INFORMATION EXTRACTION FROM NEWS RELATED TEXTS USING GRAPH MINING TECHNIQUES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

RECEP FIRAT ÇEKINEL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

AUGUST 2020

Approval of the thesis:

INFORMATION EXTRACTION FROM NEWS RELATED TEXTS USING GRAPH MINING TECHNIQUES

submitted by **RECEP FIRAT ÇEKINEL** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering** _____

Prof. Dr. Pınar Karagöz
Supervisor, **Computer Engineering, METU** _____

Examining Committee Members:

Assoc. Prof. Dr. İsmail Sengör Altıngövde
Computer Engineering, METU _____

Prof. Dr. Pınar Karagöz
Computer Engineering, METU _____

Assist. Prof. Dr. Burcu Yılmaz
Institute of Information Technologies, Gebze Technical University _____

Date:

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Recep Fırat Çekinel

Signature :

ABSTRACT

INFORMATION EXTRACTION FROM NEWS RELATED TEXTS USING GRAPH MINING TECHNIQUES

Çekinel, Recep Fırat
M.S., Department of Computer Engineering
Supervisor: Prof. Dr. Pınar Karagöz

August 2020, 79 pages

The increase in data availability and the progress in natural language processing provide a basis for the development of new techniques for information extraction. Event detection from textual content using the text mining concepts is a well-researched field in the literature. However, graph embedding techniques in recent years provide an opportunity to represent textual contents in graphs because texts can be enriched with additional attributes in graphs, and the complex relationships within graphs can be modeled better. In this thesis work, graph-based representations of textual resources such as news are examined for pattern extraction, and a method is proposed for news prediction. As the first step, graph representation techniques are investigated. Afterward, frequent subgraph mining and sequential rule mining algorithms are applied. We consider that subgraphs contain the main story of the contents, and sequential rules indicate the subgraph patterns' temporal relationships. Finally, the sequential patterns are used for recommendation according to their similarity scores. In order to measure the similarity, various graph embedding techniques are also examined.

Keywords: Graph Mining, Sequential Rule Mining, Frequent Subgraph Mining, Graph Embeddings, News Prediction

ÖZ

ÇİZGE MADENCİLİĞİ TEKNİKLERİNİ KULLANARAK HABER İLE İLGİLİ METİNLERDEN BİLGİ ÇIKARIMI

Çekinel, Recep Fırat
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Pınar Karagöz

Ağustos 2020 , 79 sayfa

Veri kullanılabilirliğindeki artış ve doğal dil işlemedeki ilerleme, bilgi çıkarma için yeni tekniklerin geliştirilmesine temel oluşturmaktadır. Metin madenciliği konseptlerini kullanarak metinsel içeriklerden olay tespiti, literatürde iyi araştırılmış bir alandır. Bununla birlikte, son yıllarda çizge vektörleri teknikleri, çizgelerde metinsel içerikleri temsil etme fırsatı sunar, çünkü metinler çizge yapısında ek niteliklerle zenginleştirilebilir ve çizgelerle karmaşık ilişkiler daha iyi modellenebilir. Bu tez çalışmasında, metinleri çizge yapısında temsil ederek örüntü çıkarımı üzerinde çalışılmış olunup haber tahminlemesi için yöntem geliştirilmiştir. İlk adım olarak çizge gösterim teknikleri incelenmiştir. Daha sonra sık alt çizge madenciliği ve ardışık zamanlı kural madenciliği algoritmaları uygulanmıştır. Bunun nedeni, alt çizgelerin haberlerin içeriğinin ana hatlarını içermesi ve sıralı kuralların alt çizgelerin zamansal ilişkilerini göstermesidir. Son olarak, sıralı örüntüler benzerlik puanlarına göre öneri için kullanılmıştır. Benzerliği ölçmek için çeşitli çizge vektörleri teknikleri incelenmiştir.

Anahtar Kelimeler: Çizge Madenciliği, Ardışık Zamanlı Kural Madenciliği, Sık Alt-çizge Madenciliği, Çizge Vektörleri, Haber Tahmini

To my family

ACKNOWLEDGMENTS

To begin with, I want to express my gratitude to my supervisor Prof. Dr. Pınar Karagöz, for her valuable guidance and persistent encouragement. I would also thank my thesis committee members, Assoc. Prof. Dr. İsmail Sengör Altıngövde and Asst. Prof. Dr. Burcu Yılmaz for their valuable contribution.

I owe my deepest gratitude to our project partners, Asst. Prof. Dr. Alev Mutlu, Can Eroğul, and Murat Yükselen for sharing their knowledge and experience. I am also grateful to Mustafa Ağrıman, Mert Erdemir, and Ege Çıklabakkal for their advice and comments. Additionally, I am thankful to my roommates Burak, Kadir, Abdullah, and Fatih, for their encouraging and supportive attitudes.

And finally, I would like to offer my special thanks to my parents and my lovely sister Beyza Çekinel for their unconditional support and love. Without them, it would not have been possible to accomplish this.

This work (No:117E566) is funded by the Scientific and Technological Research Council of Turkey (TUBITAK).

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ALGORITHMS

ALGORITHMS

LIST OF ABBREVIATIONS

ABBREVIATIONS

NLP	Natural Language Processing
CRF	Conditional Random Fields
NER	Named Entity Recognition
POS	Parts-of-speech
DFS	Depth-First Search
GCN	Graph Convolutional Networks
GPAR	Graph-Pattern Association Rules
GTAR	Graph Temporal Association Rules
FSM	Frequent Subgraph Mining

CHAPTER 1

INTRODUCTION

1.1 Motivation and Problem Definition

In recent years, advancements in computer science have created a basis for many applications, and the amount of data that needs to be processed is getting larger every day. With the utilization of computing power and the existence of large datasets, we can extract hidden patterns within the unstructured data such as texts.

Event detection is a task that tries to find incidents that happened in the specified time interval and attracts people's attention. Within the scope of this work, we aim to explore patterns or events, such as central bank meetings, sports organizations, weather forecasts, etc. from the previous news. In other words, we aim to find the patterns or causal relationships among news and predict what type of news may happen as the consequent. However, our model cannot predict a news related to an unexpected event such as the Covid-19 pandemic, a plane crash, etc. Instead, we predict main story for a candidate news by considering the previous patterns and try to discover similar cases. Besides, unlike the conventional approaches, working on the textual resources directly, we represent each news in the form of graph and generate a news prediction model that accepts graph structure as the input and executes graph analytics algorithms to extract relationships between graphs. This is because encoding a text on a graph may provide a better basis since graphs are proven to be a suitable data structure for representing textual data since there are many hidden relationships within texts [2]. We can also enrich the graph structure by introducing additional properties such as the date of the published article, author of the article, etc. Besides, we can extract the latent representation of a graph in several granularities, including

node-level, subgraph-level, and graph-level vectorial representations. These vectors are useful for several machine learning tasks. For instance, we can measure the similarity of two articles by considering their vectorial representations. Therefore, by using graph-based approaches, hidden relationships within texts can be examined successfully.

1.2 Proposed Methods and Models

In this thesis, our goal is to develop techniques for news prediction on graphs that correspond to texts. First of all, we research the literature to encode texts on a graph structure. Afterward, we extract frequent subgraphs because repeated subgraphs may be an indicator for an event and subgraphs are the core components of graphs. Subsequently, we form a subgraph sequence by ordering them according to the subgraphs' occurence time and apply sequential rule mining algorithms to discover sequential rules. This is because sequential rules demonstrate how patterns evolve in time. Finally, we devise a rule-based recommendation system to predict what type of news may occur in the future using our sequential rules. Since we use graph structure as the primary data source, we extract graph embeddings using several graph embedding algorithms and measure the similarity of these embeddings for the prediction.

Our model consists of graph representation, frequent subgraph mining, sequential rule mining, subgraph embedding, and prediction modules. Extracting frequent subgraphs and forming sequential rules from subgraph sequences are challenging tasks since the time complexities of these techniques are very high.

1.3 Contributions and Novelties

To the best of our knowledge, it is the first sequential rule-based model that predicts the core of an upcoming news by representing them in the form of graph. Our contributions are as follows:

- To the best of our knowledge, there is a very limited work that represents texts in the form of graphs for prediction tasks

- Ordered subgraph sequences are formed and sequential rules are explored for prediction
- Several graph and subgraph embedding methods are applied to calculate the similarity between subgraphs
- A generic subgraph set is predicted as the output of our model. Our model's prediction illustrates the main story of news which will take place within a predetermined time window

1.4 The Outline of the Thesis

The rest of the thesis is organized as follows. The literature survey and the related studies are presented in Chapter 2. The mathematical and technical concepts of the proposed model are presented in Chapter 3. In chapter 4, the dataset, data preprocessing, and the proposed graph models are explained. The methodology and the pipeline of this study are described in Chapter 5. Experiments and results are presented in Chapter 6. Finally, Chapter 7 covers the conclusion.

CHAPTER 2

RELATED WORK

In this chapter, recent graph-based approaches applied to several tasks, including event detection, text categorization, and graph association pattern mining techniques are summarized.

2.1 Graph Representation

Genc et al. [3] propose a graph-based model to extract information from social media data to detect events. First of all, they introduce a novel way to represent textual data such as tweets in the form of a graph. Their graph model contains a unique node for each tweet connected to the first token of the tweet. The graph consists of a chain of tokens where there is a link between consecutive tokens. Since the graph structure is very similar to a snake, they called "SnakeGraph", this model in the rest of the paper. They enrich the SnakeGraph by introducing the date information in various granularities. To be more specific, year, month, and date information are also embodied in the graph model as a node for each of them. They call this approach the "hierarchy of dates". Date nodes and the corresponding tweets have an edge to maintain the relationship. After constructing a graph model from a textual source, the node2vec [4] is applied to the graphs to generate node embeddings for each node. At the final stage, k nearest word embeddings are extracted for each date embedding using the cosine similarity. After that, the association between the most similar words and the events are evaluated by a human inspector. According to the experiments, they report 72.9% of the most similar words have a close relationship with an event.

Erdemir et al. [5] discuss the query performance of hyper-graph based and graph-

based models on Neo4j. They represent a book dataset in the form of graphs. In the graph-based model, there were nine types of nodes and five types of relationships. There is an edge between each book with its publisher, author, published year and category nodes. On the other hand, in the hyper-graph model, hyper nodes and edges are also introduced to cover some attributes (i.e. <author, book>) in a single hyper node. The query performance of these graph models was evaluated on 12 queries. According to the findings, the hyper-graph model requires more storage space but works more effectively on queries that retrieve multiple types of nodes. However, for the other queries, the former graph model's access cost is lower.

Rousseau et al. [6] propose a novel approach for the text categorization problem by considering it as a graph classification problem. First of all, they represent each document as a graph that is named "graph-of-words". This graph's nodes consist of unique terms that exist in the document, and the edges exist between terms that appear within a fixed-size sliding window. In the second stage, they extract features from graphs by applying several methods that are given to a graph classifier as an input for training. Since their graph structure captures the co-occurrence information, they extract frequent subgraphs by applying the gSpan [7] algorithm. They consider each subgraph a soft n-gram where nodes of a subgraph occur within a fixed-length sliding window. They point out that the frequent subgraph mining extracts the excessive amount of subgraphs. Therefore, they also propose an alternative solution that extracts the main core of each graph first. Then apply the frequent subgraph mining procedure to extract subgraphs. At the final stage, they execute the SVM algorithm in different scenarios. They state that mining frequent subgraphs and feeding them as features for the SVM is performed best on almost all datasets. They also mention that extracting the main core accelerates the process and perform best on some datasets.

Schenker et al. [8] propose a novel graph-based technique to cluster web documents. First of all, each web document is represented as a graph where each term is assigned to a unique node, and there is an edge directed from a vertex toward the following immediate node. The graph model contains three classes of edges: title, link, and text. If the terms appear in the title, the corresponding edge is labeled as title. If the terms exist in the URL link section, the corresponding edge is annotated as the link. Otherwise, it is classified as text, which implies that both of the terms are present in the

document's textual content. In the second phase, they adopt the k-means algorithm to cluster very similar graphs. At the final phase, the experimental results of their model are compared with the vector-based k-means algorithm, and they state that the proposed model outperforms the baseline model. The reason behind this phenomenon is that a graph representation captures the structural information such as the index of the terms and the sequential order of the terms. However, these kinds of information are lost in the vector-based representation.

2.2 Graph-pattern Association Rules

The association rules within the graphs are started to be studied recently and attracted many researchers' attention in a short time. Because graph patterns are beneficial to detect the associations between the parts of graphs. One of the main differences in the papers for GPARs is the type of graph-patterns that are going to be extracted.

GPARs are introduced by Fan et al. [9] to explore associations within social graphs. They define a new problem called Diversified Mining Problem and devised an algorithm to discover top k GPARs. Moreover, they suggest an algorithm to detect potential customers using GPARs. They discover the rules in the form of $Q(x, y) \rightarrow q(x, y)$ where $Q(x, y)$ is a graph pattern and $q(x, y)$ is an edge. In other words, the proposed model discovers patterns whose consequents contain only one edge. Finally, scalability of the proposed model is verified on two real-life datasets.

Namaki et al. [10] introduce Graph Temporal Association Rules (GTAR) to discover streaming graph patterns. In the proposed model, events are illustrated as graph patterns, and the association between the temporal events within a pre-defined time window is discovered. A joint algorithm handles the processes of event mining and rule discovery. Finally, scalability and efficiency of the GTAR algorithm and the graph rules are verified on real-life and synthetic datasets.

Wang et al. [11] propose a novel technique to extract generic GPARs. They decompose the process of mining the GPARs in two subtasks: frequent pattern mining and rule generation. For the extraction of frequent patterns, they develop a distributed and optimized algorithm called FPMiner. After generating the frequent subgraphs,

they generate rules in the form of $Q_r \rightarrow Q_l$, where both Q_r and Q_l can be any graph patterns and share at least one common node but no edge in common. Moreover, both the antecedent and the consequent contains at least one edge. The proposed method's performance is evaluated on three real-life datasets. Accuracy and precision of the proposed model are also presented on the link prediction task. They state that the model's average accuracy on the datasets is 43.8%, which is far better than the baseline, Jaccard and SimRank, models' accuracy.

Our work differs from these works in some aspects. First of all, our model discovers sequential patterns within textual contents. As a result, our primary data source is texts which make the task more challenging. Additionally, our model explores sequential patterns instead of association rules. Sequential patterns consist of an ordered list of items, so temporal relationships play a crucial role in our task.

2.3 Graph Embeddings

Graphs don't have a grid structure like an image, so applying machine learning and deep learning algorithms on a graph is challenging. In recent years because of the advancements in this field, many machine learning-based techniques are proposed to represent graphs in a lower-dimensional space.

The conventional way for information extraction from graphs is extracting handcrafted features [12]. Although these methods scale well, they may output some pairs that do not seem similar. Graph kernels [13] measure the similarity between graph pairs using kernel function. The main disadvantage of applying these methods is that they are time-consuming.

There are also other approaches to map structural information of a graph (i.e., nodes, subgraphs, entire graph) in a latent vector space. One of the most popular embedding techniques is called "shallow embedding" [14] that consists of an encoder and a decoder. The reason for this naming convention is that the encoder function is just an embedding lookup.

Random walk approaches such as DeepWalk [15] and node2vec [4] are considered as

shallow node embedding techniques. The shallow embedding methods' main limitation is that they do not share any parameters, so it is not possible to encode graphs that have not observed during training. The random walk methods try to maximize the probability of visiting a vertex v_j from an initial vertex v_i, $p(v_j|v_i)$ if v_j is visited by a walk of length T. Since optimization process of embeddings for each vertex at each step is very costly, these algorithms use negative sampling and update only small percentage of the weights of negative examples (vertices) at each iteration. struc2vec [16] introduces the structural identity of nodes while generating node embeddings. In other words, the structural similarity of nodes and their vectorial representations are correlated.

Other than node embeddings, we can also represent the whole graph or subgraph in a single vector. Sub2vec[17] uses biased random walks to learn the latent representation of an arbitrary subgraph. It provides two properties, namely neighborhood and structural, to learn subgraph embeddings. While the neighborhood property captures the local connectivity of any graph, the structural property captures structural information such as the degree of nodes. Subgraph2vec [18] learns the latent representation of rooted subgraphs. The framework uses the Weisfeiler-Lehman isomorphism test [19] to capture the structural equivalence information. Graph2vec [20] represents the whole graph in a single vector. It is based on doc2vec [21] where each graph is considered as a document and rooted subgraphs as words. Using the skip-gram model, graphs are represented in a latent vector.

More recently, graph neural networks such as GCN [22] , GraphSAGE [23] are proposed to aggregate local neighborhoods and encode structural information. At each layer, neighborhood information is aggregated that can be any function such as mean, max-pool, or LSTM [14]. Deep graph models are inherently inductive. Hence, they share hidden parameters, so they capture the encoding of a graph that they have not seen during training. Graph neural networks suffer as the model's depth increases because they could not sufficiently represent training data. Furthermore, deep models tend to flatten the node representations. To solve these issues, Zhang et al. [24] inspired by BERT [25], a transformer-based deep language model, and adopted the idea of transfer learning into graphs. Instead of word tokens, Graph-Bert samples nodes with their context. They called this structure as linkless subgraphs. Graph-Bert

is pre-trained on node attribute reconstruction and graph structure recovery tasks and fine-tuned on node classification and graph clustering tasks. Scalability of the node embeddings is a challenging issue. Lerer et al. [26] propose a distributed model, Pytorch-BigGraph, for huge graphs to find node embeddings by partitioning nodes and edges and executing threads simultaneously.

CHAPTER 3

PRELIMINARIES

Technical details about methods, paradigms, and algorithms used in this thesis are presented with the formulas and figures in this section. The thesis is based on the background information below.

3.1 TF-IDF Term Weighting

TF-IDF is often used in data mining tasks because it indicates each word's importance in a corpus [27]. The TF-IDF metric is the product of two statistics, term frequency and inverse document frequency. In our model, normalized term frequency is calculated using the formula given in Equation 3.1:

$$tf(t,d) = 0.5 + 0.5 * \frac{f_{t,d}}{max(f_{t',d} : t' \in d)} \tag{3.1}$$

The term frequency is directly proportional to the number of times each word passes in documents. On the other hand, the inverse document frequency gives more credit to rare words. The idf score is measured using the formula given in Equation 3.2.

$$idf(t,D) = log(\frac{N}{1 + |d \in D : t \in d|}) \tag{3.2}$$

where N is the number of news in the dataset and $|d \in D : t \in d|$ indicates the number of news that contains t. We measure TF-IDF score of each token in the whole dataset using the formula given in Equation 3.3. Then, terms with low TF-IDF scores are

eliminated because the TF-IDF score can be used as an indicator of significance.

$$tfidf(t, D) = max(tf(t, d) : d \in D) * idf(t, D) \tag{3.3}$$

3.2 Sequential Rule Mining

Sequential rule mining is a subfield of data mining that concerns the extraction of interesting rules from a transactional database. The transactional database consists of sequences where a sequence is formed by a group of items that is called as *itemset*. Sequential patterns are the subsequences that exist in a sequence database. A sequence consists of an ordered list of itemsets where each itemset has an unordered list of items. In other words, there is a time-based order between each itemset, but items in an itemset are not ordered.

Sequential rule mining is a task of discovering rules from a sequence database [28]. A sequential rule is in the form of $A \rightarrow B$ where A and B consist of itemsets. The left-hand side of the rule is called as *antecedent*, and the right-hand side of the rule is known as *consequent*. Note that, both A and B are frequent sequences found as sequential patterns by a sequential pattern mining algorithms. Also, it implies that B happens after A, so there is a temporal relationship and it indicates a strong association between A and B. Sequential rules are extracted according to support, confidence, and lift measures.

$$support(A) = number\ of\ sequences\ that\ contain\ A \tag{3.4}$$

$$confidence(A, B) = \frac{support(A \cup B)}{support(A)} \tag{3.5}$$

$$lift(A, B) = \frac{confidence(A, B)}{support(B)} \tag{3.6}$$

- *Support* measures the relative frequency of a sequence that exists in a sequence database.

