GraphDB based Graph Mining Library for News Prediction and Recommendation

M. Ege Çıklabakkal
Middle East Technical University
Üniversiteler Mahallesi
Dumlupınar Bulvarı No:1
06800 Çankaya/Ankara, Turkey
ciklabakkal.ege@metu.edu.tr

Mert Erdemir
Middle East Technical University
Üniversiteler Mahallesi
Dumlupınar Bulvarı No:1
06800 Çankaya/Ankara, Turkey
mert.erdemir@metu.edu.tr

Pınar Karagöz
Middle East Technical University
Üniversiteler Mahallesi
Dumlupınar Bulvarı No:1
06800 Çankaya/Ankara, Turkey
karagoz@ceng.metu.edu.tr

ABSTRACT

The task of extracting useful information from a large database is becoming increasingly challenging and important. When relationships are prioritized, it makes great sense to use a graph database thanks to performance improvements and flexibility. Various types of data can be represented using graphs. In this work, we are representing textual data as a graph. The proposed graph model produces a compact and informative representation of news content, allowing us to store sufficient amount of data while generating the results of database queries quickly. In order to simplify the process of working on such databases, we develop a Graph Mining Library that supports mining of frequent subgraphs, frequent sequences of these subgraphs, extraction of rules and calculation of similarities among subgraphs. gSpan algorithm is used to mine the frequent subgraphs. Frequent sequences are found using Apriori treating each subgraph as an item. Association rules are found by processing these frequent sequences based on a confidence threshold. By having collected association rules corresponding to a certain time period, we can make predictions for the next time slot. Frequent sequences are generated for the next time slot which are then tested for similarity against the antecedents of the rules previously found. Similarity match allows us to predict the consequent.

Keywords

Graph Theory, Graph Mining, Graph Similarity, Graph Database, Association Rule Mining

1. INTRODUCTION

With the increasing use of the Internet, the public data is getting larger every day. In particular, news web sites and social media data make a big contribution to this pile of information. The structure of these sort of data contain many relationships, which makes a graph structure a good candidate for representation. Graph based structure of information simplifies representation of complex and abundant relationships between distinct entities as well as extraction of different types of information.

Within the scope of this project, we aim to develop a method that uses graph similarity and frequent subgraph mining techniques together in order to obtain informative rules by association rule mining. By graph similarity, we imply the structural correspondence of nodes and edges of two graphs. Notion of the similarity between graphs can be isomorphism, edit distance, maximum and minimum common subgraphs, and statistical comparison. Association rule mining on graphs finds the relationships between sets/sequences of subgraphs. These rules in turn, can uncover the hidden patterns in the data and can also be used for prediction purposes.

After the application of subgraph mining, many similar subgraphs are generated, some of them with only changing the named entity tags. Our motivation is to reduce the number of subgraphs and retrieve more quality sets, so that the reduced number of rules can increase the effectiveness of association rule mining. Prediction and recommendation applications on textual data can benefit from this approach.

To represent the textual data in the form of a graph structure, we generated a graph model that is capable of storing not only bigram relationships between the consecutive words in a news content and inclusiveness relationships between news and words of its content but also the named entity property of all the words. We applied *Frequent Subgraph Mining* algorithms and techniques to the graphs to retrieve common subgraphs from the news graphs. For this purpose, we used *gSpan Algorithm*. In order to decrease the number of frequent subgraphs and eliminate similar ones, we used graph similarity. Clustering the subgraphs using Graph One Hot Encoding as the similarity algorithm allows this sort of pruning. We have mined frequent sequences from these subgraphs. Using them, we generated association rules which are used to make predictions and recommemdations about the future patterns/news.

Association rule mining is generally applied on itemsets. To the best of our knowledge, this is the first publication applying association rule mining on news data represented by a graph structure for recommendation purposes. We are mostly interested in frequent subgraphs, we mine association rules on a set and a sequence of these subgraphs. Since we are working on news data, we expect closely related news to form strong rules which can be used in recommendation or grouping of news.

The rest of the paper is organized as follows. In Section 2 we present a summary of related works, which is followed by Section 3 where we detail the dataset used along with our model to represent it in the graph database. Then in Section 4, we discuss our main work of frequent subgraph mining, graph similarity, frequent sequence mining, association rule mining and prediction. In Section 5, our experiments can be found and finally in Section 6, we summarize our work, mention about further research and conclude.

2. RELATED WORK

In the age of Big Data, extracting useful information has become a both challenging and a valuable task. Data Mining and mining frequent patterns in particular has been effective ways of solving this challenge. This section is organized into two subsections. In the first subsection, brief information regarding frequent subgraph mining are given. In the second subsection, related graph similarity studies are summarized.

2.1 Frequent Subgraph Mining

One of the most well-known algorithms in this field is **Apriori**. In their work, Agrawal and Srikant proposed this method for mining frequent itemsets and association rules [2]. An association rule is an implication of the form $A \Rightarrow B$, where both A and B are itemsets that have a certain correlation or an implication relationship. The method they proposed to mine association rules in this paper is applied in our work as well.

