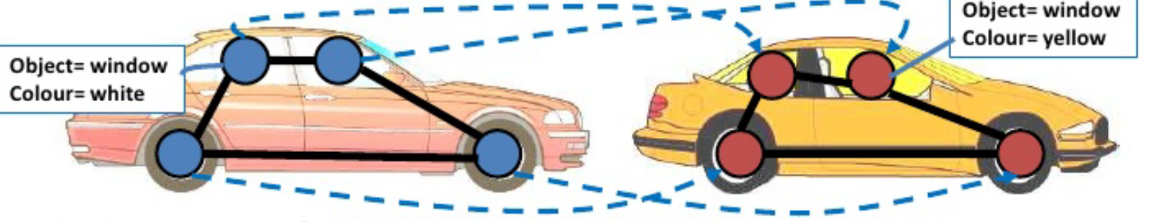


Fig 2.1: Example of attributed graphs

Fig 2.1 gives an example of attributed graphs. It could appear in a pattern recognition tasks where we detect the red car on the left and extract the components such as the windows. We put then use the objects as nodes and their features as attributes. The resulting connected graph compactly represents the detected car.

**2.1.2 Attributed graph matching**

The two graphs in Fig 2.1 are isomorphic and the node mapping is denoted by the dotted lines. However, it is not always possible to find an isomorphism between two graphs. In the car recognition task, the detected car may have slightly different structure from the stored pattern, for example, different number of windows. If the nodes are connected in a loop similar to Fig 2.1, it is impossible to find an exact match between nodes. In this case, the graph matching problem aims to find the best match. This solution to this inexact matching problem is a non-bijective correspondence between two graphs which maximize some similarity objective function. This problem is an optimization problem which is NP-hard[6]. In this work, we are interested in the attributed subgraph matching problem, that is the problem of finding the best matching subgraph in a graph *G* for a query graph *G’*.

**2.2 Existing methods**

Various attributed subgraph matching methods have been proposed for the graph matching problem. N. Kriege[]. use a subgraph matching kernel to compare nodes and edges, thereby inducing a flexible similarity measure between two subgraphs. []H. Tong proposes G\_Ray which uses random walk with restart to design a straightforward similarity measure between nodes, then use this measure to find matching nodes in the query graph sequentially. Approaches such as [] use a hybrid index structure incorporating graph structural information to prune unwanted subgraphs effectively. More recently, Zhang et al. proposed a family of algorithms FINAL[] utilizing the alignment(matching) consistency principle, which basically states that the for a pair of matching nodes across graphs, their close neighbour with similar attributes have a high chance to be matched as well. Zhang’s FIRST[] model build upon this and allows fast matching for slightly modified query graph. Though most of these methods are effective, they mainly adopt a two-step process, i.e. first handcraft some similarity measure then optimize for this measure. While the reason behind all these similarity measures are sound, all these different approaches may suggest that these measures are problem-dependent.

Most recently, Zanfir and Sminchisescu[] uses deep learning to learn feature hierarchies and induce graph structure, and uses a voting-based loss function to train the feature learning and matching steps jointly. (what I want to do that is different from this.)

**2.3 Graph Neural Network**

In this section we go through the popular graph neural network dedicated to processing graphs.

\subsection{Graph Neural Network}

\begin{figure}[H]

\centering

\includegraphics[width=0.5\textwidth]{figs/GNN.png}

\caption{An illustration of 1-degree (hop) graph neural network[]. The model works based on the principle of message passing.}

\end{figure}

Graph Neural Network(GNN), as suggested by its name, is a machine learning model specifically dedicated to processing graphs. It uses propagation to learn about neighbourhood information for every node and outputs node embeddings. Originally formalized by Scarselli et al.[], GNN uses propagation to learn node representations in graphs. It has been rapidly gaining popularity among researchers after modern deep learning methods were incorporated into the model[][].) Fig 4.1 shows the standard 4-stage process for an 1-hop GNN conceptually. 1-hop means that for each node, only information about its immediate neighbours are propagated. The model takes in a graph and its adjacency matrix as input, it outputs vector embedding of every node. In stage A, we initialize the node and edge attributes of the graph. We then find the neighbours for each node in B. Stage C shows message propagation to the central node from its neighbours, each message contains information about the neighbour and the edge. The last stage shows that the central node aggregates information it just collected and combine with its own information. The last two stages exhibits message passing, the central idea of GNN. For a n-hop GNN, the last two stages are repeated n times. The model output are the node embeddings.