- *Confidence* measures the likelihood of occurrence of B after A. Higher values of confidence imply the high association between A and B.
- Sequential rules with high confidence and high support may not be so interesting. *Lift* is an indicator of interestingness. Rules with high lift values (lift > 1) are considered as more interesting than rules with low lift values (lift < 1). Because high lift values imply that there exists strong dependency between the antecedent and the consequent. If the lift value is equal to 1, the rule happens coincidentally.

The sequential rule mining can be decomposed into two subtasks:

1. **Sequential pattern mining:** It discovers sequential patterns that frequently exist in a transactional database. Lots of sequential pattern mining algorithms are proposed such as AprioriAll [29], GSP [30], PrefixSpan [31], SPADE [32], CM-SPAM [33] and CM-SPADE [33] to find all frequent sequences in a sequence database above the user-defined support value, *minsup*. These algorithms can be classified into three groups. The first group of algorithms store sequences in horizontal format and use the Apriori property while pruning the candidate subsequences. AprioriAll [29] and GSP [30] are in this category. The second class of algorithms store sequences vertically. In other words, the sequence ID list for each item is stored in the database. SPADE [32], CM-SPAM [33] and CM-SPADE [33] belong to this class. The algorithms in the first two categories generate candidate patterns while scanning the database, which is a costly operation. However, the pattern-growth based algorithms avoid this by scanning the database in a depth-first search manner. The PrefixSpan [31] algorithm is one of the most popular pattern-growth based sequential pattern mining algorithm, and we executed the PrefixSpan algorithm to discover frequent subsequences.
2. **Sequential rule generation:** It performs the extraction of interesting rules by using the frequent sequences that were discovered in the first step. The rules are generated if the confidence of that rule is above the user-defined *minconf* value. We can also eliminate those rules that do not satisfy some other criteria such as lift.

3.2.1 PrefixSpan Algorithm

PrefixSpan [31] algorithm is a pattern-growth based frequent sequential pattern mining algorithm that discovers the frequent sequences recursively starting from sequences that consist of a single item and discovers larger sequences using a depth-first search. At each pass, it projects a database for the given prefix and explores the frequent patterns. The algorithm halts when no search space needs to be explored. The PrefixSpan algorithm does not require any candidate generation. Instead, it generates projected databases for each prefix, and as the prefix size increases, the projected database shrinks. However, constructing a projected database is a costly operation.

3.2.2 RuleGen Algorithm

The RuleGen [32] algorithm extracts all sequential rules having confidence above a threshold, *minconf*, that is adjusted by users. The RuleGen algorithm outputs sequential rules such as A $\rightarrow$ B where A is a subsequence of B. Note that, A and B must be found as a frequent sequence by a sequence mining algorithm such as the PrefixSpan.

3.3 Frequent Subgraph Mining

Frequent subgraph mining is a task that finds all the subgraphs within the collection of graphs that occur more than the user-defined support threshold, *minsup* [34]. Frequent subgraph mining is a challenging task because there may be some subgraphs that have structurally identical even though they may seem different. This phenomenon is called *subgraph isomorphism*, which is known to be an NP-complete problem.

There are various kinds of approaches to find frequent subgraphs such as:

- Apriori-based algorithms: FSG [35], AGM [36]
- Pattern-growth based algorithms: gSpan [7], MoFA [37], FFSM [38], Gaston [39]
- Partial FSM algorithms: Subdue [40], TKG [41]

Apriori-based algorithms generate candidate subgraphs similar to Apriori-based frequent itemset mining algorithms. In addition, they perform a level-wise search to find all frequent subgraphs. Therefore, a join operation has to be done for joining two k-size subgraphs, which is a costly operation. On the other hand, pattern-growth based algorithms do not need join operation. Instead, they try to add an edge to the existing subgraph at each iteration so that the same subgraph can be discovered multiple times. Finally, there also exist some algorithms that discover a fraction of the frequent subgraphs. For instance, the Subdue [40] uses the beam search to decide k best children that should be expanded to find frequent subgraphs. TKG [41] is another such algorithm that finds the most frequent k subgraphs where k is the only parameter set by the user.

3.3.1 gSpan Algorithm

Our model uses the gSpan [7] algorithm, which utilizes DFS codes and lexicographical ordering to eliminate redundant candidate generation and costly graph isomorphism test. The main purpose of using DFS codes is documenting edges and vertices in a tabular form. DFS codes represent edges as 5-tuple:

$$(v_i, v_j, l_{v_i}, l_{v_i,v_j}, l_{v_j})$$

where v_i indicates the initial node, v_j indicates the ending node, l_{v_i} and l_{v_j} are the label of these nodes respectively. l_{v_i,v_j} shows the label of the edge. The gSpan algorithm finds frequent subgraphs using a bottom-up approach. The gSpan algorithm prunes subgraphs whose DFS codes are not minimal. A DFS code is called as *min_DFS* if it is based on linear (lexicographical) order. A DFS code should hold one of the following conditions to be in a linear order:

1. $(u, v) \prec (u, w) \rightarrow v < w$
2. $(u, v) \prec (v, w) \rightarrow u < v$
3. $e_1 \prec e_2, e_2 \prec e_3 \rightarrow e_1 \prec e_3$

The gSpan algorithm produces the same subgraph several times by starting different vertices each time. For instance, in Figure 3.1, three different DFS trees for (a) are illustrated. To handle this problem, the gSpan introduces the rightmost path extension strategy. This strategy allows the insertion of new vertex only on the rightmost path. Moreover, a backward edge can also be traversed from the rightmost path.

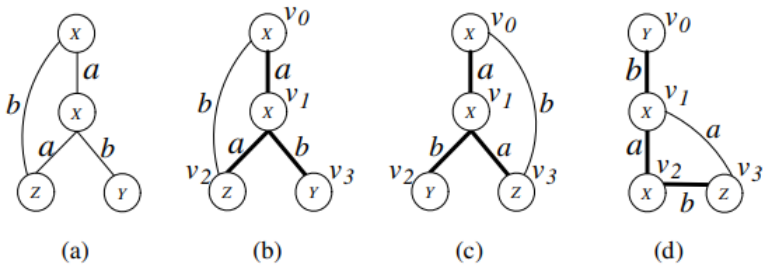


Figure 3.1: DFS Tree [1]

CHAPTER 4

DATASET AND GRAPH MODELS

4.1 Dataset

The dataset contains news from an online Turkish newspaper that is known as "Dünya Gazetesi" [1]. The newspaper covers financial and business-related news mostly, and our dataset contains 32734 news in total originated from 01.01.2015 to 31.12.2016. Each record includes the following attributes:

- unique news_id
- title
- first paragraph
- content

In all our experiments, we utilize the textual content. News are grouped in 27 categories by the newspaper, and the statistics about the dataset are presented in Figure 4.1 and Figure 4.2.

4.2 Preprocessing

Turkish is a morphologically rich and agglutinative language. Therefore, NLP tasks in Turkish are very challenging. We use open-source Zemberek [2] library for the

[1] https://www.dunya.com/
[2] https://github.com/ahmetaa/zemberek-nlp

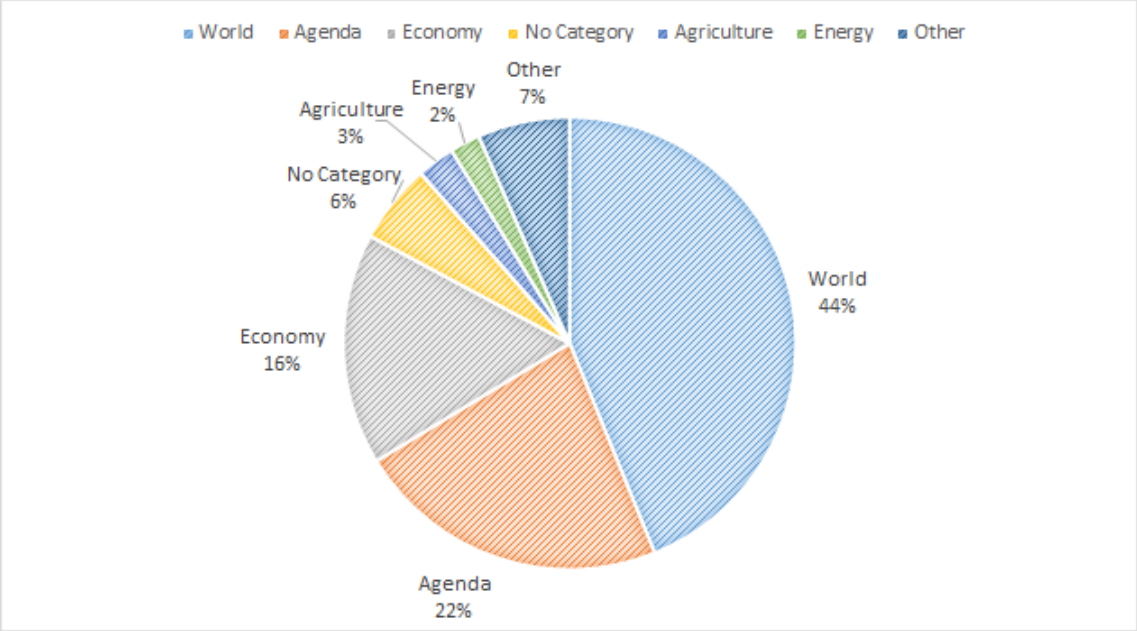


Figure 4.1: Distribution of categories in the news dataset

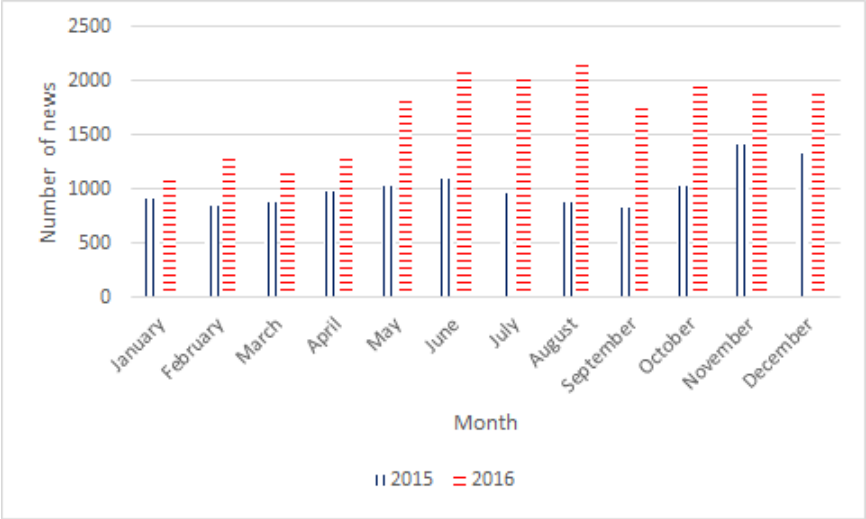


Figure 4.2: Number of news in each month

preprocessing phase because it provides rich natural language processing modules for Turkish. In this step, we apply the following processes to clean the text respectively:

- Data cleaning: Removal of all non-unicode characters, URLs and emojis
- Tokenization: Zemberek is able to split sentences into tokens that are seperated by a single space character

- Disambiguation: In Turkish, a word may have different meanings, so we need to disambiguate sentences to find the correct meaning of each token. Zemberek's morphology module provides this mechanism.

- Lemmatization: Lemmatization is a task that groups words with the same meaning by converting the inflected words into a singular form in the present tense. Zemberek's morphology module also provides several methods for lemmatization.

4.3 Graph Models

A graph is defined as G = (V, E) where V represents a set of the vertices, and E represents a set of the edges. The textual content of each news is represented as a graph whose nodes consist of lemmatized terms and a unique node for each news. We formed three graph models by using noun phrases and verbs, named entities, and TF-IDF scores.

4.3.1 Noun-Verb Graph Model

The graph of words is a common approach in the literature to represent textual content as a graph. However, scalability related problems may arise for large datasets. Therefore, we suggest that representing noun phrases and verbs might be sufficient to describe the text. For this purpose, POS tags of all articles are extracted using Zemberek's POS tagger, and we eliminate all terms except noun phrases and verbs. The graph contains two types of nodes and two types of relationships.

News nodes exist for each news, and they correspond to a specific article. Additionally, there exist *Word* nodes for all unique nouns and verbs. In addition, *Contains* relationship links *News* nodes with *Word* nodes that are included in the context of the article. It is a directed edge that is originated from *News* towards *Word* node. *Word* nodes are also connected to each other with the *Cosine_Sim* relationship. The link exists between two *Word* nodes if cosine similarity of their Word2vec [42] vectors is above the user-specified threshold. Because it indicates that the words are highly as-

sociated and they are semantically related to each other. An illustration of this graph model is presented in Figure 4.3.

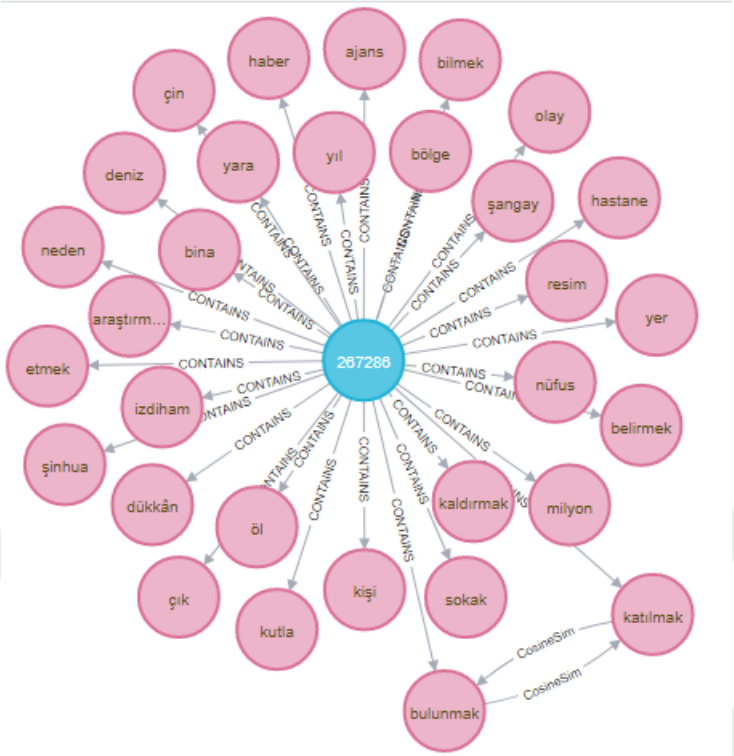


Figure 4.3: Noun-verb graph model

4.3.2 Named Entity Graph Model

Named entity recognition (NER) is a subfield of Natural Language Processing (NLP) that aims to annotate proper nouns such as person, place, and organization. We use the CRF-based NER model that was proposed by Cekinel et al. [43]. The model is trained on the Turkish news and social media datasets that were annotated by Tur et al. [44] and Seker et al. [45]. The NER model annotates the terms that fall within PERSON, LOCATION, ORGANIZATION, DATE, TIME, MONEY, or PERCENT categories.

In this graph model, a news is represented as an unweighted directed graph. In the graph model, nodes are in *News*, *Named Entity* or *Term* categories and edges are in *Contains*, *Is* or *Followed By* relationship categories.

Each graph consists of a unique *News* node, one or more *Named Entity* nodes and several *Term* nodes. The graph database contains seven *Named Entity* nodes(categories) in total. Each term that is annotated as a named entity are within *Term* class and there exists a unique node for each term. If a term exists in two or more news, the same term node is connected to unique *News* nodes.

News and *Term* nodes are connected with the *Contains* relationship. It implies that terms are in the content. Furthermore, the *Is* relationship links *Term* nodes to its corresponding *Named Entity* category. In this graph model, all terms must belong to one of the "Named Entity" categories. Finally, there is a relationship between the consecutive terms based on the order of the terms labeled as *Followed_By*. A sample graph structure is presented in Figure 4.4.

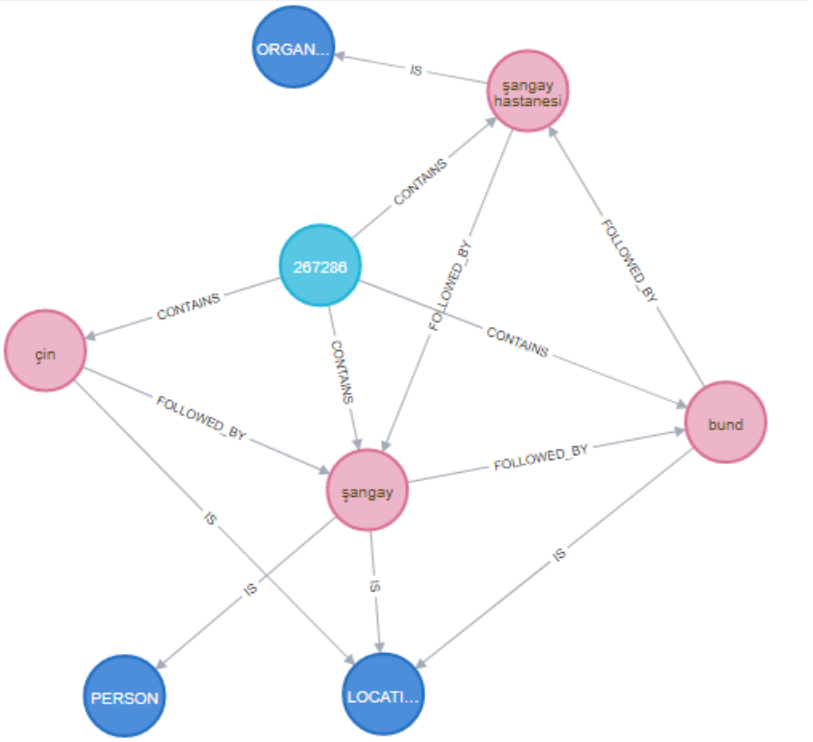


Figure 4.4: Named entity graph model

4.3.3 TF-IDF Graph Model

Each news is represented as an unweighted directed graph whose nodes are the terms with high (above the user-specified threshold) TF-IDF scores and a unique node for each news. The graphs have two types of relationships namely *Contains* and *Followed_By*. *Contains* relationship binds the unique *News* nodes to *Term* nodes that are passing in the context of the news. *Followed_By* relationship exists between consecutive term nodes. The edge direction of the *Followed_By* relationship indicates the order of the terms. A sample graph structure is presented in Figure 4.5.