Apriori takes advantage of the downward closure property which states that any subset of a frequent itemset will also be frequent. There have been other algorithms developed after Apriori that also utilize this property. An example is the work of Rajput et al. in which they propose a binary matrix model to overcome the huge candidate generation problem[6]. Their work also focuses on textual data and for preprocessing, they have eliminated stop words, used stemming, filtered terms based on upper and lower thresholds by **Tf-idf** method.

Candidate generation of Apriori makes it very expensive and it doesn't allow it to scale well. That's why Han et al. [5] proposed **FP-growth** algorithm, which does not generate any candidates. This algorithm has also been used extensively in frequent itemset mining applications, a small improvement over the tree structure of FP-growth is achieved by generating a graph based structure[10].

These methods have been focusing on itemsets and suits very well when the data is kept in a traditional SQL database. However, in our project, we are utilizing a graph database. In order not to have any overheads caused by conversion between these data structures, graph mining is preferred.

gSpan is a graph-based substructure pattern mining proposed by Yan and Hal [12]. The algorithm combines the growing and checking of frequent subgraphs into one procedure, thus it can be fast while also being memory efficient. gSpan is the algorithm we use in our work for the frequent subgraph mining task because of the aforementioned reasons.

2.2 Graph Similarity

In this subsection we give a brief background information about the graph similarity methods we used in our paper.

2.2.1 Graph Edit Distance

Since the graph similarity relies on similar pattern between two different graphs, it is highly used pattern recognition techniques. One of the popular algorithms is *Graph Edit Distance* , which is firstly proposed by Sanfeliu et al. [7] in 1983. The main aim of the algorithm is based on finding the best set of transformations, i.e. editing graph's nodes and edges, on a graph to transform it to another graph. Counting cost of the each transformation can give an information about how much similar these graphs are. For example, let G_x denote a graph with id number of x. For G_1, if the total cost for G_2 is less than the total cost for G_3, G_1 can be said that it is more similar to G_2 than G_3.

From a different perspective, Abu-Aisheh et al.[1], implemented *Graph Edit Distance* algorithm based on depth-first search. They experimented on their dataset in terms of speed, accuracy and classification rate and compare the results of their approach and A^* graph edit distance computation. This work inspired us to start with the *Graph Edit Distance* method.

Also, Cao et al. [3] proposed a similar approach to calculate the graph similarities. In addition to depth-first search, they used Levenshtein Distance, i.e string edit distance. Thus, they turned the graph matching problem into string matching.

2.2.2 Graph One Hot Encoding

Representing the textual data in vector space, is a very common problem for Natural Language Processing applications. Due to its flexibility, Bag of Words (which is Graph One Hot Encoding in our case) approach is one of the widely used methods to embed textual data into vector space. Since it is extremely scalable, it can be used in various different ways to extract information from documents. Discarding ordering information between the words prevents this method drowning in details and complex operations. Rather than structure, this method only concerns about the occurrence of the word in a text[1]. Therefore, this approach can be used for extracting the histogram of the words within a text[4], such as existence, count of the term (words in this case - term frequency). These extraction methods are also highly used in machine learning encoding methods as One-Hot-Enconding (using existence of the term), TF-IDF (using term frequency), etc.

3. DATASET AND GRAPH MODEL

In this section we firstly give a brief information about the dataset used and then explain the graph model which we used to represent the data.

3.1 Dataset

The dataset consists of almost 33k online news data, which was crawled from an online Turkish newspaper site[2]. It covers the news data between January 1, 2015 and December 31, 2016. Also, it contains approximately 500k words. The words are labeled according to their Named Entities and they are lemmatized in the preprocessing stage. We categorized the words in 8 different named entity types. These are *LOCATION*, *ORGANIZATION*, *TIME*, *DATE*, *PERSON*, *PERCENT* and *MONEY*. Labelling and preprocessing of the words are handled by other project members.

3.2 Graph Model

[1]https://machinelearningmastery.com/gentle-introduction-bag-words-model/
[2]https://www.dunya.com

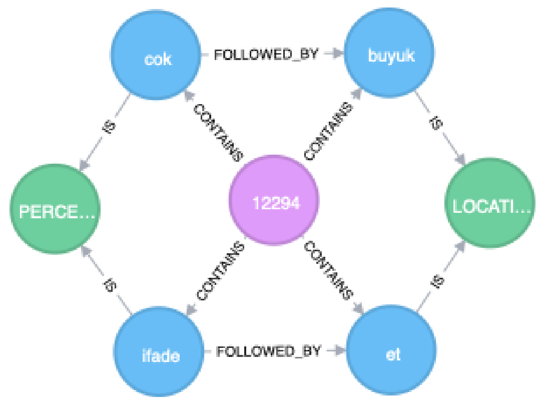

Figure 1: Basic representation of the graph model. Purple node represents a *News* node, green nodes represent *NamedEntity* nodes (*PERCENT* and *LOCATION*) and blue nodes represent *Word* nodes. The relationship types are written on the arrows.