\subsubsection{Analysis of GNN}

GNN adopts a propagation strategy to capture structural information of the input graph, its affinity towards the data structure makes it a natural choice for graph learning. There are numerous experimental results proving its effectiveness. However, there are still some limitations to the model. One problem with GNN is its difficulty to capture long-term dependencies. Consider two nodes that are more than two nodes apart, using 1-hop GNN does not study the relation between them at all. Theoretically, increase the number of hops can always solve this problem. However, with each additional hop, the time complexity of the algorithm increases exponentially because the of increase in the neighbourhood size. Another problem of GNN comes from its propagation step. For each iteration of propagation, we update the representation of each node concurrently. This iterative update process for fixed nodes may cause the distribution of node representations to be smoothed and become less informative for distinguishing each node[].

There are many other variants of GNN to avoid its limitations, such as Graph Convolution Network(GCN)[] which performs graph convolution at each step, Gated Graph Neural Networks[] which uses gate mechanisms such as GRU[] or LSTM[] in the propagation step to improve long-term propagation of neighbourhood information.

**2.4 Summary**

In this chapter, we discuss different types of graph matching problems, particularly attributed graph matching. We then review some theoretical and machine learning approaches to solve these problems. We also describe and analyze the graph neural network, which will play an important role in the design of our model.

# advantage over other methods

Automatically learn what to match, for example, pattern recognition of cars, the color of windows do not matter that much. Methods like [][] uses the euclidean distance between the all node features as a distance measure, while neural network sets weights for each input attribute and get a weighted representation of the attributes etc.

**Experiments**

This chapter describes the implementation detail of the (model name) model. We conduct experiments on on three dataset under different settings and interpret the results. We have evaluated the following research questions:

1. How our proposed model matches subgraphs of different sizes and feature complexity
2. How the choice of hyperparameters, particularly $\lambda$ in the matching loss function, affect the matching result.

\section{Datasets}

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

\subsection{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

\begin{figure}[H]

\centering

\includegraphics[width=0.80\textwidth]{embedding.jpg}

\caption{Pseudocode for generating the synthetic dataset}

\end{figure}

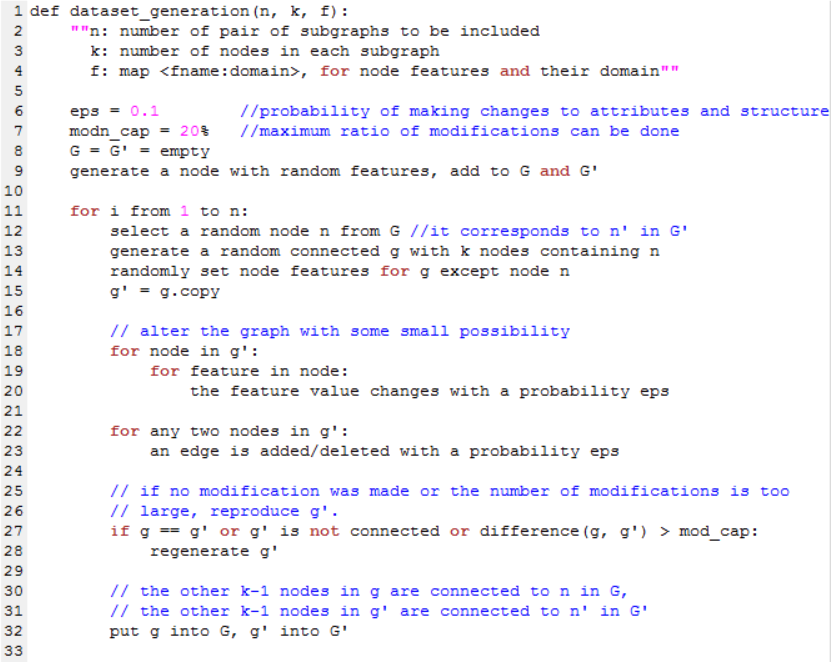


Fig 5.1 shows the algorithm for dataset generation. The generator takes in 3 parameters. n is number of pairs of subgraphs to be generated, k is the number of nodes in each subgraph, and f stores the node features and their domain..