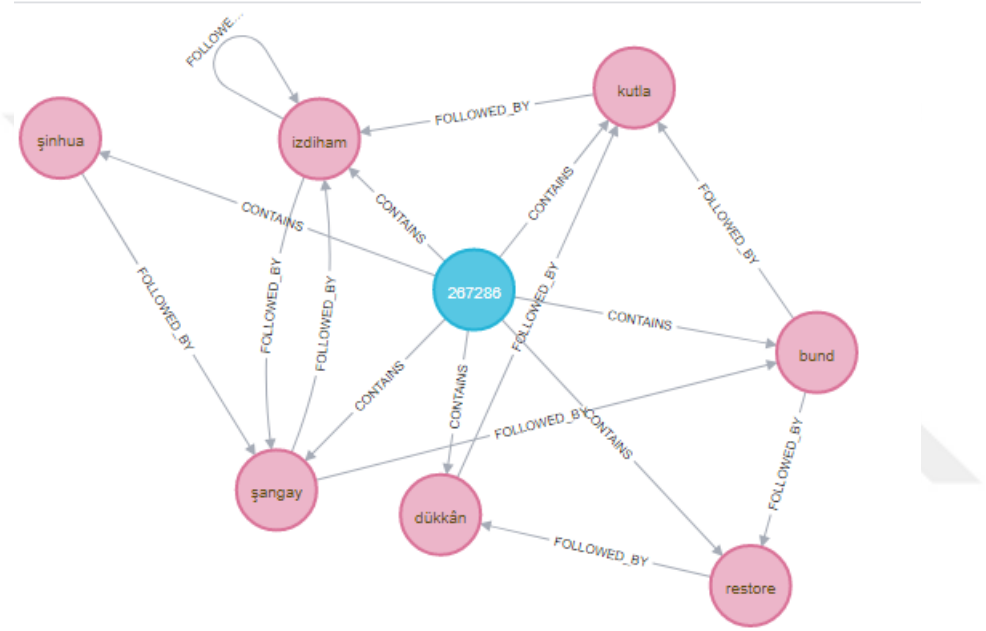


Figure 4.5: TF-IDF graph model

CHAPTER 5

METHODS

Our pipeline is demonstrated in Figure 5.1. According to this pipeline, we preprocess our textual data and create a graph-of-words structure for each news. In the second step, the gSpan [7] algorithm is executed for each time window to extract the frequent subgraphs. The frequent subgraphs indicate skeleton of important patterns that occur within these periods. In the third step, to discover the periodic patterns in the news, we form subgraph sequences where each subgraph is represented as an item and subgraphs that happen on the same day form an itemset. The RuleGen [32] algorithm is applied to discover the sequential rules in the sequence transactions. In the fourth phase, we search subgraphs in the test set that are almost identical to the patterns in the rule set. We measure the similarity between the subgraphs using our pairwise similarity algorithm and several graph embedding metrics. After that, we predict a rule's consequent if its antecedent is similar to some items in the test sequence. Basically, our model predicts news in the form of subgraph itemsets, where each subgraph is an indicator of a core of some patterns. Therefore, our model discovers the upcoming stories by extracting the sequential rules among news that happened in the past.

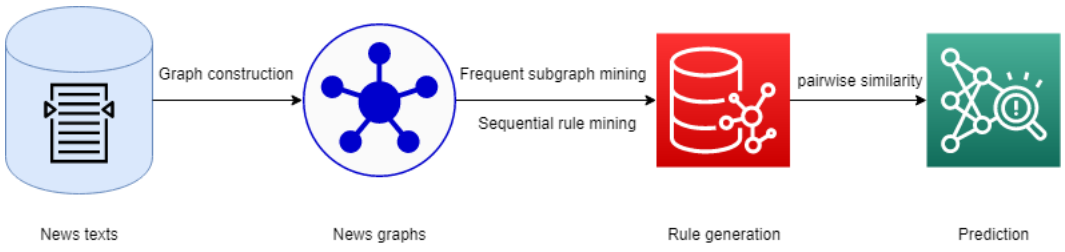


Figure 5.1: The general pipeline

5.1 Graph Representation

We constructed three graph models that we described in Chapter 4 and decided to continue with the TF-IDF graph model. Because the TF-IDF graph model consists of tokens with high TF-IDF scores. As a result, this model is more indicative and captures most of the contexts better than the other graph models. Besides, both the noun-verb and named entity graph models output very similar subgraph patterns, which are not so determinative and descriptive. Notably, the noun-verb model's recommendations contain many non-identifier words so that we couldn't get the main idea of the stories. Because of these reasons, we use the TF-IDF graph model in our experiments.

5.2 Frequent Subgraph Mining

We used a Python implementation [1] of the gSpan algorithm and the graph data file format of this implementation is presented below:

t # <graph_id>
v <vertex_id> <v_label>
⋮
e <init_id> <dest_id> <e_label>
⋮

The topmost line indicates the definition of graph with the id of "graph_id". "v" is a specific keyword for vertex definitions and it means that vertex with the "vertex_id" in this graph is labeled as "v_label". Additionally, "e" is a keyword for edge definitions and it means that there is an edge from the vertex with "init_id" to the vertex with "dest_id". The edge is labeled as "e_label".

We divided the dataset into configurable time windows and prepare gSpan input files for each time window. Then, we execute the gSpan [7] algorithm for each period separately and extract frequent subgraphs. This is because we can extract subgraphs that are related to the local stories that had happened in these periods.

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

5.3 Sequential Rule Mining

In our model, we formed sequences of configurable window size by ordering frequent subgraphs according to the date of the news that they appeared. Each itemset represents a day, and items in an itemset appear on different days. Hence, subgraphs in an itemset belong to the same day. For instance, S = < {g1, g2, g3}, {g1, g4}, {g5, g6, g7}, {g4, g5, g8, g9}, {g7, g9}, {g6, g7, g8}, {g9}> be a sequence of subgraphs. Each itemset in this sequence happened simultaneously, and there is a time-based ordering between the itemsets. In this example, the second itemset appears later than the first itemset. We executed the PrefixSpan [31] to find all frequent sequences and modified version of the RuleGen [32] algorithm to find all sequential rules whose support and confidence values are higher than a user-defined *minsup* and *minconf*. Our implementation of the PrefixSpan and the RuleGen algorithms are based on the SPMF library [46]. The original RuleGen algorithm finds rules such as $A \rightarrow B$ where $A \subset B$. However, our implementation discovers rules in the form of $A \rightarrow (B - A)$.

5.4 Rule-based Subgraph Matching

We implemented a rule-based subgraph matching algorithm that discovers similar pairs of our rules and the test sequences. For this purpose, we need to represent subgraphs in a lower-dimensional space and assign a vector for each subgraph. As we described in Chapter 2, several techniques encode graphs in lower-dimensional vectors. We apply node embedding (i.e. DeepWalk[15], node2vec [4]), subgraph embedding(i.e. Sub2Vec [17], subgraph2vec [18]) and graph embedding (i.e. graph2vec [20]) techniques to represent frequent subgraphs as vectors. Instead of vectors, there are also other methods such as graph edit distance [47] that can measure the similarity of two graphs using minimum graph operations required to transform one graph into another form. However, this method has some drawbacks. It was proven that computing the edit distance is NP-hard [48]. Therefore, as the size of the graph become larger, the number of computation will increase exponentially. After obtaining the embeddings for all frequent subgraphs, we measured the cosine similarity between the sequential subgraph rules and test subgraph sequences. We calculated the pair-

wise similarity between the rules and test sequences, and details of the algorithm are given in section 5.4.2.

5.4.1 Subgraph Embedding Techniques

5.4.1.1 One-hot Encoding

Graph one-hot encoding is a bag-of-word model for graphs that represents each subgraph as a vector. In that vector, all weight values are zero except the terms that appear in the subgraph. It is a simple model and the vector size should be at least as large as the size of the vocabulary. One hot representation of two subgraphs are similar as long as they contain the same terms. Therefore, it is not able to capture semantic relationships between the terms.

5.4.1.2 Node2vec

Node2vec [4] uses short random walks to learn node embeddings similar to DeepWalk [15]. The main difference between DeepWalk and node2vec is that DeepWalk uses pure random walks. However, node2vec uses biased random walks, and we can control randomness of each walk with an adjustable parameter. Both node2vec and DeepWalk are inspired by the skip-gram model of Word2vec [42]. Instead of words and sentences, there are nodes and random walks. The training objectives of these models are also very similar to the training objective of the skip-gram Word2vec model.

The node2vec model is trained on our news graphs. Node2vec produces a vector for each node, so we take the average of node embeddings that appear in the subgraph to calculate the subgraph's embedding. Note that a node may have several vectorial representations for different news graphs, so we take account of the embeddings that support our frequent subgraph. For instance, $S = \{v_1, v_2, v_3\}$ is a subgraph that appears in G_1, G_2, G_3, G_4. The node2vec is a node embedding algorithm, so the

subgraph embedding can be calculated using the formula given in Equation 5.1:

$$E_S = avg(v_1, v_2, v_3) \tag{5.1}$$

Note that, the representation of v_1, v_2, v_3 may differ according to graph structures. Therefore, we need to store subgraph embeddings for each supporting graph. In this example, the support of S is 4. Hence, we should store four different subgraph embeddings for S.

5.4.1.3 Sub2Vec

Sub2Vec [17] is another random walk based method that learns the latent representation of a subgraph. Sub2vec [17] and Subgraph2vec [18] learn the subgraph embeddings but the major difference of the Sub2vec is that it can learn a representation for arbitrary subgraphs that preserve second-order proximity or structural equivalence. The second-order proximity is related to local connectivities of the subgraph, and the structural equivalence refers to the overall structure of the subgraph. The main objective of the Sub2vec algorithm is to maintain the neighborhood or the structural properties while learning the embedding of a subgraph. Since the Sub2vec model produces a vector for each subgraph, we used these vectors as a subgraph representation.

5.4.1.4 Graph2vec

Graph2vec [20] is a neural embedding model that learns the representation of a whole graph using non-linear substructures particularly rooted subgraphs. It is inspired by the idea of Doc2vec [21], a framework that learns the latent representation of a document, that uses word embeddings and a document embedding for training a neural model. Graph2vec views the whole graph as a document and rooted subgraphs as words of the document and trains these vectors with neural networks to learn the lower-dimensional representation of the graph structure. The graph2vec model accepts the frequent subgraphs as input and learns the latent representation of each

frequent subgraph. Since the graph2vec model provides a single embedding for each graph, we used these vectors as a subgraph embeddings.

Algorithm 1: calcSim (subgid1, subgid2, matchAnte, emb)

```
input : subgid1: Subgraph id of the first subgraph
        subgid2: Subgraph id of the second subgraph
        emb: indicator for the graph embedding technique
output: cosine similarity value of the subgraphs according to the desired
        graph embedding technique
vec1 ← getEmbedding(subgid1, emb)
vec2 ← getEmbedding(subgid2, emb)
if emb == "NODE2VEC" then
    maxSim ← -1
    for K in range(len(vec1)) do
        for j in range(len(vec2)) do
            enc1 ← vec1[k]
            enc2 ← vec2[j]
            sim ← cosineSim(enc1, enc2)
            if maxSim < sim then
                maxSim = sim
            end
        end
    end
    return maxSim
end
return cosineSim (vec1, vec2)
```

5.4.2 Pairwise Similarity Calculation

We aim to find patterns on the test subgraph sequences that are very similar to antecedent of a sequential rule. Because it implies that there is an association between news that happened in the past and the forthcoming news. Since graphs have a complex structure, we need to represent them in a lower-dimensional space such as vectors. In the previous section, we represent each frequent subgraph as a vector using one of the embeddings. We measure the cosine similarity between embeddings of the

sequential rules and the test sequences in this part using Algorithm 1. If the node2vec embedding is chosen, "getEmbedding" method returns a list of vectors for the given subgraph. Each vector represents aggregated subgraph embeddings that are extracted from supporting graphs. We measure the similarity using all supporting vectors and return the greatest similarity score. Since our sequences may consist of many subgraphs, we calculate the cosine similarity of antecedent of each rule and subgraphs in the test sequence one-by-one. If the similarity is greater than the *sim_thres* value, we treat them as similar subgraph pairs and continue to seek matching for the remaining parts of the rule with the test sequence. Let the rule antecedent consists of several itemsets, and we find a pattern in the test sequence that is similar to the rule itemset. Then, we start searching a pattern for the rule's next itemset from the next day (next itemset of the sequence). Therefore, it is guaranteed that a similar pattern in the sequence does not violate the time-based order of the rule. For this purpose, we implement an algorithm and pseudocode of the algorithm is given in Algorithm 2. If we fully match rule antecedent with the test sequence, then we recommend the rule's consequent as the prediction which means that in the rest of the test sequence there will be a pattern which will be semantically identical and it includes the terms of the rule consequent in its context.

Algorithm 2: pairwiseSim (sequence, ruleAnte, matchAnte, sim_thres, emb, day)

input : seq = $([S_1, S_2], [S_1, S_3, S4], ..., [S_n])$: subgraphs that appear on the same day are located at the same index.
ruleAnte : either antecedent or consequent of a sequential rule
matchAnte : it will be assigned to the matched part of the sequence
sim_thres : cosine similarity threshold
emb: indicator for the graph embedding technique
day: points to the current itemset of the sequence

output: Integer value that indicates whether there is a match or not

```
i ← 0 ;
while day < len(sequence): do
    itemset ← [], seqIdx ← 0, ruleIdx ← 0;
    while i < len(ruleAnte) and seqIdx < len(sequence[day]) and ruleIdx <
     len(ruleAnte[i]): do
        sim = calcSim(ruleAnte[i][ruleIdx], sequence[day][seqIdx], emb)
        if sim > sim_thres then
            ruleIdx += 1
            itemset.append(sequence[day][seqIdx])
            if ruleIdx == len(ruleAnte[i]) then
                matchAnte.append(tuple(itemset)) i += 1
            end
        end
        seqIdx += 1
    end
    day += 1
    if i == len(ruleAnte) and ruleIdx == len(ruleAnte[i-1]) then
        return day;
    end
end
matchAnte ← []
return -1
```

5.5 Running Example

In this section, we want to demonstrate the pipeline with a running example. For simplicity, we selected 5 news from the dataset. Some parts of the news are presented below.

News #1: ... Türkiye, reel ücretlerdeki düşüş ve döviz kurlarındaki gelişmelere bağlı olarak ucuz üretimde dünyanın en ucuz üreticisi olarak bilinen Çin'in düzeyini yakaladı. ...

News #2: ABD'de ISM imalat endeksi Aralık'ta 55.5 ile beklentilerin altında açıklandı. Endeks Aralık'ta son 6 ayın en düşük seviyesine gerilemiş oldu. Reuters anketine katılan ekonomistler, Kasım'da 58.7 değerini alan endeksin Aralık'ta 57.6 seviyesinde açıklanmasını bekliyorlardı. Endeksin 50'nin altında bir değer alması sektörde daralmaya, üzerinde bir değer alması ise büyümeye işaret ediyor.

News #3: ... CNR EXPO Yeşilköy'de İSMOB Fuarı'nın bu yıl 11'incisi düzenlenecek. Rusya ve Ukrayna arasında yaşanan krize rağmen özellikle Rusya'dan İSMOB'a binlerce ziyaretçi kaydı sektörü sevindirdi. ...

News #4: ... 2014 sonunda 3 milyar dolar olması beklenen mücevher ihracatında hedef aşılarak 3.1 milyar dolar rakamına ulaşıldı. ...

News #5: ... Türkiye'de sayıları yaklaşık 9 milyonu bulan SSK ve Bağ-kur emeklisinin maaşlarına yapılacak zam oranı belli oldu. ...

To begin with, we removed all non-unicode characters and tokenized and lemmatized the texts using the Zemberek library. Afterward, we calculated the TF-IDF score of each term and pruned those words that do not satisfy the minimum threshold condition. We determined the threshold value after obtaining the scores. Therefore, TF-IDF pruning is done according to the observations. After pruning non-interesting tokens, we had 428 unique terms to construct news graphs. The news graphs are illustrated in Figure 5.2. Note that, we used Neo4j [2] database to visualize the graphs.

[2] https://neo4j.com/

Secondly, we generated subgraphs by applying the frequent subgraph mining module and constructed subgraph sequences by executing the sequential rule mining module. The training set consists of the News [1,4], and the test set consists of News 4 and News 5. The fourth news exists in both training, and the test set because we need more than one news in the test set to match our sequential rules. We executed the gSpan algorithm by setting the minsup paremeter to 3 and 2 respectively and the minimum number of vertices to 7 and formed subgraph sequences of length 2 and slided one day for each window. The format of our sequences is illustrated in Table 5.1. The output of the frequent subgraph module is illustrated in Table 5.2

Table 5.1: Sequence format with granularity=1

Sequences	Days
Sequence 1	1 2
Sequence 2	2 3
Sequence 3	3 4
Test Sequence	4 5

Table 5.2: Subgraph sequences of the sample news

Sequences	Itemsets
Sequence 1	{{0,1,2,3,4,5}}
Sequence 2	{{0,1,2,3,4,5}}
Sequence 3	{{0,1,2,3,4,5}, {0,1,2,3,4,5}}

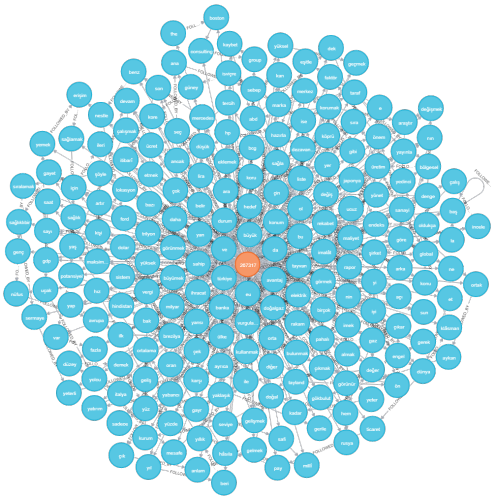
Subsequently, we extracted sequential rules using our sequential rule mining module. We set *minsup*=1 to find all sequential rules. The module outputs 961 sequential rules with 31 unique rule antecedents.

Finally, we executed our prediction module to find similar patterns in our test set. We measured the similarity of two subgraphs using the node2vec [4] algorithm. Our pairwise similarity algorithm tries to find identical pairs from rule antecedents and

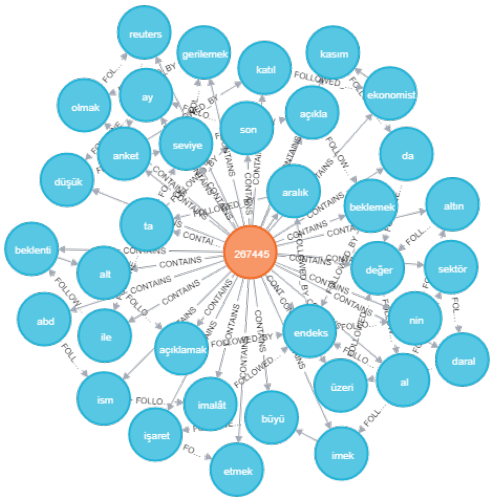
the test sequences. Whenever it finds a similar pair, it repeats the same procedure for the other subgraphs in rule antecedent and the rest of the test sequence. We executed this module by setting the minimum cosine similarity threshold to 0.7. Our algorithm found matching for a rule with the test sequence, and the matched part is given below.

- Sequential rule: {1,2,3,4} $\rightarrow$ {1}
- Rule antecedent: {1,2,3,4}
- Rule consequent: {1}
- Matched test sequence: {13, 15, 25, 27}

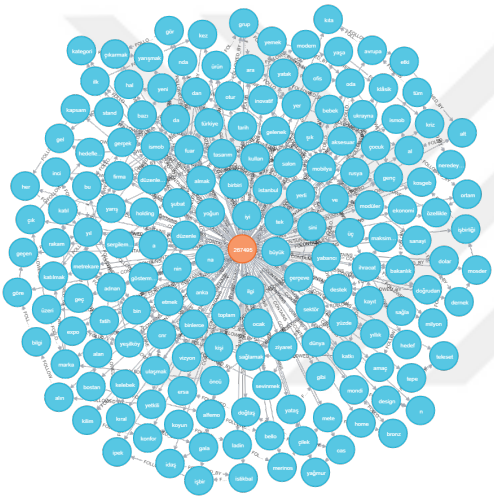
According to the matching, a subgraph similar to the Subgraph 1, is going to happen in the near future. Because our model detected that the rule's antecedent is very similar to the subsequence of test sequence which indicates that a news that contains a subgraph identical to Subgraph 1 may occur in the near future. The matched subgraph pairs are presented in Figures 5.3, 5.4, 5.5, 5.6 respectively and the predicted subgraph in Figure 5.7.