Item	News	Nodes	Relationships	Properties
# of Item	10000	54457	323339	108914

Table 1: Number of Items in the Graph Model

As Wang et al. stated in Preliminaries section of [11], a labeled graph can be defined as the following:

A graph G is formally defined by a quadruple: $G = (V, E, a, b)$, where V is a set of vertices, $E \subseteq V \times V$ is a set of edges connecting the vertices, $a : V \to \Sigma_V$ is a labeling function for vertices, and $b : E \to \Sigma_E$ is a labeling function for the edges (Σ_V and Σ_E being the sets of labels that can appear on the nodes and edges, respectively).

As Zhou et al. mentioned in Preliminaries section of [13], a hypergraph can be defined as:

Let V_H denote a finite set of objects, and let E_H be a family of subsets e of V_H such that $\cup e \in E_H = V_H$. Then we call $G_H = (V_H, E_H)$ a hypergraph with the vertex set V_H and the hyperedge set E_H. A hyperedge containing just two vertices is a simple graph edge.

In our graph model, we used a graph-based approach rather than direct hypergraph model approach. Our graph model adds hypergraph functionality, ex. categorizing words according to their named entity properties, due to the nature of its relationships. However, it cannot fully be considered as a hypergraph due to its node structure. We created 3 different types of nodes as *News* node, *Word* node and *NamedEntity* node. Also, we used 3 different types of relationships as *CONTAINS* edge, *FOLLOWED_BY* edge and *IS* edge. To store the graph model in a graph database, we used Labeled Property Graph[3] approach in Neo4j Graph Database[4].

Representing text content in the form of a graph structure and storing the bigram relationships (in a directed graph) between the words of the text content can be seen in the work of Schenker et al. [8]. Besides the follow-up relationship between the words, which they used for all of the relationships of their model, their work differs from ours in that they used this model for each web document(text content) separately, and assigned distinct relationships to the different web page sections such as title, hyperlink and text. However, rather than only using the relationships between the words we also used the relationships between the different types of nodes such as Word-Document, Word-Named Entity. In the following subsections we will investigate on nodes of our graph model.

[3]https://neo4j.com/blog/rdf-triple-store-vs-labeled-property-graph-difference/
[4]https://neo4j.com

3.2.1 Nodes

In our model, to represent the news data in the form of a graph, we used 3 different nodes.

News node is used for representing a news document itself. A *News* node can only correspond to a specific document. These nodes are connected to Word nodes. Thus, it can be said that they add hypergraph functionality to our graph model since they are grouping words in a news document.

Word node is used for representing each unique word in all documents. Which means that, since all *Word* nodes are unique and a word can be in different news context, a *Word* node can have a relationship with several different *News* nodes. Therefore, the possibility of being connected with several *News* nodes adds hypergraph functionality to our graph model.

Lastly, *NamedEntity* nodes show named entity types of the words. There are exactly 7 different types. *LOCATION* node connected a *Word* node if that *Word* node is a location word. *ORGANIZATION* node connected a *Word* node if that Word node is a organization word. *PERSON* node connected a *Word* node if that *Word* node is a person word. *TIME* node connected a *Word* node if that *Word* node is a time word. *DATE* node connected a *Word* node if that *Word* node is a date word. *PERCENT* node connected a *Word* node if that *Word* node is a percent word. *MONEY* node connected a *Word* node if that *Word* node is a money word. Since these nodes categorizes words according to their named entity properties, they can also bring hypergraph functionality to our graph model.

3.2.2 Relationships

In our model, to connect nodes to each other, we used 3 different types of relationships.

CONTAINS relationships are used for connecting the *News* nodes and *Word* nodes to each other. If a word exists in a content of a news document, we connect such *Word* nodes to the corresponding *News* node. The starting point of the edge is a *News* node and the end point of the edge is a *Word* node. There is no property embedded in the relationship.

IS relationships are used for connecting *Word* nodes to *NamedEntity* nodes. For example, the word 'Larry' is labeled as person in the aspect of named entity property. Then, we connect the node of 'Larry' to *PERSON* node by using *IS* relationship. The starting point of the edge is a *Word* node and the end point of the edge is a *NamedEntity* node. There is no property embedded in the relationship.

At last, *FOLLOWED_BY* relationships are used for connecting *Word* nodes to each other. The main purpose of this relationship is preserving which word comes after another word in a news content. This relationship is used in a similar manner to the relationships stated in [8]. If a word is followed by another word in the content, they are connected to each other by using this edge. However, since all the *Word* nodes are unique and can be member of different news subgraphs, in the existence of a *FOLLOWED_BY* relationship between two common words, it is impossible to understand which news content contains the follow up relationship between these words. To solve this problem, we embedded a property, *contained_in*, to *FOLLOWED_BY* relationship to hold the information of which news content has these successive words. To identify the news we used *News* node id's. Therefore, when we are trying to extract a news subgraph, we neglect the *FOLLOWED_BY* relationships which don't contain the corresponding news id in its *contained_in* property.