We start with $G$ and $G'$ both empty, then perform a recurrent process of two steps:

\newline

\textbf{Step 1:}Generate one pair of matching subgraphs $(g,g')$

\newline

\textbf{Step 2:}Aggregate $g$ into $G$, $g'$ into $G'$

\subsubsection{Pair Generation}

Line 12 to 14 in Fig 5.1 generates a random connected graph $g$ of k nodes. Using $g$ as the reference graph, line 15 to 23 then construct a matcing 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, it is vital that this id is not visible to the model. The probabilities for either edge or features being edited are set to be 0.1. The modification cap is 20 percent.

\subsubsection{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 $G’$.

\subsection{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 the 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 difference between the distance matrices, similarly as in the matcing loss. For this experiment, I generated 200 pairs of matching subgraphs of 3, 4 and 5 nodes each.

\section{Experimental Setup}

\subsection{MovieLens Dataset}

MovieLens is a dataset which describes 5-star 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. The latest version is generated in 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 corresponds to the movie genres, sorted in alphabetical order, the corresponding feature is set to 1 if the movie belongs to the genre. 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.

\section{Experimental Setup}

The model is implemented in Pytorch. For all the datasets, we divide the dataset into a 3:1:1 ratio for training, validation and testing. For categorical input features such as color, we encode them using one-hot encodings. For numerical features 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 feature and k is the current feature value. We set the learning rate is 0.01 and use theAdam optimizer[] for gradient backpropagation. We terminate the training step if the validation accuracy is above a certain threshold. The selection of the threshold is based on the problem instance.

\newline

There are three main purposes of our experiments. Firstly, evaluating how well does our model perform on the attributed subgraph matching problem. Secondly, how does the size of subgraphs, size of the whole graphs and complexity of node features affect the matching result. Lastly, what is the significance of the hyperparameter \lambda in the matching loss function affect the result. To investigate these problems, we prepare two sets of experiments.

\newline

The first set compares the performance of our model with the same hyperparameters across datasets. Obviously, the three datasets have different node feature complexity: The synthetic dataset has only one node feature, the MovieLens dataset has 22 while the feature dimension is 1433 for the Cora dataset. Moreover, the graphs obtained from the datasets have very different size. For each dataset, we also study the matching result for different sizes of subgraphs. For this experiment, we set \lambda to be 0.5.

\newline

In the second set of experiments, we compare the performance of the model with different \lambda. Essentially, this parameter decides significance of structural similarity in this task. For each dataset, we choose three \lambda values 0, 0.5 and 1 and keep all other settings constant. We choose to match subgraphs of size 4.

\section{Results}

\subsection{First Experiment}

For each instance of this experiment, we present the testing accuracy. 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. Note that the precision in this case is greater than or equal to the accuracy. We can use the discrepancy between these two values to analyze the reason behind the matching performance of our model.

|  |  |  |  |
| --- | --- | --- | --- |
| Dataset | Subgraph size | Accuracy | Precision |
| Synthetic | 3 | 0.61 | 0.70 |
| 4 | 0.57 | 0.70 |
| 5 | 0.50 | 0.62 |
| Cora | 3 | 0.62 | 0.65 |
| 4 | 0.52 | 0.57 |
| 5 | 0.50 | 0.55 |
| MovieLens | 3 | - | - |
| 4 | - | - |
| 5 | - | - |

Table 1: Average experimental results over 10 runs on the three datasets using different sizes of subgraphs. $\lambda$ is 0.5.

|  |  |  |  |
| --- | --- | --- | --- |
| Dataset | lambda | Accuracy | Precision |
| Synthetic | 0 | 0.48 | 0.57 |
| 0.5 | 0.61 | 0.70 |
| 1 | 0.58 | 0.66 |
| Cora | 0 | 0.43 | 0.51 |
| 0.5 | 0.62 | 0.65 |
| 1 | 0.63 | 0.65 |
| MovieLens | 0 | - | - |
| 0.5 | - | - |
| 1 | - | - |