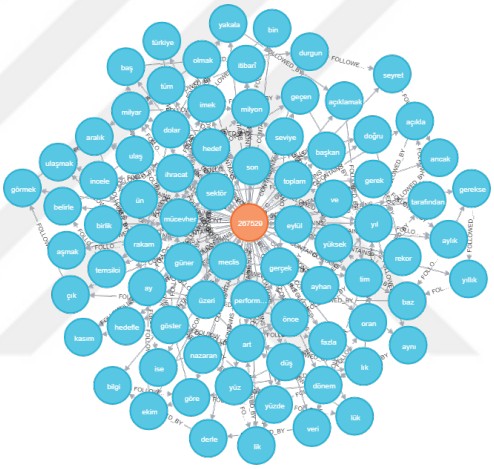
(a) News 1



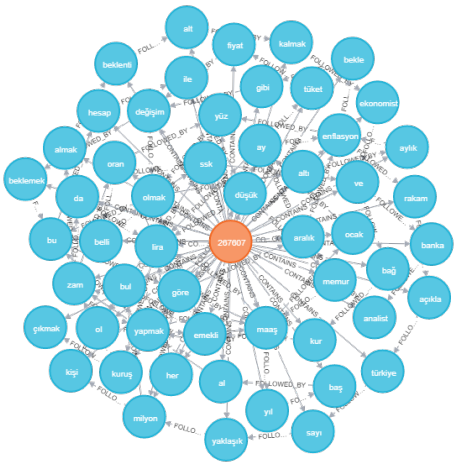
(b) News 2



(c) News 3



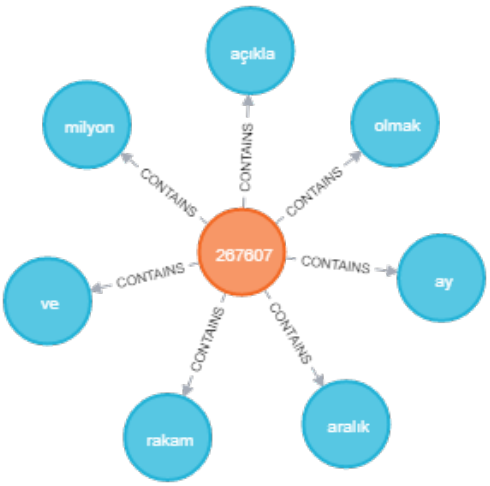
(d) News 4



(e) News 5

Figure 5.2: Sample news graphs in Neo4j

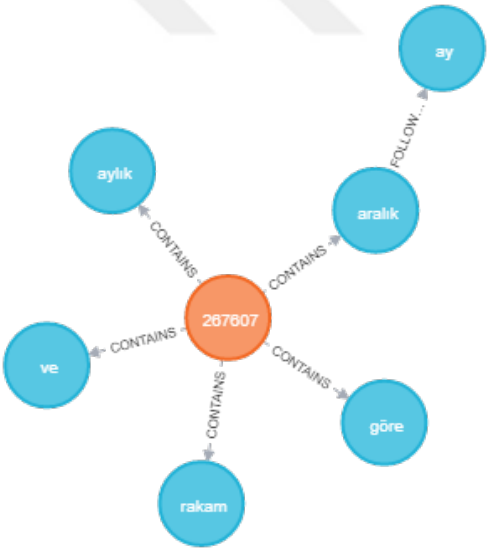
(a) Subgraph 1



(b) Subgraph 13

Figure 5.3: Similar subgraph pairs for the first matching

(a) Subgraph 2



(b) Subgraph 15

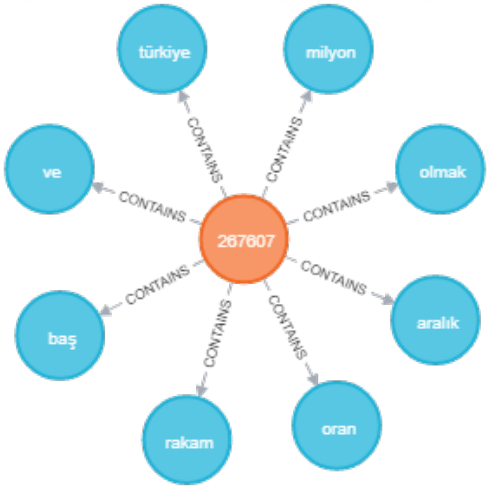
Figure 5.4: Similar subgraph pairs for the second matching

(a) Subgraph 2

(b) Subgraph 15

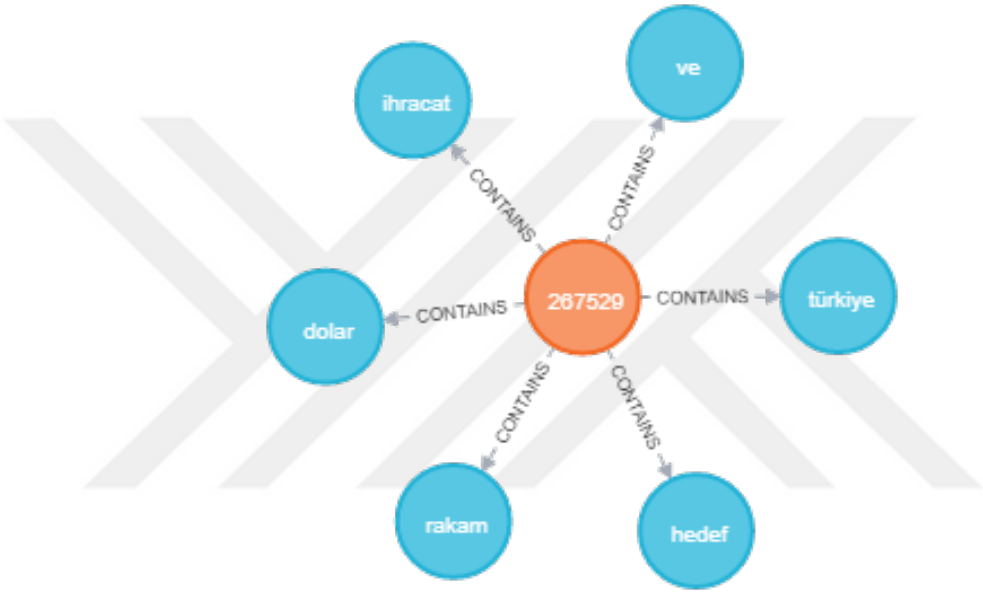
Figure 5.5: Similar subgraph pairs for the third matching

(a) Subgraph 2



(b) Subgraph 15

Figure 5.6: Similar subgraph pairs for the fourth matching



(a) Subgraph 2

Figure 5.7: The rule consequent

CHAPTER 6

EXPERIMENTS

6.1 Experimental Setting

We used the TF-IDF graph model during the experiments as we observed that it captures the context better than the other graph models. After generating the graphs, frequent subgraphs for each month are extracted by using the gSpan algorithm. Note that dynamic support values are selected to set an upper bound and restrict the number of frequent subgraphs each month and prevent bias towards any month. The number of frequent subgraphs and support thresholds for each month are presented in Figures 6.1 and 6.2.

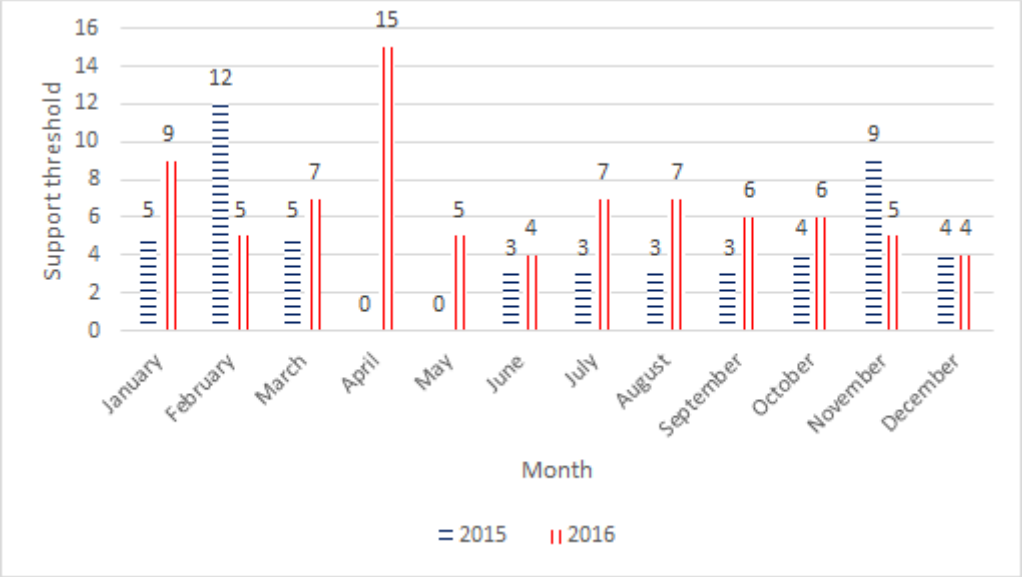


Figure 6.1: Support thresholds in each month

The gSpan algorithm finds too many subgraphs that are the mirror image of each

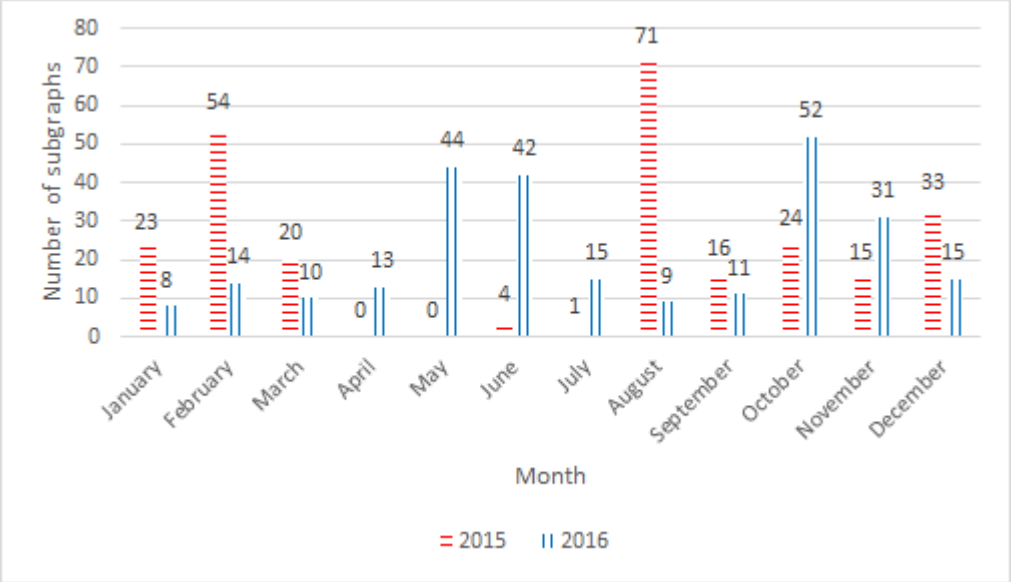


Figure 6.2: Number of frequent subgraphs in each month

other. Therefore, we encode subgraphs as a one-hot vector and cluster those subgraphs that are very similar. This is because one-hot encoding preserves syntactic meaning, and the sampling procedure reduced the number of subgraphs significantly. We group the subgraph pairs if cosine similarity value is higher than 0.8. Then, we take a sample from each cluster randomly. Finally, sequences are formed from the selected subgraphs. To be more specific, subgraphs that appear on the same day form an itemset, and a sequence consists of several itemsets where each itemset represents a day. The length of the sequence, in other words, sequence window size, is set as a parameter.

Secondly, the PrefixSpan and our RuleGen algorithm are executed on our sequences to find all frequent sequential rules that hold the following criteria:

$$minsup \geqslant 0.50 \text{ and } minconf \geqslant 0.66$$

Note that, we performed validation experiments to find optimum values for these parameters. According to our observations, these values discover optimum number of rules in a reasonable time period.

Finally, we measure our model's precision, recall, and f1-score using our pairwise similarity algorithm. We use the Equations 6.1, 6.2 and 6.3 while evaluating our

model's success.

- *Ante_Match:* We consider a sequence matches with a rule's antecedent if there exist some itemsets in the sequence that are similar (above the similarity threshold) to all itemsets of the rule antecedent.
- *Cons_Match:* A rule's consequent matches with a sequence if the rule's antecedent had been matched and there exist some itemsets in the sequence that are similar to the rule consequent.

$$precision = \frac{Cons\_Match}{Ante\_Match} \tag{6.1}$$

$$recall = \frac{Cons\_Match}{Total\ Sequences} \tag{6.2}$$

$$F1Score = 2 * \frac{precision * recall}{precision + recall} \tag{6.3}$$

6.2 Experiments and Results

6.2.1 Experiment 1: Graph Embeddings User Study

In the first experiment, we analyzed which embedding techniques are the most applicable for the news prediction. Therefore a user study was conducted by the participation of six judges. We formed a multiple-choice survey and wanted to examine which embedding techniques would be useful for further experiments. We provided 20 news subgraphs selected randomly, and our judges picked the most similar subgraphs from the choices. We provided four options that were the most similar news graphs found by one-hot encoding, graph2vec, sub2vec, and node2vec embeddings. Note that we measured the cosine similarity between subgraph pairs. We presented sample survey questions in Appendix A. We divided the judges into two groups. The judges in the first group did not see the context of the news. On the other hand, the second group was allowed to see the news context for each question. Judge 1, Judge 2, and Judge 3

are in the first group, and the rest are in the second group. The judges chose the most similar choices from the options. If none of the options are appropriate, they do not select any of them.

By performing this survey, we aim to capture user behavior for the subgraph similarity. In other words, we provided several candidate subgraphs and compare which ones are the most similar from the users' perspective. Because the determination of similar subgraphs is the essence of this work and it states the quality and evaluation of the results.

Evaluations of the judges are presented in Table 6.1. According to the reviews of both the first and second groups, one-hot encoding and node2vec embeddings generate more similar outputs with the given news subgraph. Thus, in the rest of the experiments, we focused on the one-hot encoding and node2vec embeddings.

Table 6.1: Graph embedding survey results

Survey		One-hot	Graph2vec	Sub2vec	Node2vec
Blind Review	Judge 1	**14**	7	12	10
	Judge 2	**17**	7	12	**17**
	Judge 3	**17**	7	11	16
Review with content	Judge 4	11	4	9	**15**
	Judge 5	**13**	3	9	10
	Judge 6	**18**	9	12	**18**
Average		**15.0**	6.2	10.8	**14.3**
Percentage (%)		**75**	31	54	**71.5**

6.2.2 Experiment 2: Effects of the Sequence Length

In this experiment, we investigated the effect of sequence length on the model's success. For this purpose, we formed sequences of lengths 5, 7, and 10. From now on, we denote them as *Seq5*, *Seq7*, and *Seq10*, respectively. An itemset in a sequence contains the subgraphs that happened within the same day. Note that, we shift the sequence windows according to a granularity parameter. The sequence structures with

varying window sizes and granularities are shown in Table 6.2, 6.3 and 6.4, respectively. In our sequences, each day correspond to an unique itemset where subgraphs that happen in a day are an item. For instance, Day 1 is an itemset that includes set of subgraphs that occur in the news of Day 1. Besides, for *Seq5*, the granularity is set as 3. Likewise, for *Seq10*, we set the granularity as 5. The reason for varying granularity values is to prevent bias towards specific dates. In other words, if we had selected lower granularity values, dates in the middle of each month would appear more than twice, which may cause a bias towards these dates during rule generation. Likewise, if we had selected higher granularity values, we would miss inter-sequence associations.

Table 6.2: Sample *Seq5* with granularity = 3

Sequences	Days
Sequence 1	1 2 3 4 5
Sequence 2	4 5 6 7 8
Sequence 3	7 8 9 10 11
Sequence 4	10 11 12 13 14
Sequence 5	13 14 15 16 17
Sequence 6	16 17 18 19 20
Sequence 7	19 20 21 22 23
Sequence 8	22 23 24 25 26
Sequence 9	25 26 27 28 29
Sequence 10	28 29 30 31

After forming the sequences, we fed them into our pipeline to generate rules. During the rule generation we set minsup = 50% and minconf = 66% . Moreover, we selected 1000, 2000, 4000, 8000, and 16000 rules according to the interestingness metric and measured the model's success using these rules. The reason for putting a limit on the rules is to examine the effect of sequence length better. We measured the interestingness of a rule by using the formula given in Equation 6.4. The rules with high-interest

Table 6.3: Sample *Seq7* with granularity = 4

Sequences	Days
Sequence 1	1 2 3 4 5 6 7
Sequence 2	5 6 7 8 9 10 11
Sequence 3	9 10 11 12 13 14 15
Sequence 4	13 14 15 16 17 18 19
Sequence 5	17 18 19 20 21 22 23
Sequence 6	21 22 23 24 25 26 27
Sequence 7	25 26 27 28 29 30 31

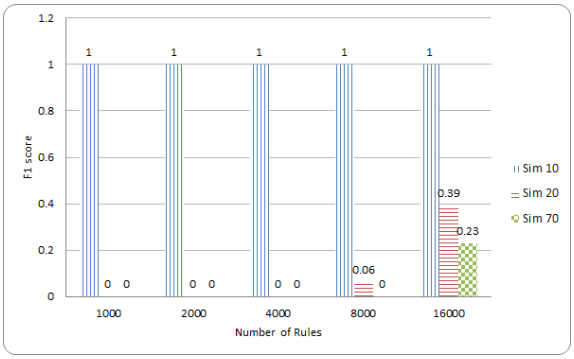
Table 6.4: Sample *Seq10* with granularity = 5

Sequences	Days
Sequence 1	1 2 3 4 5 6 7 8 9 10
Sequence 2	6 7 8 9 10 11 12 13 14 15
Sequence 3	11 12 13 14 15 16 17 18 19 20
Sequence 4	16 17 18 19 20 21 22 23 24 25
Sequence 5	21 22 23 24 25 26 27 28 29 30
Sequence 6	26 27 28 29 30 31

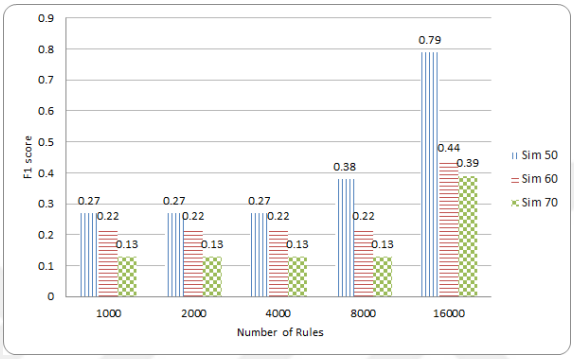
values (above 0.5) are considered to be more promising.

$$Interest(I \rightarrow j) = |conf(I \rightarrow j) - Pr[j]| \tag{6.4}$$

Figure 6.3, Figure 6.4 and Figure 6.5 show the F1-scores vs the effects of the increase in the number of rules for *Seq5*, *Seq7*, and *Seq10* respectively. Additionally, the precision and the recall results are presented in Appendix B. We measured the change using different cosine similarity thresholds. For instance, "Sim 20" indicates the case where cosine similarity threshold is set as 0.20. According to the experiments, F1 scores of both one-hot encoding and node2vec models were improved as the number



(a) One-hot encoding



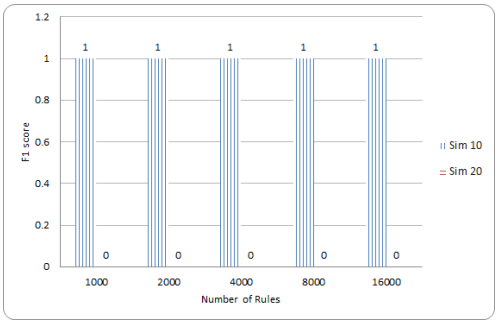
(b) node2vec

Figure 6.3: F1 scores vs top k interesting rules for *Seq5*

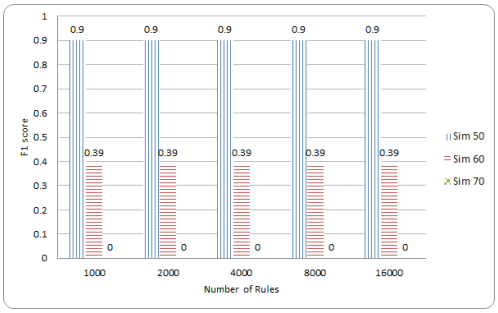
of rules increases. Besides, as we decreased the cosine similarity threshold, the F1 score increases.