4. PROPOSED WORK

This section, which presents the proposed work is organized into 6 subsections. We start by explaining the preprocessing done on the news data in detail, then, frequent subgraph mining and gSpan is introduced. That is followed by graph similarity methods. Moreover, we explain frequent sequence mining and association rule mining. Lastly, we conclude this section with prediction work.

4.1 Preprocessing

Turkish language is very rich morphologically, words such as *"gitmek"*(to go) may appear in many surface forms[9]. Thus, it is logical to use lemmas instead of surface forms. Preprocessing is done in stages, we will be explaining each step in the following subsections.

4.1.1 Tokenization

Data is tokenized so that words and punctuations are single tokens. For apostrophes, the words before and after the apostrophe (included on the after case) are separately tokenized. The reason is that suffixes are not part of the named entities [9].

4.1.2 Disambiguation and Lemmatization

Disambiguation is applied in order to identify the semantics of the word more accurately. Then, words are lemmatized so that forms of the words are grouped. By doing so, we also decrease the number of nodes that would be created from these words.

4.1.3 Parts-of-Speech(POS) Tagging

In order to extract the named entites using Conditional Random Fields(CRF), POS tags should be present in the input. It should be noted that zemberek[5] is used for these mentioned preprocessing stages. The input that CRF is given is in the following:

$$Token - POS - Label$$

where Token is tokenized word, POS is the parts of speech tag such as Noun, Verb, Adj, Punc etc. and Label is just "O" as CRF predicions will modify that field. Also sentences are separated by an empty line.

4.1.4 Named Entity Predictions

CRF is used for this stage, its output contains the fields Token and Predicted label. Since the data is large, CRF input is divided into 3 files and separately processed. With this, we conclude the preprocessing part and data can be loaded to the database and used for the main operations.

[5]https://github.com/ahmetaa/zemberek-nlp

4.2 Frequent Subgraph Mining

In order to define what a frequent subgraph is, we should start by defining support of a graph(or subgraph).
Let $S = \{G_1, G_2, \ldots G_n\}$ be a set of graphs and g be a subgraph appearing in at least one of the graphs in S. Then,

$$support(g) = \#\ of\ graphs\ in\ S\ that\ contain\ g\ as\ a\ subgraph$$

Frequent Subgraphs are subgraphs that have support over a certain threshold. The importance of finding these subgraphs is that we can infer the general structure of the data, inspect interesting patterns and process these subgraphs to generate rules.

As mentioned in Dataset section, we are working on news data. We store the news content in a graph database. A single news and its relationships with the words is taken to constitute a single graph, and we are able query and retrieve these graphs. Since we already have them in a graph structure, frequent subgraph mining can be directly applied.

gSpan is the algorithm used in our work to mine frequent subgraphs, given a set of graphs. gSpan applies depth first search and does not deal with candidate generation or false positive pruning [12]. After querying each graph from the database, they are converted to a format suitable as input to gSpan algorithm. We should mention that we are using the python implementation of gSpan which is available on github[6] with small modifications such as collecting the frequent subgraphs as they are found and being able to stop it's run method so that we can quickly get the results. The data format that this implementation of gSpan accepts is as follows:

$$\begin{array}{llll} t & \# & <graphID> & \\ v & <vertexID> & <label> & \\ & \vdots & & \\ e & <fromID> & <toID> & <label> \\ & \vdots & & \end{array}$$

where t indicates the start of a graph, v indicates a vertex and e indicates an edge. It should be noted that $vertexID$, $fromID$ and $toID$ represent ID's local to the graph, whereas *label* can be anything. We use the *label* attribute to specify it's ID relative to database. This is needed when generating queries as the local ID doesn't help us there. For edge *label*, we mapped as follows:

1 : CONTAINS
2 : IS
3 : FOLLOWED_BY

Subgraphs are contained in the gSpan object and we are able to get all the vertices and edges from the object. Then what is left is to generate the query for that subgraph using the *label* attributes.

4.3 Graph Similarity

Graph similarity methods are used to understand the correlation between the nodes and edges of two graphs. We have experimented with two different similarity algorithms, *Graph Edit Distance* (GED) and *Graph One Hot Encoding* (GOHE).

[6]https://github.com/betterenvi/gSpan

The reason why we included graph similarity to our project relies on the excessive amount of frequent subgraphs extracted by the *gSpan* algorithm from our textual data. When association rule mining applied on these subgraphs, we become aware of that a large number of poor-quality rules are generated. In order to eliminate these rules and reduce the number of rule set, we use graph similarity on frequent subgraphs and ignore the most of the similar graphs that can give us the same information.

