

**ANALYSIS OF FEATURE PATTERN MINING
APPROACHES ON SOCIAL NETWORK: A CASE
STUDY ON FACEBOOK**

**A Thesis Submitted to
the Graduate School of Engineering and Sciences of
İzmir Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of**

MASTER OF SCIENCE

in Computer Engineering

**by
Elife ÖZTÜRK**

**December 2017
İZMİR**

We approve the thesis of **Elife ÖZTÜRK**

Examining Committee Members:



Asst. Prof. Dr. Selma TEKİR

Department of Computer Engineering, İzmir Institute of Technology



Asst. Prof. Dr. Damla OĞUZ

Department of Software Engineering, Yaşar University



Asst. Prof. Dr. Serap ŞAHİN

Department of Computer Engineering, İzmir Institute of Technology

25 December 2017



Asst. Prof. Dr. Serap ŞAHİN

Supervisor, Department of Computer Engineering
İzmir Institute of Technology

Assoc. Prof. Dr. Yusuf Murat ERTEN

Head of the Department of
Computer Engineering

Prof. Dr. Aysun SOFUOĞLU

Dean of the Graduate School of
Engineering and Sciences

ACKNOWLEDGMENTS

First of all, I would like to express my precious gratitude to my advisor, *Asst. Prof. Dr. Serap Şahin* who provide full support at all stages of my thesis process and contribute to my academic development.

I would also like to thank *Asst. Prof. Dr. Selma Tekir* and *Assoc. Prof. Dr. Belgin Ergenç*, for their suggestions and evaluations in some stages of this study.

I would like to thank all of my colleagues for their warm friendship and desirable behaviors.

And finally, I would like to express my endless thanks to my family for attention, motivation, and love.

ABSTRACT

ANALYSIS OF FEATURE PATTERN MINING APPROACHES ON SOCIAL NETWORK: A CASE STUDY ON FACEBOOK

Pattern mining algorithms obtain patterns frequently seen in a database and complex graphs which are available from gene networks to social networks. Complex graphs contain lots of valuable information on their nodes or edges. For this reason, pattern mining algorithms can be used to extract data from complex networks. However, these algorithms usually work on the graphs whose nodes have a single label. If these algorithms are implemented on multi labeled (multi-attributed) complex graphs, their complexities belong to NP-Complete. For this reason, in this study, different approaches have been evaluated to find patterns. The goal is to understand related methods and algorithms with their pros and cons to obtain common feature patterns from multi-attributed complex graphs. We also selected Facebook social network complex graph data set (SNAP - Stanford University FaceBook anonymized data set) as an application domain and we analyzed the most frequent feature patterns on friendship relations.

ÖZET

SOSYAL AĞLARDA ÖZELLİK ÖRÜNTÜ MADENCİLİĞİ YAKLAŞIMLARININ ANALİZİ: FACEBOOK ÜZERİNDE DURUM ÇALIŞMASI

Günümüzde, kompleks çizgeler gen ağlarından sosyal ağlara kadar her alanda bulunmaktadır. Kompleks çizgeler diğer yapılara göre daha fazla veri içerdiğinden, kompleks çizgelerin madenciliği sonucunda daha anlamlı ve değerli bilgiler elde edilebilir. Bu çalışmada kompleks çizgelerden ortak özelliklere sahip örüntüler elde edilmek istenmiştir. Örüntü madenciliği algoritmaları bir veritabanından sıklıkla görülen örüntülerin elde edilmesini sağlar. Bu nedenle kompleks ağlardan veri elde edebilmek için örüntü madenciliği algoritmaları kullanılabilir. Ancak bu algoritmalar genellikle düğümleri tek etikete sahip olan çizgeler üzerinde çalışmaktadır. Bu algoritmalar, sosyal ağ çizgeleri gibi çok etiketli kompleks çizgelere uygulandığında maalesef algoritmaların karmaşıklık derecesi NP-tam sınıfına ait olmaktadır. Bu nedenle, bu çalışma kapsamında ilgili çizge algoritmaları, çizgenin veri setlerine dönüştürülmesi ve dönüşüm sonrası oluşturulan data setinde standart desen bulma algoritmalarının kullanılması gibi yöntemler incelenmiştir. Bu yöntemler güçlü ve zayıf yönleri ile, çok etiketli kompleks çizgelerinde desen analizi hedefi için değerlendirilmiştir. Bu çalışma sürecinde, uygulama alanı olarak Facebook SNAP veri seti kullanılmış, arkadaşlık ilişkilerinde en yüksek sıklıkla görülen ortak etiket deseni araştırılmıştır.

TABLE OF CONTENTS

LIST OF FIGURES	viii
LIST OF TABLES	ix
LIST OF ABBREVIATIONS	x
CHAPTER 1. INTRODUCTION	1
1.1. Thesis' Motivation	2
1.2. Thesis' Aim and Contributions	3
1.3. Methodology of Thesis.....	4
1.4. Outline of Thesis	4
CHAPTER 2. DATASET	6
CHAPTER 3. GRAPH-BASED FEATURE PATTERN MINING	12
3.1. Background.....	12
3.1.1. Candidate Generation	14
3.1.1.1. Apriori-based Approach	14
3.1.1.2. Pattern-Growth Approach	14
3.1.2. Support Computation	15
3.1.2.1. Support Computation in a Set of Graphs	16
3.1.2.2. Support Computation in a Single Graph.....	16
3.2. Related Work	18
3.3. Algorithms.....	19
3.3.1. gSpan Algorithm	20
3.3.2. Grami Algorithm	23
3.3.2.1. Explanation of Grami Algorithm	24
3.3.2.2. Grami Algorithm using Single-label Graph	25
3.3.2.3. Grami Algorithm using Multi-label Graph	26
3.4. Experimental Work.....	28
3.4.1. Complexity Analysis of gSpan Algorithm	29