We also evaluated how our model's success is affected when we change the sequence length and granularity. According to the measurements, the highest F1 scores were achieved with *Seq5*. Both the one-hot encoding model and the node2vec model performed best when we used the top 16000 interesting rules. To be more specific, when we set the cosine similarity threshold as 0.70, F1 scores of one-hot encoding and the node2vec models are 0.23 and 0.39 respectively.

Finally, we used all sequential rules without taking account of the interest metric. The results of this experiment are presented in Figure 6.6. Besides, we presented the precision and the recall results in Appendix B. In this experiment, we analyzed the effect of sequence lengths on the model's success without rule restrictions. According

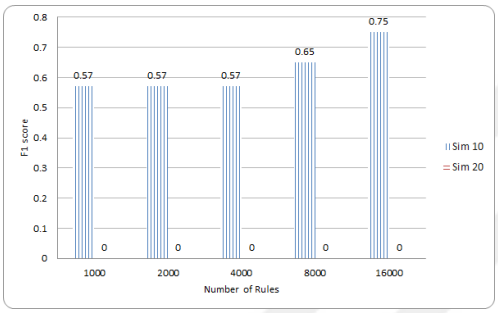


(a) One-hot encoding

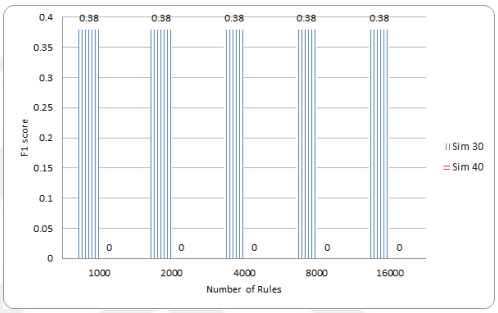


(b) node2vec

Figure 6.4: F1 scores vs top k interesting rules for *Seq7*



(a) One-hot encoding



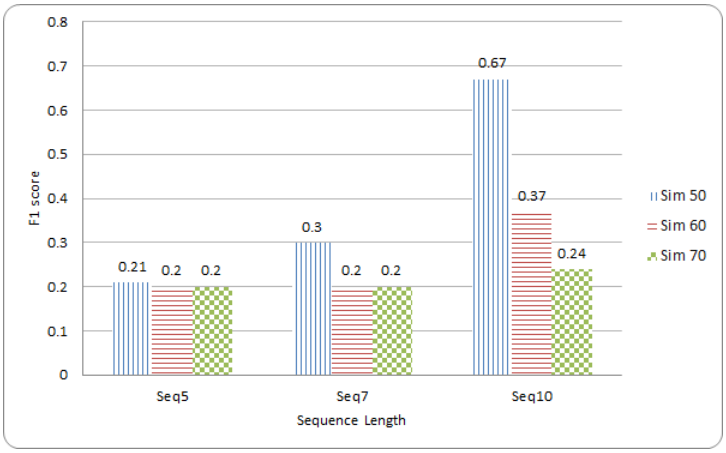
(b) node2vec

Figure 6.5: F1 scores vs top k interesting rules for *Seq10*

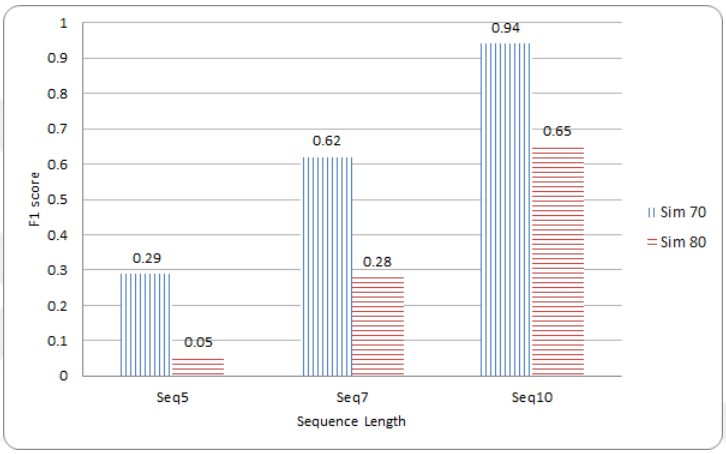
to the results, *Seq10* outperformed other models using both the one-hot encoding and the node2vec embeddings. *Seq7* was performed better than *Seq5* and placed in the middle. Note that, we extracted more rules with *Seq10* than the other configurations. It seems that rule size plays a crucial role, but not the only factor, on the model's success. Because when we restricted the number of rules, *Seq10* performed worst, and *Seq5* was the leading sequence configuration.

6.2.3 Experiment 3: Comparison with the Baselines

In this experiment, we compared our model's success with three baseline models. The first baseline that we introduced is the recommendation of the most frequent subgraphs. The second baseline that we presented is predicting the most frequent



(a) One-hot encoding



(b) node2vec

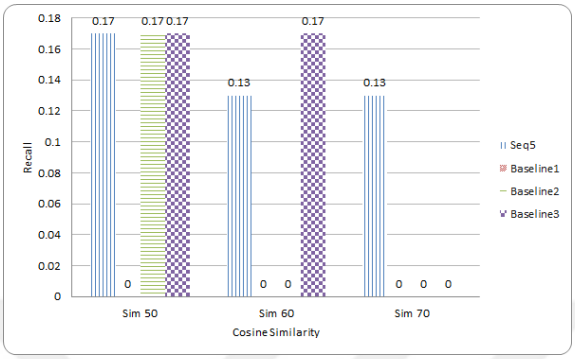
Figure 6.6: F1 scores vs sequence length for various similarity thresholds using all rules

subgraphs that occur in the last month of the training set. In our case, the last month of the training set was September 2016. The final baseline model that we offered is the recommendation of the most frequent sequences within the last month of the training set. We denoted them *Baseline1*, *Baseline2*, and *Baseline3* respectively in the following parts.

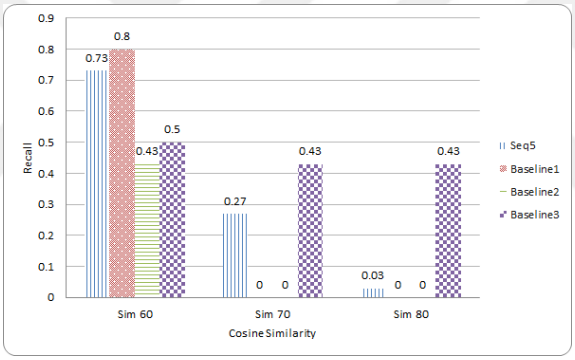
Our precision metric is not applicable for the baselines because it measures how many sequences that had already been matched with some rule antecedents also paired with the rule consequent. Our baselines neither contain antecedent nor consequent, so we

considered the recall values while evaluating the models' success.

In the experiments, we used the one-hot encoding and the node2vec embeddings with various cosine similarity thresholds. Besides, we used all sequential rules whose minsup is greater than 0,5 and minconf is greater than 0,66. Experimental results for *Seq5*, *Seq7* and *Seq10* are presented in Figure 6.7, Figure 6.8, and Figure 6.9.



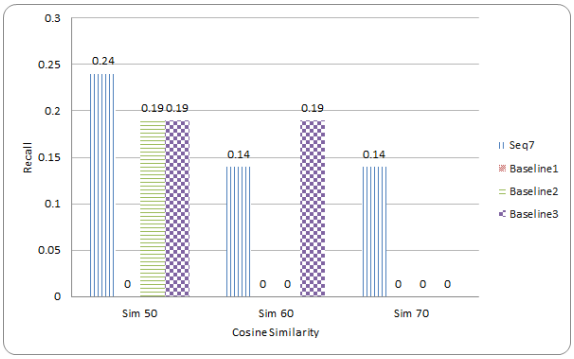
(a) One-hot encoding



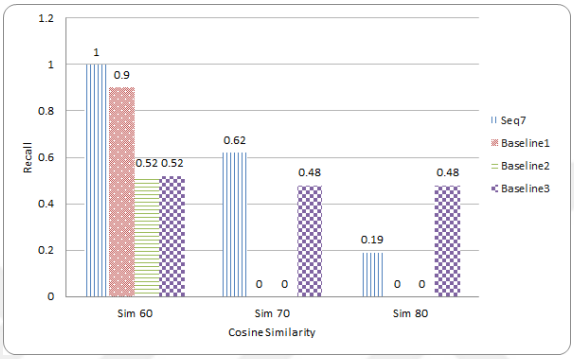
(b) node2vec

Figure 6.7: Recall vs cosine similarity for Seq5 and the baselines

According to the measurements, *Seq10* performed better than the baselines for both the one-hot encoding and the node2vec embeddings for all similarity thresholds. The only exception is a tie with the *Baseline3* using the one-hot encoding with the similarity threshold of 0,60. Seq7 also performed better than *Baseline1* and *Baseline2* in all configurations. However, *Baseline3* outperformed *Seq7* using node2vec embeddings with the Sim 80 and using one-hot encoding with Sim 60. Finally, *Seq5* performed worst among the proposed models. Its recall values were less than both *Baseline1* and *Baseline3* for some configurations. For instance, the first baseline's recall value



(a) One-hot encoding



(b) node2vec

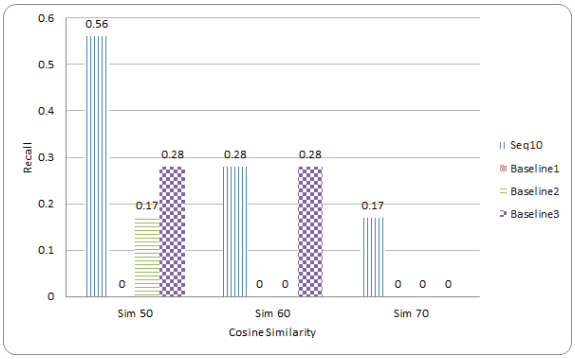
Figure 6.8: Recall vs cosine similarity for Seq7 and the baselines

for node2vec embeddings with Sim 60 is greater than *Seq5*'s recall.

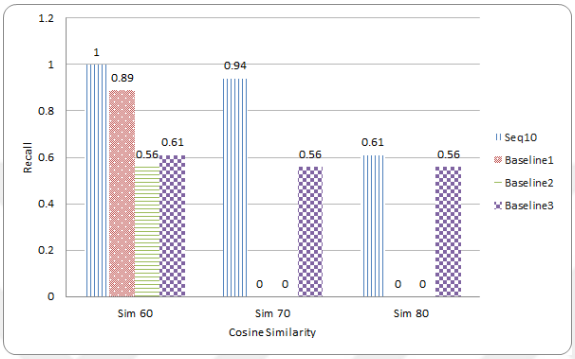
6.2.4 Experiment 4: Time Performance of the Proposed Model

In our final experiment, we evaluated how a change in the number of rules affects the proposed model's time performance. For this purpose, we measured the time for top k interesting rules. We selected 1000, 2000, 4000, 8000, and 16000 top interesting rules in the second experiment. Table 6.5 shows how many unique rule antecedent exists for each setting. This information is essential because our pairwise similarity algorithm first tries to find matching using the rule antecedents. Therefore, if we cannot find any matched pair for the antecedent, we do not search for the consequents.

Note that in all of our experiments, node2vec takes more time because node2vec gives a node embedding for each node in a given graph. However, our subgraphs



(a) One-hot encoding



(b) node2vec

Figure 6.9: Recall vs cosine similarity for Seq10 and the baselines

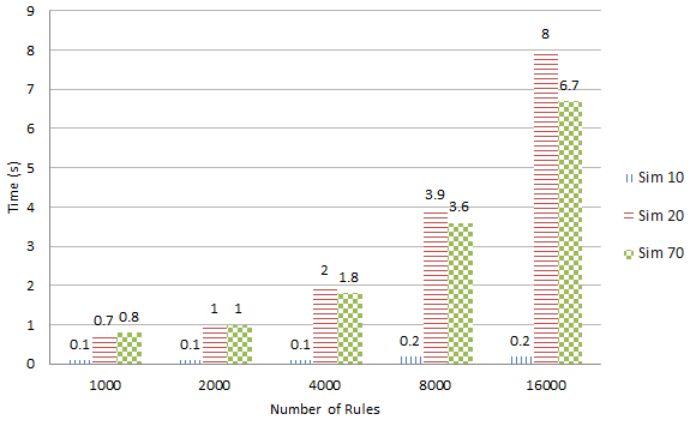
Table 6.5: Number of unique rule antecedents for varying sequence lengths and rule count

Sequence Length	**Top-k rules**				
	1000	**2000**	**4000**	**8000**	**16000**
5	11	20	38	74	145
7	9	11	11	11	11
10	1000	2000	4000	8000	16000

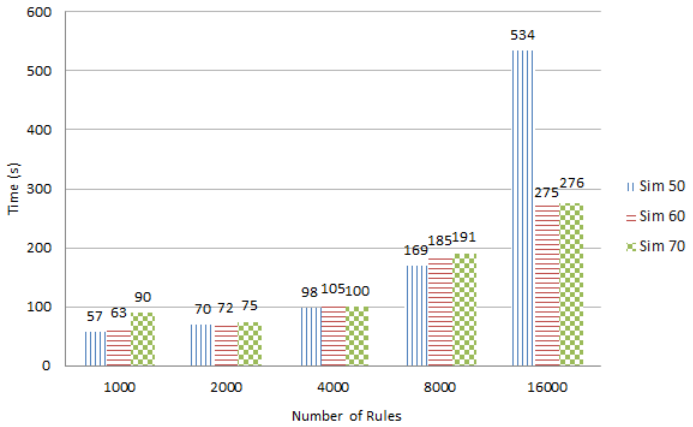
exist in multiple graphs, so for a node within a subgraph, there may be multiple embeddings representing the given node. As we mentioned in Chapter 5, we store all of these embeddings and take the average of node embeddings to find the subgraph embeddings. Because of this reason, using node2vec, we obtained several subgraph embeddings that represent the same subgraph, and we calculated similarity for each

of these embeddings.

Figure 6.10, Figure 6.11, and Figure 6.12 indicates the duration of the prediction module for *Seq5*, *Seq7*, and *Seq10* respectively. According to the figures, as the number of rules rises, the duration of the pairwise similarity and prediction processes are also increased. However, there is an exception. For sequences of length 7, the time required for the prediction module did not change significantly. Because unique rule antecedents remained almost same when we increased the number of rules.

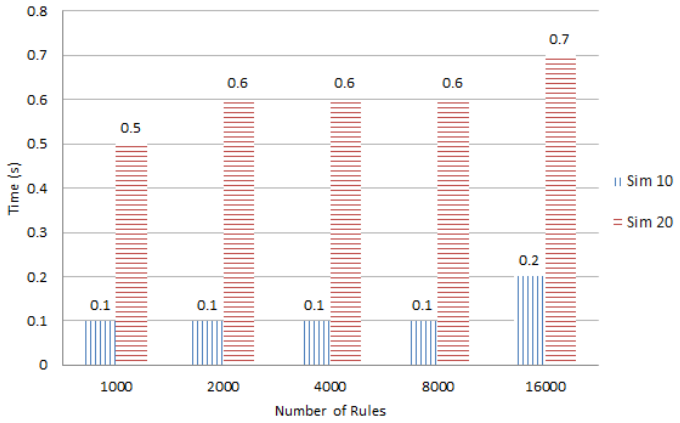


(a) One-hot encoding

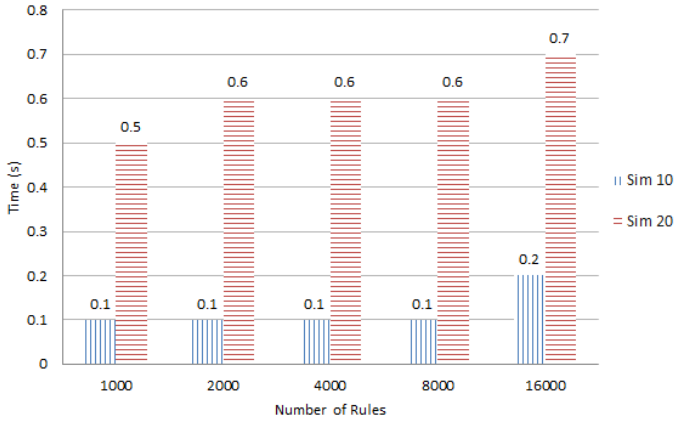


(b) node2vec

Figure 6.10: Time vs Number of Rules for Seq5

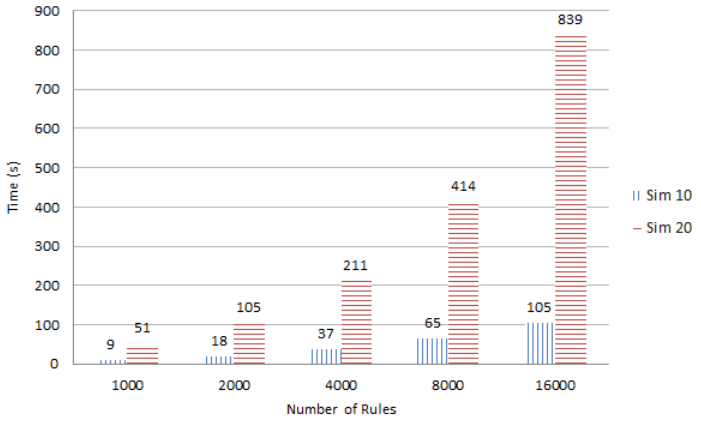


(a) One-hot encoding

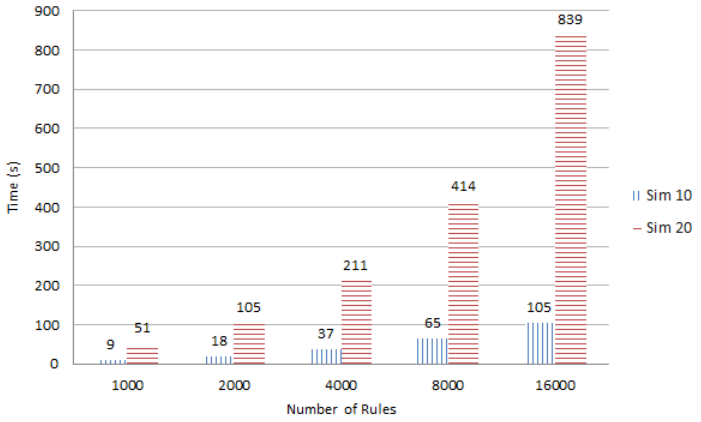


(b) node2vec

Figure 6.11: Time vs Number of Rules for Seq7



(a) One-hot encoding



(b) node2vec

Figure 6.12: Time vs Number of Rules for Seq10

6.3 Case Study

In this section, we verified that the model's predictions are similar to the near future news. We sampled predicted news graphs using *Seq7* model with the node2vec embedding. In the first case, we tested our model's success with *Sim80* cosine similarity threshold. In the second case, we presented a case with *Sim60* threshold. These cases are an instance of the successful recommendation. On the other hand, the third case is an example of the unsuccessful prediction. Because in this example, while the rule antecedent was matched with the test sequence, the prediction did not satisfy the

similarity criteria. In this scenario, we selected the similarity threshold as *Sim60*.