Firstly, let's start with *Graph Edit Distance* algorithm. As stated in the work of Abu-Aisheh et al. [1], assume $G1 = (V1, E1, \mu_1, \zeta_1)$ and $G2 = (V2, E2, \mu_2, \zeta_2)$ be two graphs. The *Graph Edit Distance* between these graphs can be calculated as:

$$GraphEditDist(G1, G2) = \min_{e_1,\ldots,e_k \in \gamma(G1,G2)} \sum_{i=1}^{k} cost(e_i)$$

In the formula, $cost(e_i)$ denotes the cost of the edit operation e_i, which is a member of the set of edit paths, $\gamma(G1, G2)$, that transform $G1$ to $G2$. The edit operations consists of deletion, insertion, substitution operations for both nodes and edges. In our implementation, which is based on the GED implementation in *networkx*[7] Python library, we determined cost of each edit operation as 1 since all of the nodes and relationship types are unique in our graph model.

Our main goal was to eliminate some of the subgraphs that were too similar. A method we devised for this purpose is called *Average Distance*. In this method, we map our *gSpan* frequent subgraphs to *networkx* graph format one by one. Then, we calculate the average graph edit distance of each subgraph with all other frequent subgraphs.

$$GraphEditDistAvg(G_x) = \sum_{i=1}^{length(G)} \frac{GraphEditDist(G_x, G_i)}{length(G)}$$

In the formula above, G_x denotes any subgraph and $length(G)$ denotes the number of subgraphs in our subgraph set.

In order to define a threshold value, we measure the mean of these average graph edit distances.

$$GraphEditDistThresh(G) = \sum_{i=1}^{length(G)} \frac{GraphEditDistAvg(G_i)}{length(G)}$$

Lastly, we eliminate the subgraphs whose average graph edit distance value is below the threshold. Therefore, we end up with reduced number of subgraphs, which are less similar to each other, and apply association rule mining to this new subgraph set.

$$AboveThreshold(G_i) = GraphEditDistAvg(G_i) \geq GraphEditDistThresh(G)$$

$$SubgraphStatus(G_i) = \begin{cases} S & AboveThreshold(G_i) == T \\ D & AboveThreshold(G_i) == F \end{cases}$$

SubgraphStatus indicates whether the corresponding subgraph will be selected or discarded. S indicates selection, where D indicates discard. T and F letters stand for $True$ and $False$.

[7]https://networkx.github.io

Clustering the subgraphs is another approach to reducing the number of subgraphs. The idea is to cluster graphs together which have *GED* below some user specified threshold ratio. For a frequent subgraph, graph edit distance to every other frequent subgraph is calculated. The distances being small is an indication of similarity, however high distances are not informative, thus we should apply the same method to remaining subgraphs that are yet to be put in a cluster. Using this approach, we do not need to specify how many clusters there should exist. After generating the clusters, one can randomly sample any single subgraph from each of the clusters. We suggest choosing the maximal one (having the highest number of nodes and edges). The problem of both this method and Average Distance one is that they are inherently slow due to graph edit distance being computationally intensive and do not scale well to a large set of subgraphs.

The second algorithm for graph similarity is Graph One Hot Encoding. The aim of this algorithm is to embed each subgraph as a vector and apply a distance metric to the vectors to obtain the similarity. We use node labels which in our case correspond to named entities, to create the vocabulary over all the graphs. Then, we generate encodings for each of the subgraphs. These encodings are one hot[8], meaning that the vector's length is as big as the vocabulary and if a label exists in the subgraph, it's entry in the embedding vector is 1, else 0. We generate embeddings for every subgraph. Then the similarity among two subgraphs can be found by applying a distance metric (we use cosine distance) to their embeddings.

While it may take some time to construct the vocabulary and generate embeddings, the similarity calculation is very fast. This makes Graph One Hot Encoding a suitable choice as a similarity algorithm in our work. We still apply clustering method, but instead of calculating GED, we calculate GOHE to every other subgraph. The rest of the method is the same, we still aply a threshold, generate clusters and sample representatives from them.

4.4 Frequent Sequence Mining

News data is naturally progressive. One can inspect them daily, monthly or even yearly. In order to determine how to form the sequences, let us define two concepts related to time *group* and *window*.

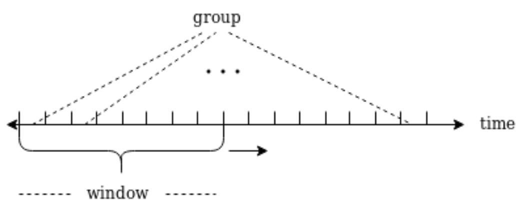

Figure 2: Group and Window

We divide our timeline into groups. Groups are the smallest unit that contain a group of news data. Each group will give us a sequence. For example, for weekly grouping, the

[8]https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html

news in that week form a sequence. Next, we will gather multiple groups in a window. Now, by only specifying a minimum support, frequent sequences can be mined for that specific window. The window is then shifted according to a granularity parameter and mining is applied again. This way, we end up considering multiple time windows separately. We should mention that, we use the implementation from pymining available on github[9].

4.5 Association Rule Mining

Association rules help us discover interesting patterns and relations among (frequent) subgraphs. A prediction or a recommendation system can be developed based on processing association rules.

A rule in the form $A \Rightarrow B$ where both A and B are itemsets is found to be true if it satisfies a minimum rule threshold. A useful threshold concept is confidence. Confidence is defined as follows:

$$confidence(A \Rightarrow B) = \frac{support(A \cup B)}{support(B)}$$

In this formulation $support(A \cup B)$ is the number of graphs that contain both subgraphs in A and subgraphs in B together.

Our goal is to mine association rules from the frequent subgraphs we have generated using gSpan. Thus, after mining the frequent subgraphs, we apply association rule mining based on the following proposition by Agrawal et al. [2]:

> To generate rules, for every large itemset l, we find all non-empty subsets of l. For every such subset a , we output a rule of the form $a \Rightarrow (l-a)$ if the ratio of $support(l)$ to $support(a)$ is at least minconf.

A naive implementation of this proposition needs to generate all the subsets, which does not scale well. In the same paper, there is a faster algorithm proposed, which we use to generate the rules given the subgraphs.

In our work, we use the one stated as the **Faster Algorithm**. This one makes use of downward closure property, very similar to how it has been used in Apriori, and makes fewer tests. For example, if a rule of the form $A \Rightarrow BC$ is found to be true, both $AB \Rightarrow C$ and $AC \Rightarrow B$ will also be true since $support(BC) \leq support(B)$ and $support(BC) \leq support(C)$.

In this algorithm, firstly the rules with one item consequents are found. Then, candidates having two items are generated according to Apriori (once again using the downward closure property) using the one item consequents. These item sets constitute our next consequents and algorithm continues generating larger consequents.

In our implementation, we have stayed loyal to the original work in terms of how the rules are generated and eliminated. The method *mineAssociationRulesFaster* expects 2 arguments, one is *support_where* which is the list of graph IDs that constitute the support for this subgraph and the second is $minconf$ which is the minumum confidence required in order to designate the found rule as true or not.

After mining frequent sequences, we derive rules from them if they satisfy the confidence threshold. An approach is to use Apriori once again to check each subset of the sequence. The downside of doing this is that we end up generating too many rules and some of those rules might end up getting generated multiple times as frequent sequences are similar among each other. Another problem is that Apriori should not be applied directly as it will not consider the order of the sequence. What we do instead is to randomly divide the sequence into two and let them be A and B if confidence requirement is satisfied. This way, we respect the order of the sequence and put a reasonable upper bound to the number of rules generated given the sequences. It is also possible to try more than one division. For example, after dividing the sequence into A and B, one could divide any of them again to get a obtain a smaller set. It is also possible to apply this division multiple times if you need to generate more rules.

Generating rules in this way makes sure that the relation between the antecedent (A) and the consequent (B) is an ordered one. Both A and B are ordered sets of subgraphs, also any subgraph of A precedes that of B.

4.6 Prediction

In prediction stage, we mine frequent sequences from our current data, apply graph similarity to these sequences and previously generated rules and predict a subgraph and news.

Firstly, we need rules. In a way, these rules serve as our training data. Remember that rules are of the form $A \Rightarrow B$ where A, B are sets of subgraphs. With the current data, we generate the frequent sequences. Then, for each sequence (Let A' denote a single one of such sequences), we try to find a similar antecedent of the training rules. Similarity is obviously in graph similarity terms. Then, upon finding a similar antecedent, we can predict the consequent of the rule that we have to be related with A'. We expect this relation to be in form of an association rule such as $A' \Rightarrow B$. Thus, we predicted the next sequence for the sequence we currently have. Although the subgraphs of these sequences might be very similar or even the same, the news that contained these subgraphs are different. So, we can recommend any (or all) of the news that we know to be containing B as a subgraph.

In Figure 3 we can see the pipeline of the whole method. Frequent subgraphs are mined, then reduced. Frequent sequences are mined from reduced subgraphs and rules are generated from past data. With the current data, we also mine frequent sequences, apply similarity to find a suitable rule to make a prediction.

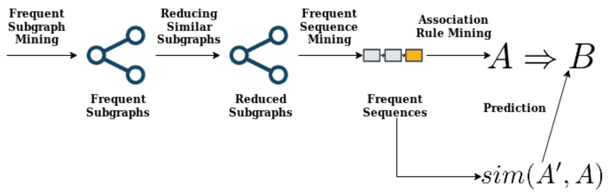

Figure 3: Pipeline of the Proposed Method

5. EXPERIMENTS

In this section, our experimental results are presented. We should mention that our data corresponds to exactly one year of news from 2015. We decided to mine rules from the first 11 months and perform prediction on December. This requires us to mine frequent subgraphs and frequent sequences for every month. Moreover, using the frequent

[9]https://github.com/bartdag/pymining

sequences, we mine association rules for the every month excluding December, as we will predict its rules.