6.3.1 Case 1: Successful prediction with the node2vec embedding and Sim80

The first case is an example of a successful prediction. We used sequences of a window size 7 with a granularity of 4, as we described above. The node2vec found an identical pair for a rule and a test sequence given below:

- Itemsets of the test sequence:

  1. {481,482,483,484,485,486,487,488,501,502,503,504,505,506,507,508,509}
  2. {490,492,494,500}
  3. {**489**}
  4. {481,483,484,485,486,487,**489**,502,503,504,505,506,508,509}

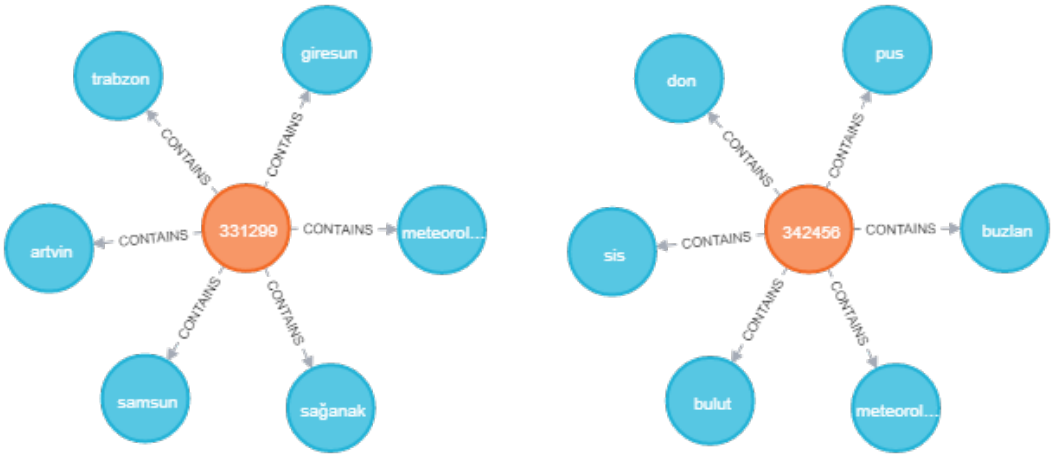
- Matched test sequence:$\{489\} \rightarrow \{489\}$

- Matched rule: $\{393\} \rightarrow \{394\}$

In this scenario, the model's prediction is a subgraph that is identical to Subgraph 394. Our pairwise similarity algorithm found an identical subgraph to 394 in the test sequences. According to the model, Subgraph 489 was found as a similar subgraph to the rule's consequent. Therefore, the model's prediction is considered as a successful prediction. Subgraph visuals for the Subgraph 394 and Subgraph 489 are presented in Figure 6.13. Additionally, we present first paragraph of the news that contain these subgraphs. Both news are in the meteorology category, so they are treated as similar patterns.

News #394: Meteoroloji Genel Müdürlüğünden yapılan son değerlendirmelere göre; yurdun kuzey, iç ve doğu kesimlerinin parçalı yer yer çok bulutlu, Batı Karadeniz, Orta Karadeniz kıyıları, Doğu Karadeniz ile Kocaeli, Sakarya, Amasya, Erzurum'un kuzey ve doğu kesimleri, Kars, Ardahan, Ağrı, Iğdır ve Van çevrelerinin sağanak yağışlı, diğer yerlerin az bulutlu ve açık geçeceği tahmin ediliyor. Yağışların Samsun'un doğusu, Ordu, Giresun, Trabzon, Rize, Artvin ve Ardahan çevrelerinde kuvvetli, Trabzon'un doğusu, Rize ve Artvin'in kıyı kesimlerinde çok kuvvetli ve yer yer şiddetli olması bekleniyor. ...

News #489: Meteoroloji Genel Müdürlüğünden yapılan son değerlendirmelere göre yurdun batı kesimleri parçalı zamanla çok bulutlu, öğle saatlerinden sonra Marmara'nın batısı, akşam ve gece saatlerinde Marmara, Kuzey ve Kıyı Ege yağmur ve sağanak yağışlı geçecek. Yağışların akşam ve gece saatlerinde Edirne, Tekirdağ, Çanakkale, Balıkesir, Manisa, İzmir, Aydın'ın kıyı kesimleri ile Kuzey Ege kıyılarında kuvvetli olması bekleniyor. Bu sabah Marmara'nın güney ve doğusu ile sabah ve gece saatlerinde iç kesimlerde yer yer pus ve sis hadisesi ile birlikte Doğu Anadolu'nun kuzeydoğusunda kuvvetli olmak üzere iç ve doğu kesimlerde buzlanma ve don olayı bekleniyor. Hava sıcaklığında önemli bir değişiklik olmayacak. ...



(a) Subgraph 394

(b) Subgraph 489

Figure 6.13: Case 1 Predicted Subgraph and Matched Test Subgraph

6.3.2 Case 2: Successful prediction with the node2vec embedding and Sim60

The second case is also an example of the successful recommendation. In this case, we observed the effect of using lower cosine similarity threshold on the model's success at prediction. The node2vec found an identical pair for a rule and a test sequence given below:

- Itemsets of the test sequence:

  1. {510,**514**,516,523}
  2. {511,515,520,**521**,522}
  3. {511,**513**,514,515,521,522}
  4. {515,516,520,521,523,524}
  5. {516,520,523,524}

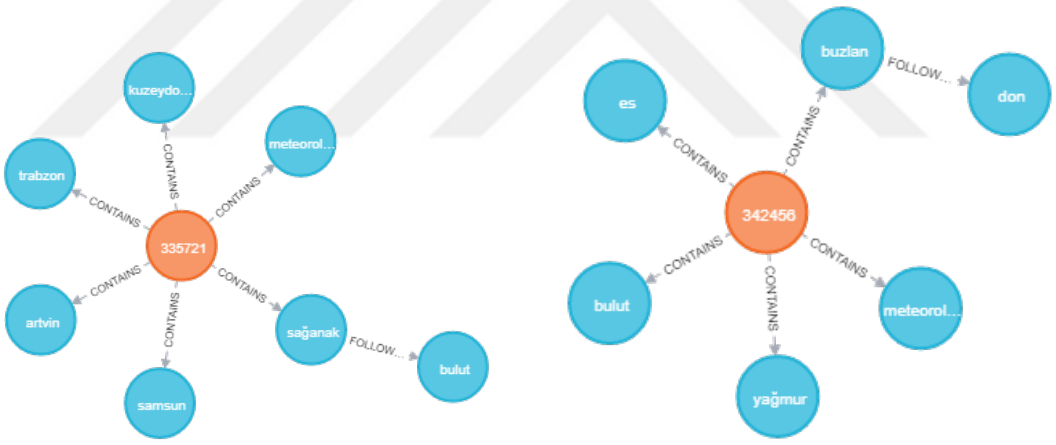
- Matched test sequence:{514}, {521} $\rightarrow$ {513}

- Matched rule: {395}, {403} $\rightarrow$ {393}

In this scenario, the model's prediction is a subgraph that is identical to Subgraph 393. Our pairwise similarity algorithm found an identical subgraph to 393 in the test sequences. According to the model, Subgraph 513 was found as a similar subgraph to the rule's consequent. Therefore, the model's prediction is considered as a successful prediction. According to the results, even though we decreased the threshold, content of the predicted subgraph and the appeared subgraph are considered to be similar. Because both news are related to the weather conditions. Subgraph visuals for the Subgraph 393 and Subgraph 513 are presented in Figure 6.14. Additionally, we present first paragraph of the news that contain these subgraphs.

News #393: Bugün Marmara'nın doğusu, Karadeniz, İç Anadolu'nun kuzeyi, Doğu Anadolu'nun kuzey ve doğusu ile Kayseri, Adana ve Hatay kıyılarının aralıklı yağışlı geçeceği tahmin ediliyor. Akşam saatlerinden sonra Doğu Anadolu'nun kuzey-doğusunda karla karışık yağmur ve yükseklerinde yer yer kar şeklinde görülecek olan yağışların, Trabzon merkez ve doğu ilçeleri ile Rize ve Artvin kıyılarında yerel olarak kuvvetli olması bekleniyor. ...

News #513: Yapılan son değerlendirmelere göre, ülkenin kuzey, iç ve batı kesimlerinin parçalı ve yer yer çok bulutlu, Marmara, İç Anadolu'nun kuzeyi, Batı ve Orta Karadeniz ile Kütahya, Uşak, Afyonkarahisar, Isparta, Konya, Gümüşhane, Bayburt ve Giresun ile gece saatlerinde Hatay, G.Antep ve Kilis çevrelerinin karla karışık yağmur ve kar yağışlı, diğer yerlerin az bulutlu ve açık geçeceği tahmin ediliyor. Doğu kesimlerde kuvvetli olmak üzere iç kesimlerde buzlanma ve don olayı ile birlikte yer yer sis ve pus hadisesi görüleceği tahmin ediliyor. ...



(a) Subgraph 393

(b) Subgraph 513

Figure 6.14: Case 2 Predicted Subgraph and Matched Test Subgraph

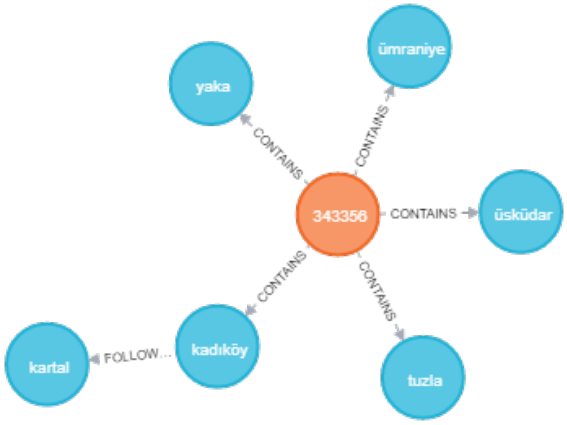
6.3.3 Case 3: Unsuccessful prediction with the node2vec embedding and Sim60

The third case is an example of an unsuccessful recommendation. Although there was a matching pair for the rule antecedent, the rule consequent - or the prediction -

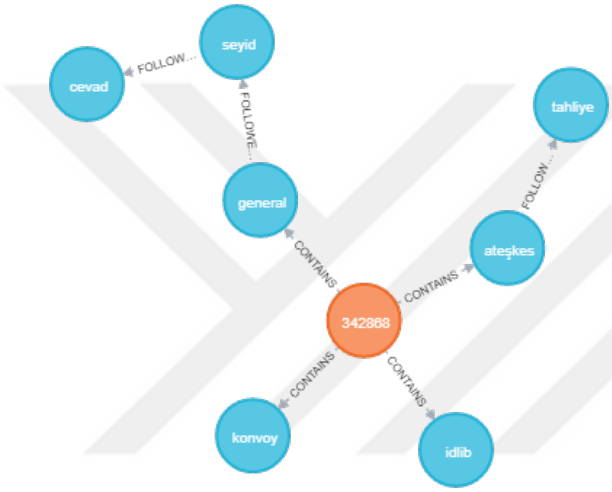
did not match with the rest of the test sequence. As a result, we considered this case as an unsuccessful prediction. A sample case is given below:

- Itemsets of the test sequence:
  1. {510, 514, **516**, 523}
  2. {511, **515**, 520, 521, 522}
  3. {511, 513, 514, 515, 521, 522}
  4. {515, 516, 520, 521, 523, 524}
  5. {516, 520, 523, 524}
- Matched test sequence:{516}, {515}
- Matched rule antecedent: {408}, {415}
- Rule consequent (prediction): {407}

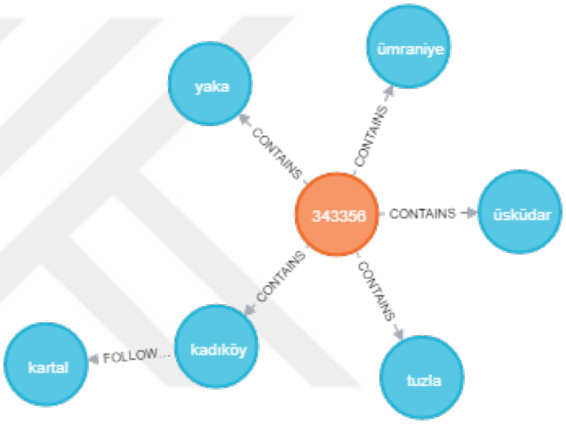
In this scenario, a subgraph identical to Subgraph 407 was predicted. However, the model could not find a similar pair with that subgraph. Therefore, it is an example of an unsuccessful recommendation. Figure 6.15 shows the matching pairs of rule antecedent and test sequence. According to the visuals, Subgraph 408 is about the real estate industry in Istanbul. However, the matching subgraph, Subgraph 516, is about the evacuation of Eastern Aleppo. Even though, the content of the subgraphs are not very similar, they were considered as similar by our model. This is because, our similarity threshold is relatively low and both subgraphs contain locations. On the other hand, the second matching itemsets are more similar. Because both Subgraph 415 and Subgraph 515 are about suicides. The former subgraph is about the suicide of Hrant Dink, and the latter is about the suicide of Russian Ambassador, Andrey Karlov. Finally, the predicted subgraph, Subgraph 407, is presented in Figure 6.16. The predicted subgraph's content is related to the weather conditions in the cities of Turkey. Our model could not match any subgraph in the rest of the test sequence with this subgraph.



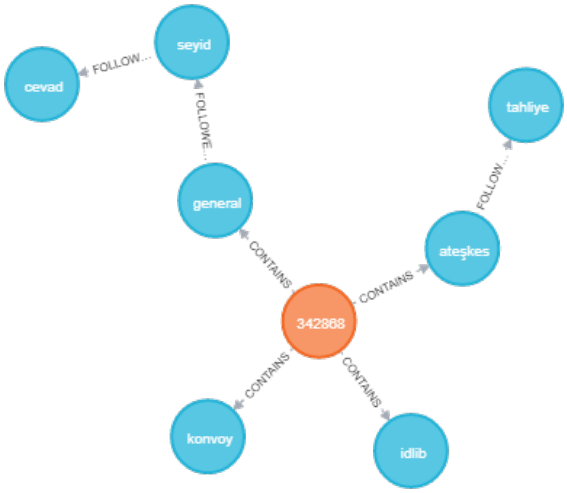
(a) Subgraph 408



(b) Subgraph 516

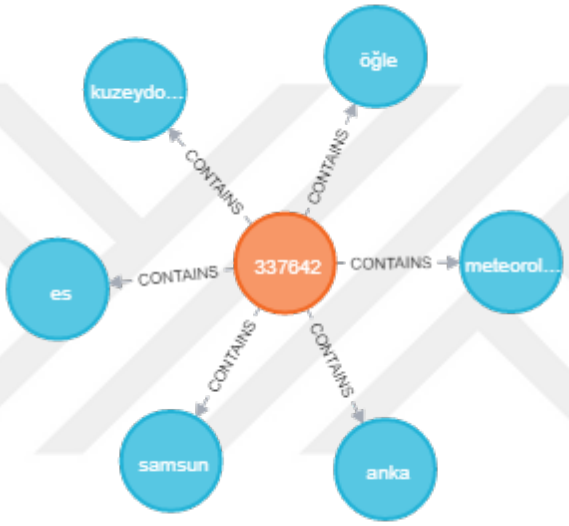


(c) Subgraph 415



(d) Subgraph 515

Figure 6.15: Case 3 Rule Antecedent - Test Sequence Matchings



(a) Subgraph 407

Figure 6.16: Case 3 Prediction

CHAPTER 7

CONCLUSION

In this thesis work, we proposed a sequential-rule based news pattern prediction model using graphs. We decided to use the power of graphs to reveal the hidden relationships within texts. Because they are commonly used in various fields, including social networks, bioinformatics, and natural language processing. Our framework consists of independent submodules that are summarized below.

The first module represents textual contents in the form of a graph. For this purpose, we proposed several ways, including noun-verb, named entity, and TF-IDF graph models. We chose to go on with the TF-IDF model because it captures sufficient information to represent the essential terms and their relationships better than other models.

Secondly, we extracted subgraphs using the gSpan [7] algorithm, a frequent subgraph mining algorithm, and formed subgraph sequences. We paid attention to subgraphs since they are the core components of graphs, and frequently repeated substructures may indicate important events. Afterward, we extracted sequential rules using the SPMF's [46] sequential rule mining framework. Sequential rules are critical component of our model since they indicate temporal patterns and relationships.

Finally, in the prediction framework, we evaluated our model's performance using the sequential rules. Our recommendations were based on the observed patterns (sequential rules), and we measured the similarity of subgraphs using different graph embedding techniques. To choose which embedding techniques are optimal for our setting, we performed a user study. According to the user study, one-hot encoding and node2vec [4] embeddings presented the most promising representations. Therefore

during our experiments, our latent representations were based on these embeddings.

Experimental results have shown that the proposed approach performed more effectively than the proposed baselines. Subgraph sequences that consist of 10 itemsets performed the best among all settings when we used all sequential rules. On the other hand, when we restricted the number of rules, it performed worst. We also measured the time required for the recommendation. According to the measurements, as we increased the number of unique rule antecedents, the duration of the process also increases linearly.

As future work, we are planning to apply deep graph embedding models because they are inherently inductive, which means that they can represent graphs that were not seen during the training. Additionally, we may change the graph structure because it is the main source of information on this model. Besides, we may also do some changes in subgraph extraction. For instance, we may consider getting cliques or relaxed clique structures such as quasi-cliques instead of frequent subgraphs.

REFERENCES

[1] X. Yan and J. Han, "Discovery of frequent substructures," *Mining graph data*, pp. 99–115, 2006.

[2] B. YILMAZ, H. Genc, M. Agriman, B. K. Demirdover, M. Erdemir, G. Simsek, and P. Karagoz, "Recent trends in the use of graph neural network models for natural language processing," in *Deep Learning Techniques and Optimization Strategies in Big Data Analytics*, pp. 274–289, IGI Global, 2020.

[3] H. Genc and B. Yilmaz, "Text-based event detection: Deciphering date information using graph embeddings," in *International Conference on Big Data Analytics and Knowledge Discovery*, pp. 266–278, Springer, 2019.

[4] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864, 2016.

[5] M. Erdemir, F. Goz, A. Mutlu, and P. Karagoz, "Comparison of querying performance of neo4j on graph and hyper-graph data model," in *KDIR*, pp. 397–404, 2019.

[6] F. Rousseau, E. Kiagias, and M. Vazirgiannis, "Text categorization as a graph classification problem," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 1702–1712, 2015.

[7] X. Yan and J. Han, "gspan: Graph-based substructure pattern mining," in *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pp. 721–724, IEEE, 2002.

[8] A. Schenker, M. Last, H. Bunke, and A. Kandel, "Clustering of web documents

using a graph model," in *Web Document Analysis: Challenges and Opportunities*, pp. 3–18, World Scientific, 2003.

[9] W. Fan, X. Wang, Y. Wu, and J. Xu, "Association rules with graph patterns," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1502–1513, 2015.

[10] M. H. Namaki, Y. Wu, Q. Song, P. Lin, and T. Ge, "Discovering graph temporal association rules," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pp. 1697–1706, 2017.

[11] X. Wang, Y. Xu, and H. Zhan, "Extending association rules with graph patterns," *Expert Systems with Applications*, vol. 141, p. 112897, 2020.

[12] D. Koutra, A. Parikh, A. Ramdas, and J. Xiang, "Algorithms for graph similarity and subgraph matching," in *Proc. Ecol. Inference Conf*, vol. 17, 2011.

[13] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph kernels," *Journal of Machine Learning Research*, vol. 11, no. Apr, pp. 1201–1242, 2010.

[14] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.

[15] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710, 2014.

[16] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo, "struc2vec: Learning node representations from structural identity," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 385–394, 2017.