In Table 2 various settings for the experiments can be found. Each month is listed with the following properties: number of news in that month, support value used to mine frequent subgraphs, number of subgraphs and number of subgraphs after they are reduced by similar ones pruned. We should also mention some values that are uniform throughout each month. Minimum number of nodes for subgraphs has been set to 7, cluster similarity ratio which defines how similar two graphs should be to get put in the same cluster has been set to 0.9 (90%), groups are daily, windows are weekly. If a sequence of subgraphs is present in 3 groups (days) out of 7, then it is has been declared as frequent. Lastly 0.8 is set as the minimum confidence for rule mining.

The reason for varying value of support and number of subgraphs is that we would like to have similar number of rules from each month so as not to have bias towards any month. These values are found after many trial and errors.

In Figure 4 we see the number of rules generated as we modify the similarity threshold. These rules are found by generating sequences from the first 11 months. Then, when a similar sequence to a rule antecedent is found from the last sequence, we check for a rule (confidence test). Obviously, as we increase the threshold, less rules are generated. A note in efficiency is that the rules from past months are stored as a dictionary with keys being the antecedents and values being a list of consequents. This way, we end up making less similarity tests.

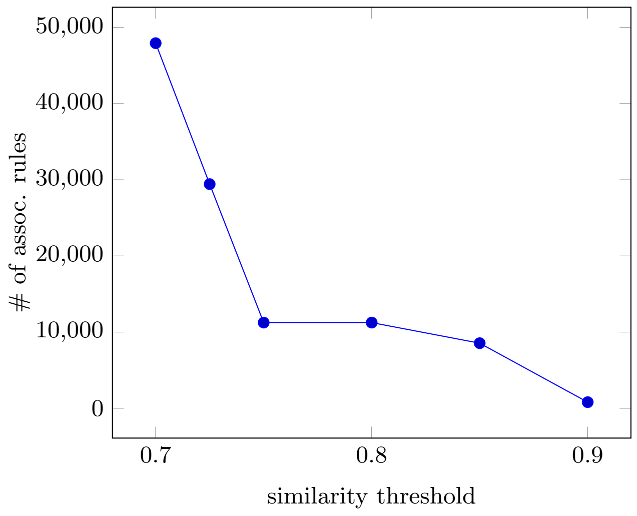

Figure 4: Conf. Threshold vs. # of Assoc. Rules

In Figure 5 we compare the different graph similarity methods we have used in an attempt to eliminate similar subgraphs. GED Avg. Dist. is the method where we calculate graph edit distances among each subgraph, find *GraphEditDistThresh* eliminate the ones that have average below this threshold. GED Clustering is the method where we calculate the GED for each subgraph to each subgraph and cluster the ones that have a small edit distance together. Lastly, GOHE Clustering encodes the subgraphs as One Hot Vectors and calculates cosine distance to find similarity. Similar to GED clustering, similarities to all subgraphs are found and clustered. We notice right away that GOHE Clustering is much faster compared to GED based algorithms. GED is a computationally intensive task and because the method grows in $O(n^2)$, the time spent to reduce the subgraphs becomes intolerable.

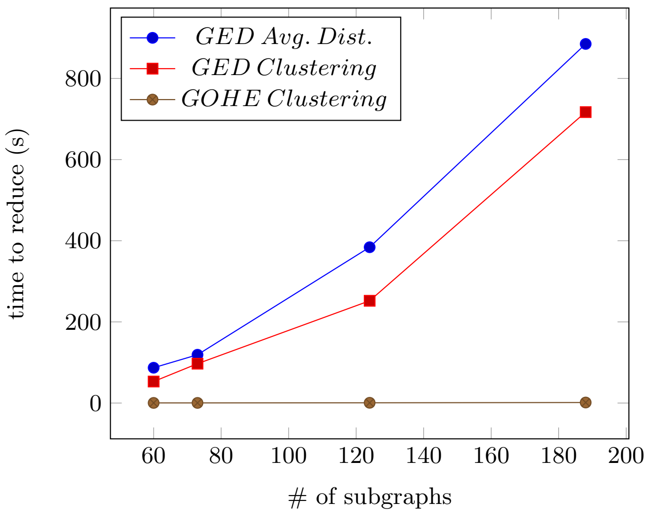

Figure 5: # of Subg. vs. Reduction Time (s)

In Figure 6, we see the results of reducing the subgraphs. We want to get rid of very similar graphs and represent them with a single graph. Since GED is very precise, we can take it as a baseline. We see that GED Avg. Dist. fails to reduce graphs to their representatives, but still able to eliminate some. GOHE Clustering actually eliminates more graphs compared to GED Clustering at the same similarity ratio for clusters (each graph in a cluster is 90% similar). This indicates that there might be some loss of subgraphs. Nevertheless, GED Clustering and GOHE clustering has similar results.

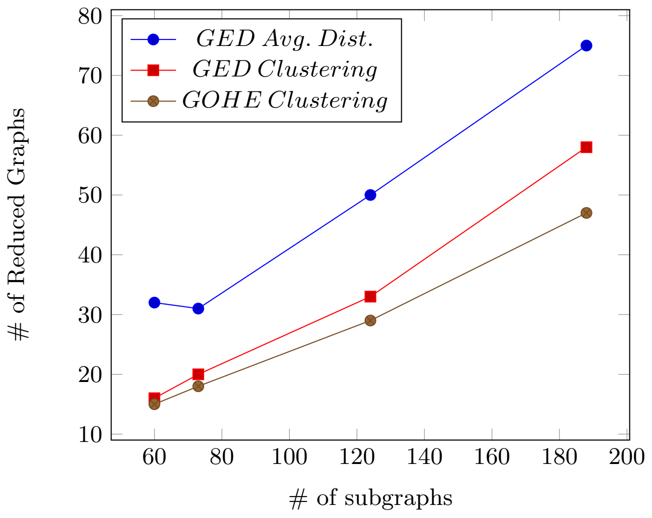

Figure 6: # of Subg. vs. # of Reduced Graphs

For our experiments, a machine with Ubuntu 18.04 OS has been used which has Memory of 3.6 GiB and CPU of Intel Core i5-7200U 2.50GHz x 4.

Month	News	Supp.	Subgraphs	Reduced
January	895	10	124	29
February	831	13	907	40
March	852	9	154	37
April	955	16	20	6
May	1016	10	33	9
June	1077	12	72	16
July	949	12	76	24
August	874	10	107	21
September	828	9	98	19
October	1015	9	176	38
November	1394	13	145	32
December	1308	18	67	15

Table 2: Settings of the Experiments

6. CONCLUSION

In this work, we propose a graph mining library that has the ability to mine frequent subgraphs, apply graph similarity methods to cluster subgraphs and reduce frequent subgraphs into particular representatives. Mine frequent sequences and generate association rules from them, which can be used to make predictions about news patterns we might expect to see in the future and recommend similar news from past.

Among the graph similarity methods, we found out that Graph Edit Distance is precise but very slow. It does not scale well to large sets of subgraphs. The faster alternative is to use Graph One Hot Encoding. Then similar graphs can be clustered together and representatives of clusters can be chosen.

The patterns predicted might be very similar or essentially the same as the patterns tested ie. antecedent and consequent of the rule can be very similar. In those cases, prediction doesn't end up being too useful, however, recommendation of a news can still be useful.

As further work, there might be some changes to the graph structure. For example, during subgraph mining, named entity tags move and cause almost the same graph to be mined once again.

7. REFERENCES

[1] Z. Abu-Aisheh, R. Raveaux, J.-Y. Ramel, and P. Martineau. An Exact Graph Edit Distance Algorithm for Solving Pattern Recognition Problems. In *4th International Conference on Pattern Recognition Applications and Methods 2015*, Lisbon, Portugal, Jan. 2015.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of 20th Intl. Conf. on VLDB*, pages 487–499, 1994.

[3] B. Cao, Y. Li, and J. Yin. Measuring similarity between graphs based on the levenshtein distance. volume 7, pages 169–175, 2013.

[4] Y. Goldberg and G. Hirst. *Neural Network Methods in Natural Language Processing.* Morgan & Claypool Publishers, 2017.

[5] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, pages 1–12, New York, NY, USA, 2000. ACM.

[6] D. S Rajput, R. S Thakur, and G. Thakur. Rule generation from textual data by using graph based approach. 03 2019.

[7] A. Sanfeliu and K.-S. Fu. A distance measure between attributed relational graphs for pattern recognition. volume SMC-13, pages 353–362, 1983.

[8] A. Schenker, M. Last, H. Bunke, and A. Kandel. Clustering of web documents using a graph model. 12 2003.

[9] G. A. Seker and G. Eryigit. Initial explorations on using crfs for turkish named entity recognition. In *COLING*, 2012.

[10] V. Tiwari, V. Tiwari, S. Gupta, and R. Tiwari. Association rule mining: A graph based approach for mining frequent itemsets. In *2010 International Conference on Networking and Information Technology*, pages 309–313, June 2010.

[11] J. T. Wang, K. Zhang, and G.-W. Chirn. Algorithms for approximate graph matching. *Information Sciences*, 82(1):45 – 74, 1995.

[12] X. Yan and J. Han. gspan: graph-based substructure pattern mining. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 721–724, Dec 2002.

[13] D. Zhou, J. Huang, and B. Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. volume 19, pages 1601–1608, 01 2006.