[17] B. Adhikari, Y. Zhang, N. Ramakrishnan, and B. A. Prakash, "Sub2vec: Feature learning for subgraphs," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 170–182, Springer, 2018.

[18] A. Narayanan, M. Chandramohan, L. Chen, Y. Liu, and S. Saminathan, "subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs," *arXiv preprint arXiv:1606.08928*, 2016.

[19] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research*, vol. 12, no. 77, pp. 2539–2561, 2011.

[20] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, "graph2vec: Learning distributed representations of graphs," *arXiv preprint arXiv:1707.05005*, 2017.

[21] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *International conference on machine learning*, pp. 1188–1196, 2014.

[22] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[23] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, pp. 1024–1034, 2017.

[24] J. Zhang, H. Zhang, L. Sun, and C. Xia, "Graph-bert: Only attention is needed for learning graph representations," *arXiv preprint arXiv:2001.05140*, 2020.

[25] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018.

[26] A. Lerer, L. Wu, J. Shen, T. Lacroix, L. Wehrstedt, A. Bose, and A. Peysakhovich, "Pytorch-biggraph: A large-scale graph embedding system," *arXiv preprint arXiv:1903.12287*, 2019.

[27] A. Rajaraman and J. D. Ullman, *Mining of massive datasets*. Cambridge University Press, 2011.

[28] P. Fournier-Viger, J. C.-W. Lin, R. U. Kiran, Y. S. Koh, and R. Thomas, "A survey of sequential pattern mining," *Data Science and Pattern Recognition*, vol. 1, no. 1, pp. 54–77, 2017.

[29] R. Agrawal and R. Srikant, "Mining sequential patterns," in *Proceedings of the eleventh international conference on data engineering*, pp. 3–14, IEEE, 1995.

[30] R. Srikant and R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements," in *International Conference on Extending Database Technology*, pp. 1–17, Springer, 1996.

[31] J. Han, J. Pei, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu, "Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth," in *proceedings of the 17th international conference on data engineering*, pp. 215–224, Citeseer, 2001.

[32] M. J. Zaki, "Spade: An efficient algorithm for mining frequent sequences," *Machine learning*, vol. 42, no. 1-2, pp. 31–60, 2001.

[33] P. Fournier-Viger, A. Gomariz, M. Campos, and R. Thomas, "Fast vertical mining of sequential patterns using co-occurrence information," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 40–52, Springer, 2014.

[34] H. Cheng, X. Yan, and J. Han, "Mining graph patterns," in *Frequent pattern mining*, pp. 307–338, Springer, 2014.

[35] M. Kuramochi and G. Karypis, "An efficient algorithm for discovering frequent subgraphs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1038–1051, 2004.

[36] A. Inokuchi, T. Washio, and H. Motoda, "An apriori-based algorithm for mining frequent substructures from graph data," in *European conference on principles of data mining and knowledge discovery*, pp. 13–23, Springer, 2000.

[37] C. Borgelt and M. R. Berthold, "Mining molecular fragments: Finding relevant substructures of molecules," in *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pp. 51–58, IEEE, 2002.

[38] J. Huan, W. Wang, and J. Prins, "Efficient mining of frequent subgraphs in the presence of isomorphism," in *Third IEEE International Conference on Data Mining*, pp. 549–552, IEEE, 2003.

[39] S. Nijssen and J. N. Kok, "A quickstart in frequent structure mining can make a difference," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 647–652, 2004.

[40] N. S. Ketkar, L. B. Holder, and D. J. Cook, "Subdue: Compression-based frequent pattern discovery in graph data," in *Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations*, pp. 71–76, 2005.

[41] P. Fournier-Viger, C. Cheng, J. C.-W. Lin, U. Yun, and R. U. Kiran, "Tkg: Efficient mining of top-k frequent subgraphs," in *International Conference on Big Data Analytics*, pp. 209–226, Springer, 2019.

[42] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, pp. 3111–3119, 2013.

[43] R. F. Cekınel, M. Ağriman, P. Karagöz, and B. Yilmaz, "Named entity recognition with conditional random fields on turkish news dataset: Revisiting the features," in *2019 27th Signal Processing and Communications Applications Conference (SIU)*, pp. 1–4, IEEE, 2019.

[44] G. Tür, D. Hakkani-Tür, and K. Oflazer, "A statistical information extraction system for turkish," *Natural Language Engineering*, vol. 9, no. 2, pp. 181–210, 2003.

[45] G. A. Seker and G. Eryigit, "Extending a crf-based named entity recognition model for turkish well formed text and user generated content.," *Semantic Web*, vol. 8, no. 5, pp. 625–642, 2017.

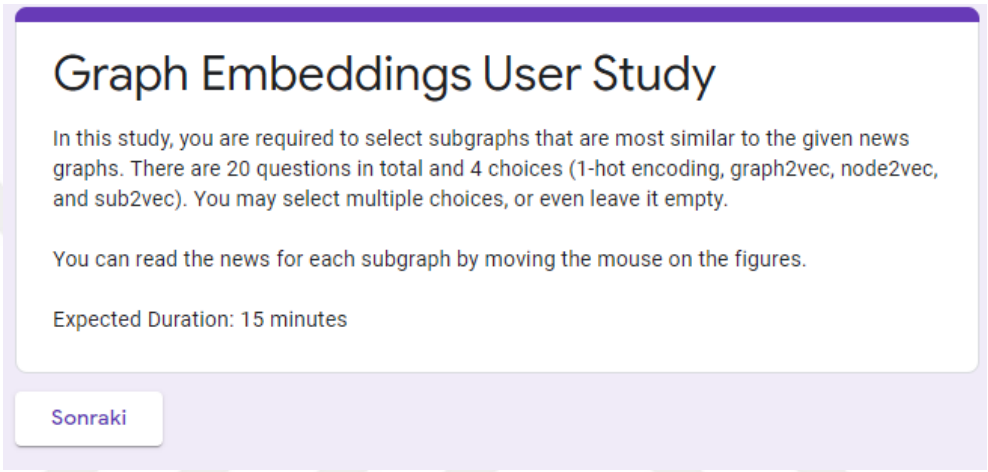
[46] P. Fournier-Viger, J. C.-W. Lin, A. Gomariz, T. Gueniche, A. Soltani, Z. Deng, and H. T. Lam, "The spmf open-source data mining library version 2," in *Joint European conference on machine learning and knowledge discovery in databases*, pp. 36–40, Springer, 2016.

[47] A. Sanfeliu and K.-S. Fu, "A distance measure between attributed relational graphs for pattern recognition," *IEEE transactions on systems, man, and cybernetics*, no. 3, pp. 353–362, 1983.

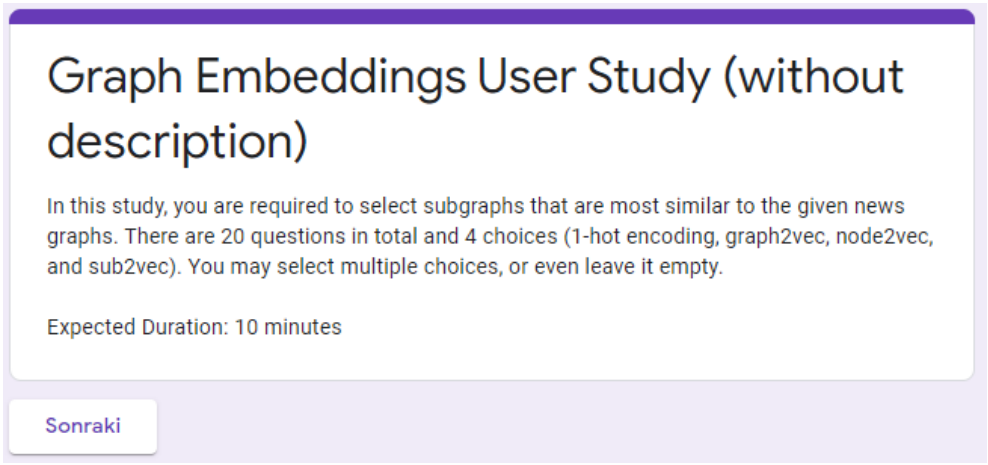
[48] Z. Zeng, A. K. Tung, J. Wang, J. Feng, and L. Zhou, "Comparing stars: On approximating graph edit distance," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 25–36, 2009.

APPENDIX A

USER STUDY SAMPLE SURVEY QUESTIONS



(a) With contents



(b) Without contents

Figure A.1: Information screens for the user study

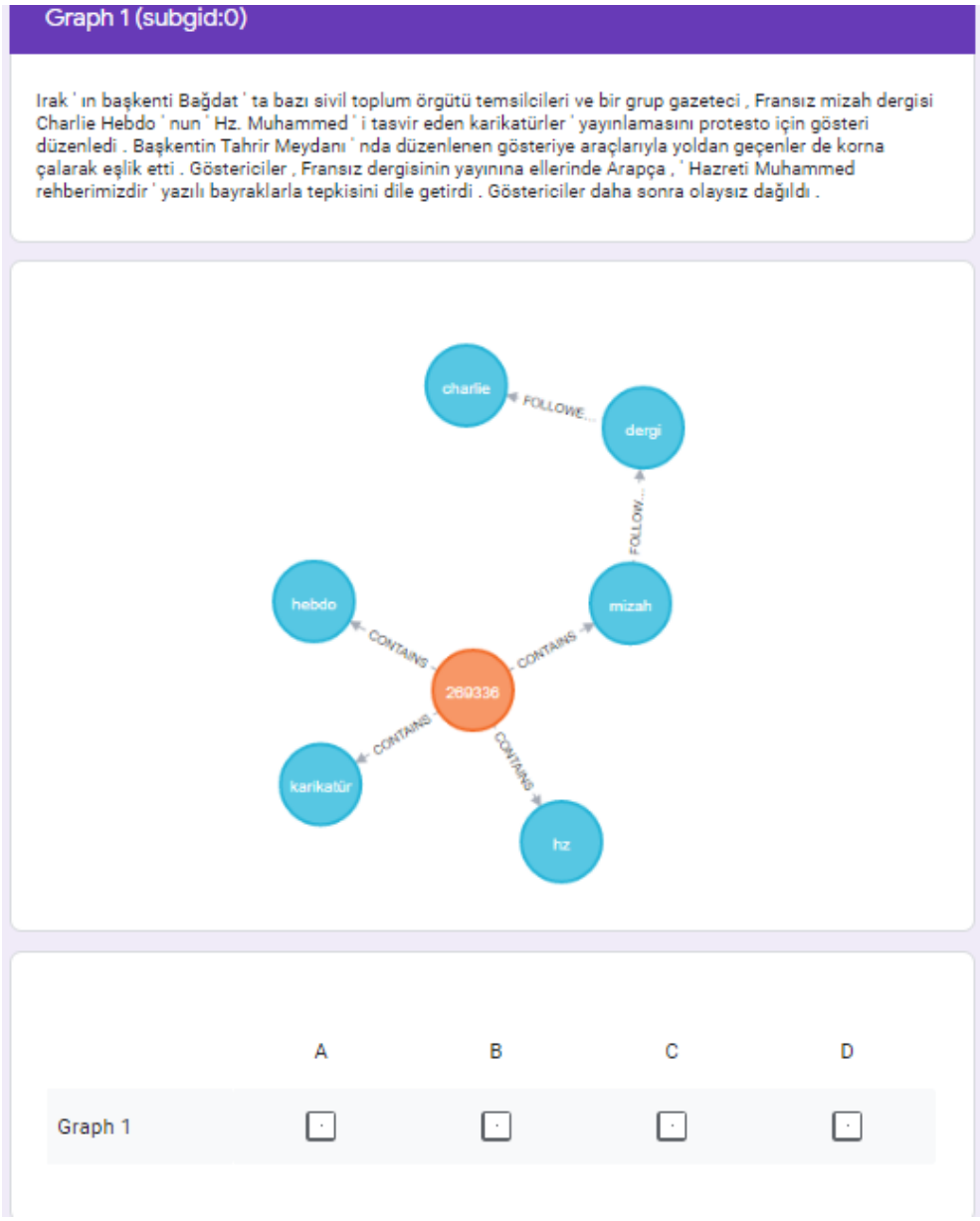
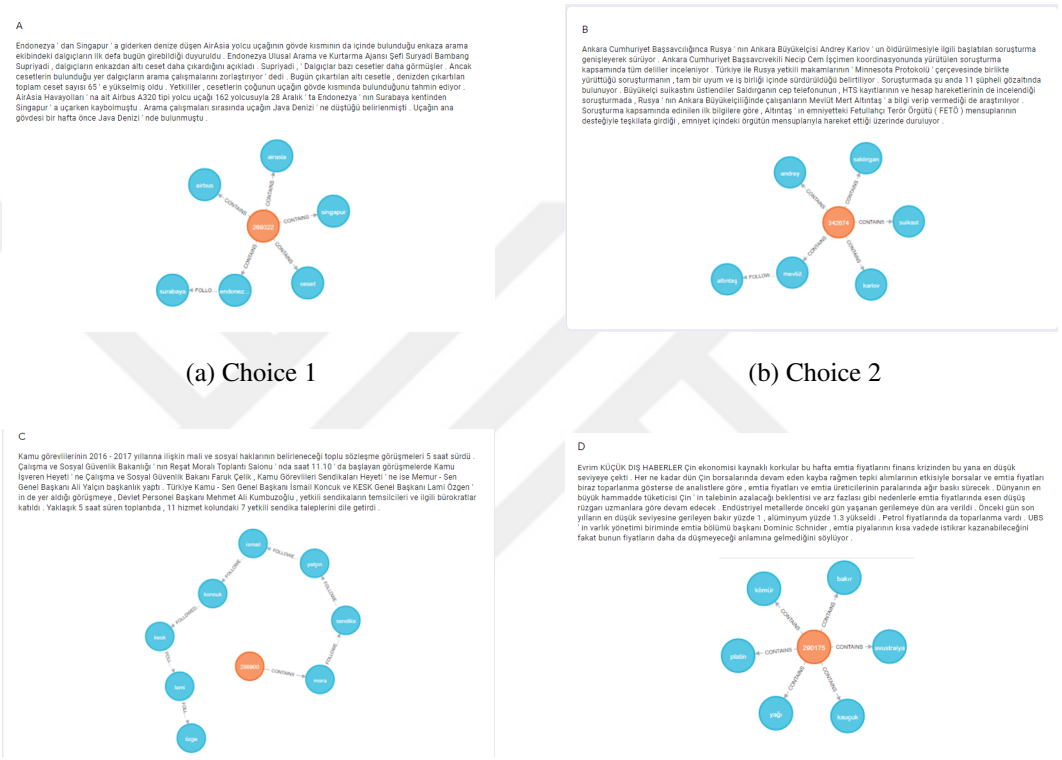


Figure A.2: The first survey question with content



(a) Choice 1

(b) Choice 2

(c) Choice 3

(d) Choice 4

Figure A.3: Choices for the first question

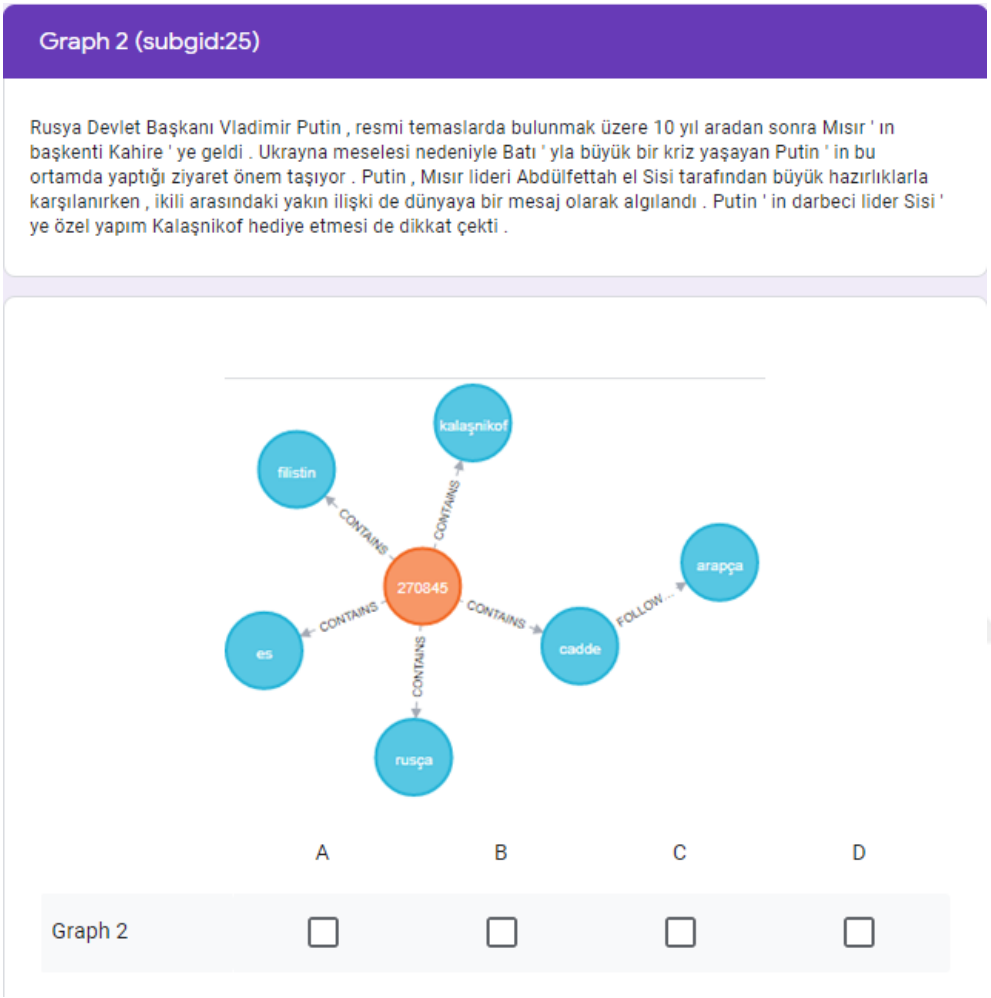
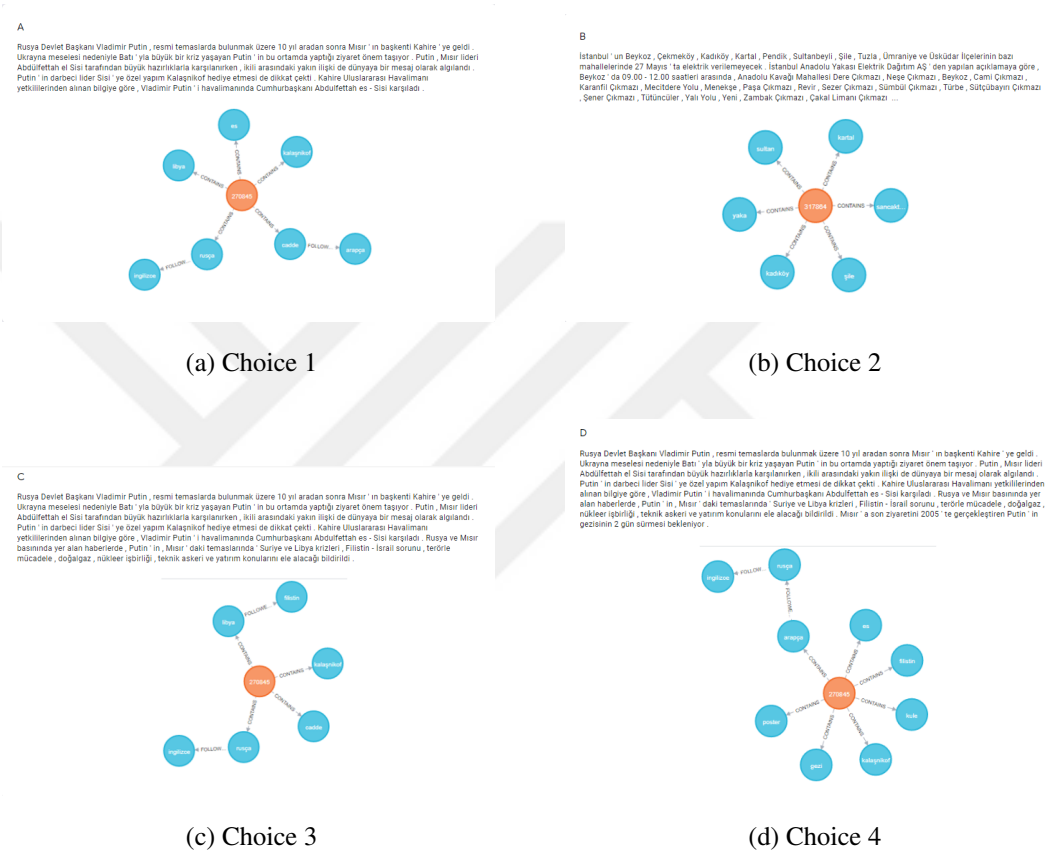


Figure A.4: The second survey question with content



(a) Choice 1

(b) Choice 2

(c) Choice 3

(d) Choice 4

Figure A.5: Choices for the second question

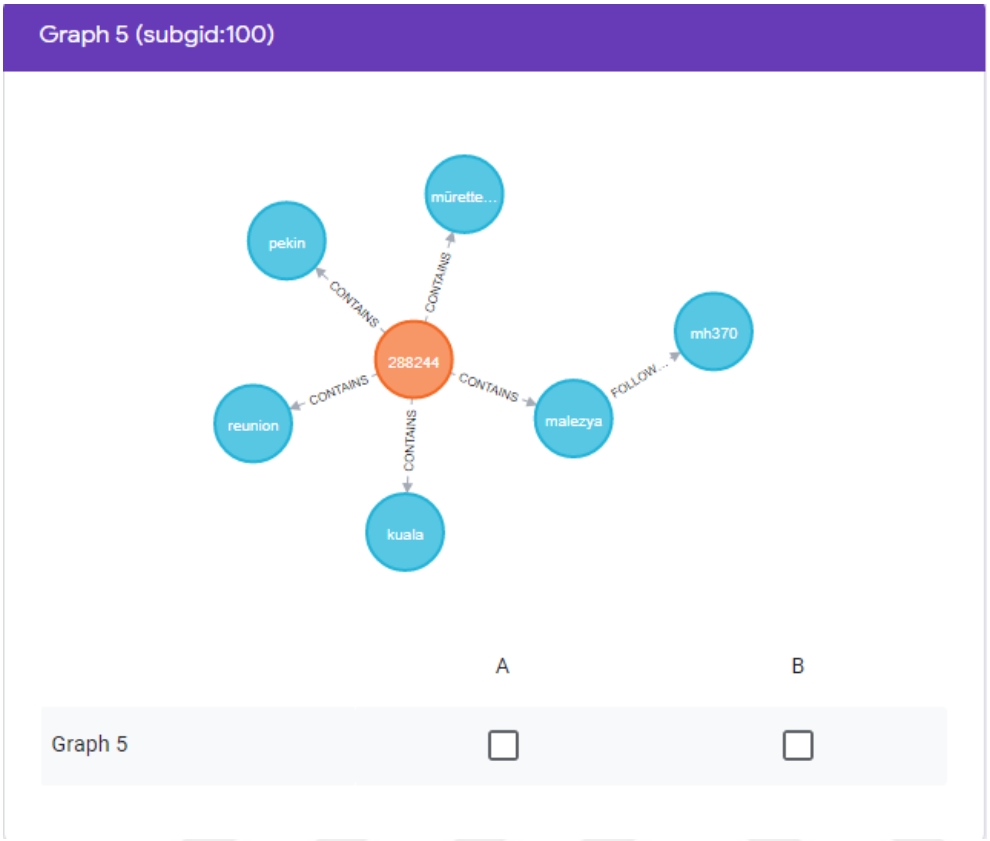
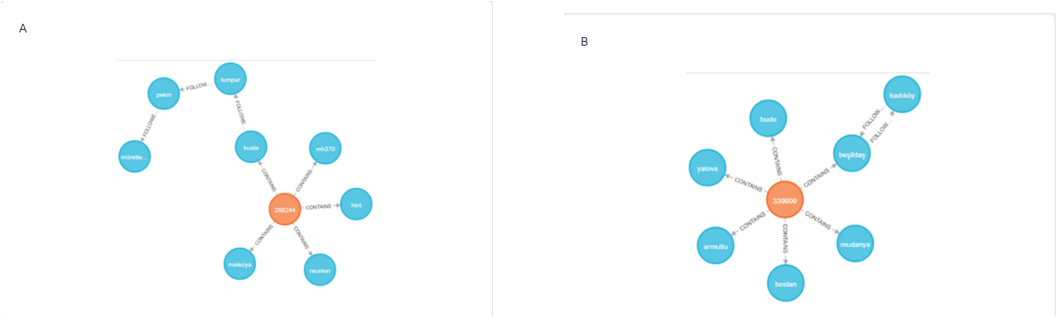


Figure A.6: The fifth survey question without content



(a) Choice 1

(b) Choice 2

Figure A.7: Choices for the fifth question

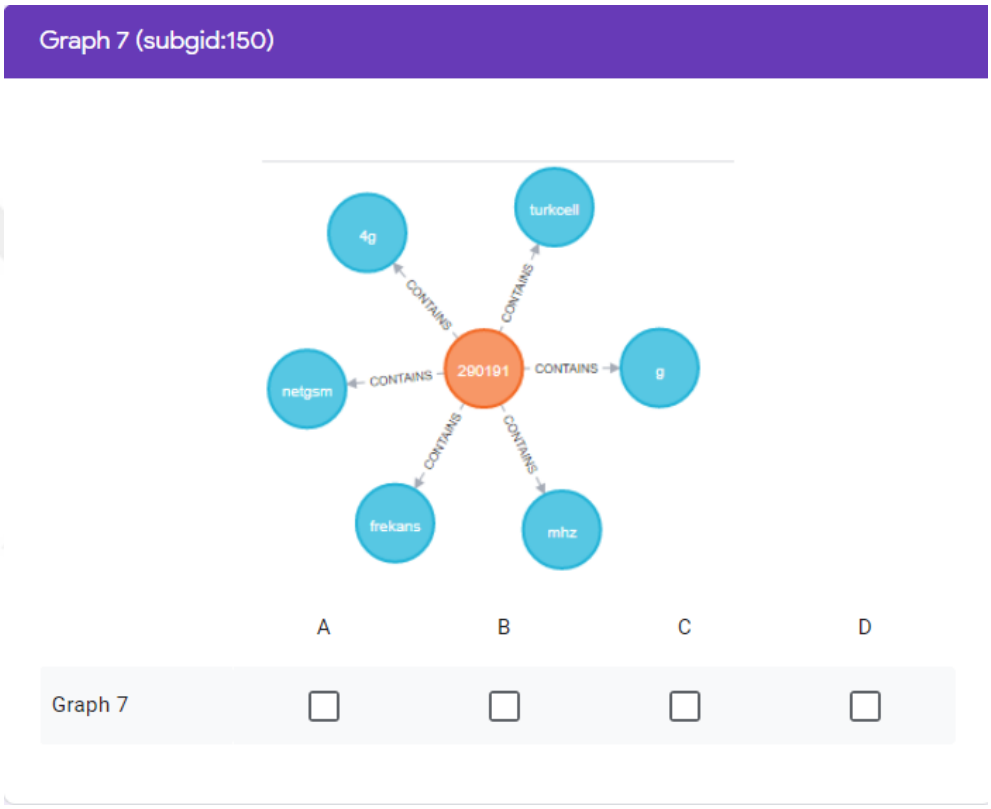
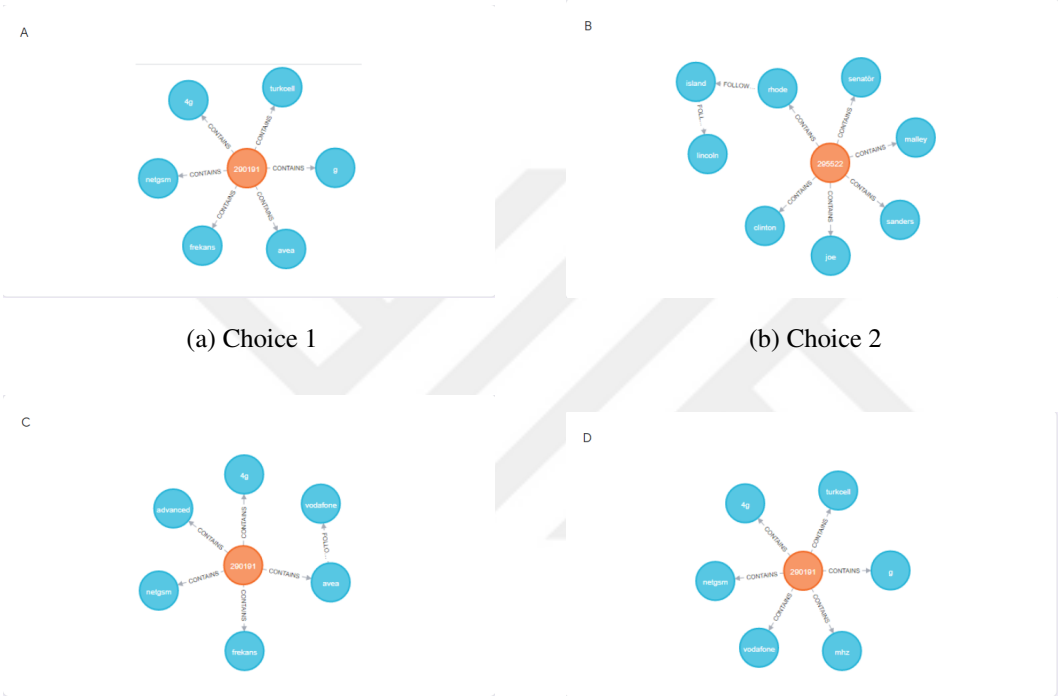


Figure A.8: The seventh survey question without content



(a) Choice 1

(b) Choice 2

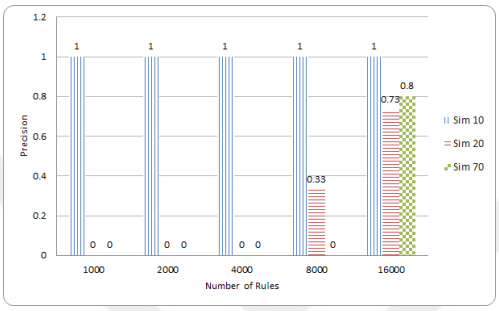
(c) Choice 3

(d) Choice 4

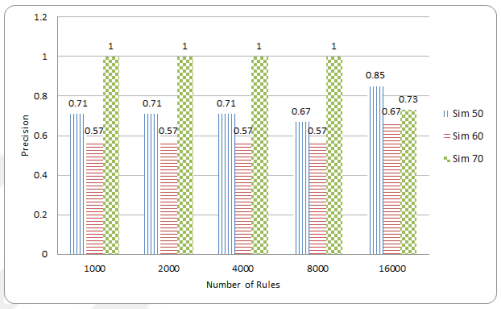
Figure A.9: Choices for the seventh question

APPENDIX B

PRECISION AND RECALL SCORES OF THE EXPERIMENT 2

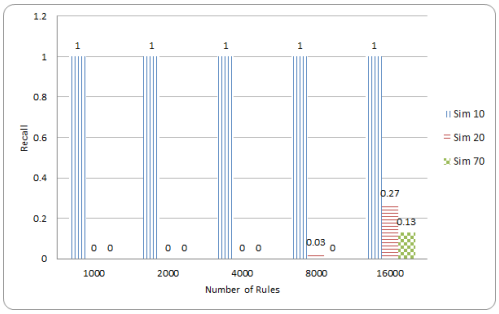


(a) One-hot encoding

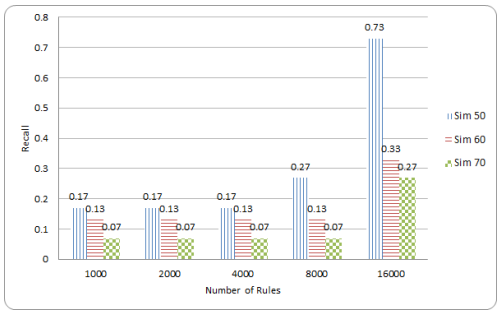


(b) node2vec

Figure B.1: Precision scores vs top k interesting rules for *Seq5*

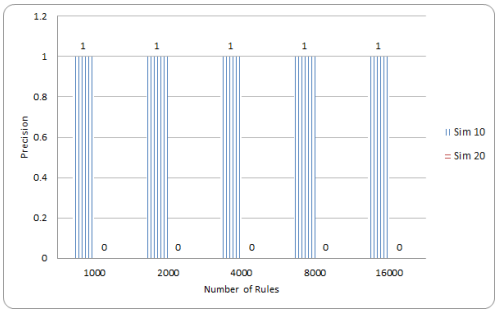


(a) One-hot encoding

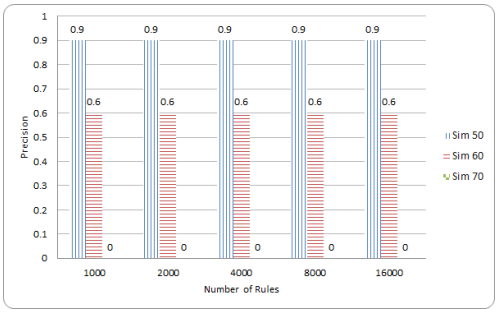


(b) node2vec

Figure B.2: Recall scores vs top k interesting rules for *Seq5*

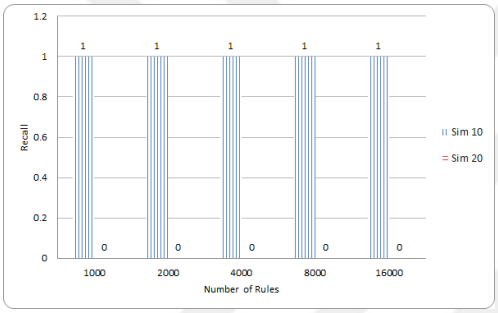


(a) One-hot encoding

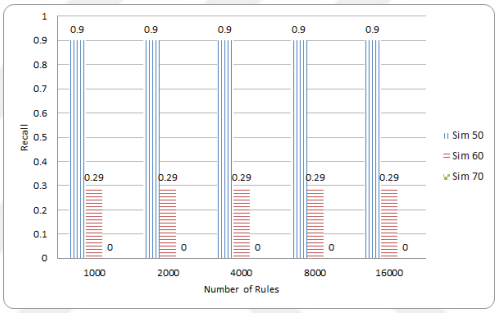


(b) node2vec

Figure B.3: Precision scores vs top k interesting rules for *Seq7*

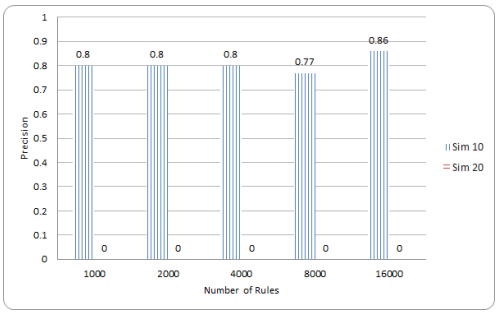


(a) One-hot encoding

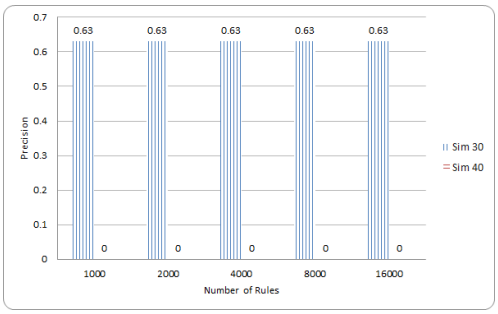


(b) node2vec

Figure B.4: Recall scores vs top k interesting rules for *Seq7*

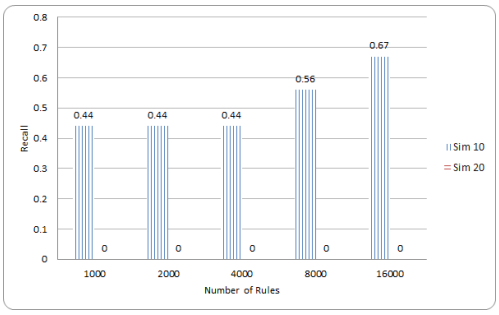


(a) One-hot encoding

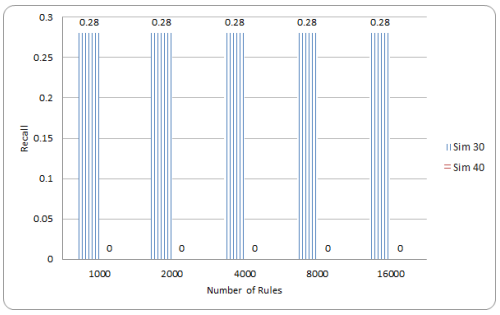


(b) node2vec

Figure B.5: Precision scores vs top k interesting rules for *Seq10*

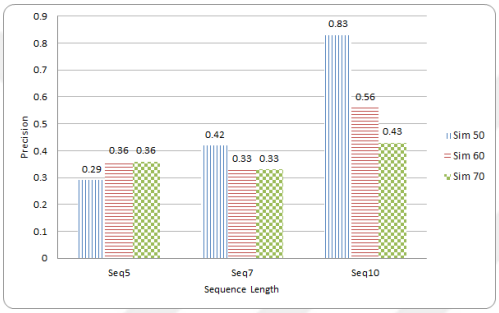


(a) One-hot encoding

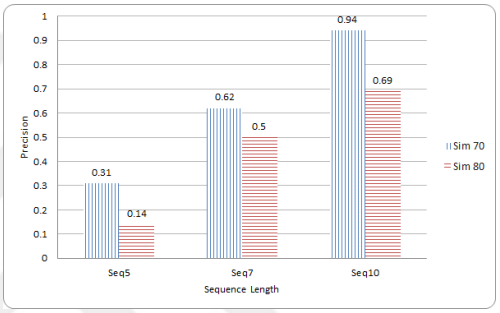


(b) node2vec

Figure B.6: Recall scores vs top k interesting rules for *Seq10*

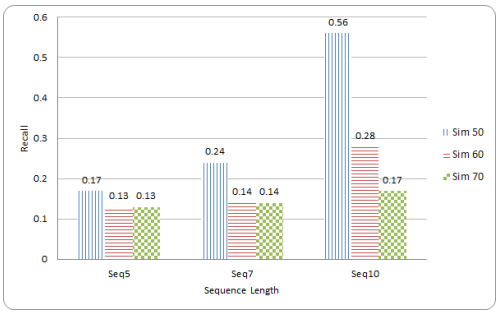


(a) One-hot encoding

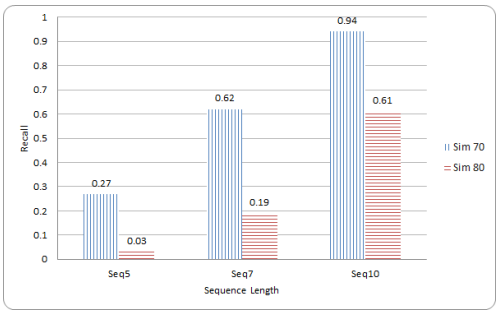


(b) node2vec

Figure B.7: Precision scores vs sequence length for various similarity threshold using all rules



(a) One-hot encoding



(b) node2vec

Figure B.8: Recall scores vs sequence length for various similarity threshold using all rules