3.4.2. Complexity Analysis of Grami Algorithm	29
3.4.3. gSpan Algorithm Results from ego_network_3980	30
3.4.4. gSpan Algorithm Results from ego_network_698	31
3.4.5. Grami Algorithm Results from ego_network_3980	31
3.5. Results	32
CHAPTER 4. TRANSFORMATION OF GRAPH DATA TO TRANSACTIONAL DATA	33
4.1. Background and Related Work	33
4.2. Experiment	35
4.3. Result	36
CHAPTER 5. ITEMSET PATTERN MINING ON TRANSACTIONAL DATA	38
5.1. Background	38
5.2. Algorithms	39
5.2.1. Apriori Algorithm	39
5.2.2. FP-Growth Algorithm	41
5.2.3. Max-Miner Algorithm	42
5.3. Experimental Work	44
5.3.1. Analysis of Itemset Mining Algorithms	46
5.3.1.1. Analysis of Apriori Algorithm	46
5.3.1.2. Analysis of FP-Growth Algorithm	49
5.3.1.3. Analysis of Max-Miner Algorithm	49
5.4. Results	50
CHAPTER 6. EVALUATION OF IMPLEMENTED METHODS	51
CHAPTER 7. CONCLUSION AND FUTURE WORK	52
REFERENCES	54
APPENDIX A. PSEUDOCODE OF ALGORITHMS	58

LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
1.1 Example of attributed graph data.	1
1.2 Figure shows the nodes that have common features with the same color.	2
2.1 Facebook Dataset.	6
2.2 Smallest Ego Network from Facebook Dataset.	7
2.3 Ego nodes with degrees.	8
2.4 First and second smallest ego networks	10
2.5 Small samples with node ids are shown from networks.	10
3.1 Examples from directed and undirected graphs	13
3.2 Example of graph and subgraph isomorphism	13
3.3 Two k-size similar graphs join to generate candidate	15
3.4 Pattern-growth approach explanation	15
3.5 Frequent pattern example from graph dataset	16
3.6 Violation of downward closure property	17
3.7 MNI technique	18
3.8 Example of attributed graph data.	20
3.9 Separated subgraphs according to node features.	21
3.10 Two simple graphs for gSpan example.	22
3.11 Possible frequent edges are generated from two graphs.	22
3.12 Right most path extensions from C_1	23
3.13 Node images.	24
3.14 Grami single-labeled graph and pattern example.	26
3.15 Multi-Labeled Graph	27
3.16 Grami support computation of a multi-labeled pattern	28
5.1 The Apriori Algorithm on sample dataset	40
5.2 Frequent 1-itemsets are found and sorted	42
5.3 Step 1. FP-Tree Construction	43
5.4 Step 2. Constructing Conditional FP-Trees to generate itemsets	44
5.5 Frequent Itemsets	45
5.6 Working procedure of MaxMiner algorithm.	46

LIST OF TABLES

<u>Table</u>	<u>Page</u>
2.1 The number of nodes and edges of all ego networks are given in the table.	7
2.2 Facebook dataset includes 26 distinct features.	9
2.3 Features and ids of Ego_network_698	9
2.4 Table shows the some examples from 698.feaname file.	11
3.1 Frequent pattern mining algorithms	19
3.2 Support values of initial edges in Graph G.	28
3.3 gSpan algorithm results on the smallest ego network 3980.	30
3.4 gSpan algorithm results on the smallest ego network 698.	31
3.5 Grami algorithm results on the smallest dataset 3980.	32
4.1 Some nodes and their feature ids are shown in the table.	36
4.2 This table shows common features of nodes.	36
4.3 Properties of two smallest ego networks	37
5.1 Result of the smallest ego network	47
5.2 Result of the second smallest ego network	48
5.3 Execution time of algorithms according to given thresholds	48

LIST OF ABBREVIATIONS

SN	Social Network
BFS	Breadth First Search
DFS	Depth First Search
gSpan	Graph-based Substructure Pattern Mining
MIS	Maximum Independent Set Support Computation
HO	Harmful Overlap Support Computation
MNI	Minimum Image Based Support Computation
AGM	Apriori-based Graph Mining
FSG	Frequent Subgraph Discovery
CSP	Constraint Satisfaction Problem
ISG	Itemset-based Subgraph Mining

CHAPTER 1

INTRODUCTION

Graph data are used for a variety of subjects from DNA and protein sequences to social networks (SN). They are becoming increasingly preferred over relational databases to represent real-world and complex relationships [30, 3]. These relationships can be shown with different kinds of graphs such as chemical components are modeled as small graphs, and SNs are represented as a single large graph [1].

Graph pattern mining aims to discover frequent patterns in a graph dataset which can be small set of graphs or a single large graph. In graph pattern mining studies, graphs are generally represented with one label for one vertex; these are called *single-labeled graphs*. However, the vertices in large graphs such as those representing SNs related with additional information called features or attributes. Such graphs are called *multi-labeled* and corresponding networks are called *complex* or *attributed*. A small example of attributed graph from the dataset used in our study is shown in Figure 1.1, which indicates a social network with profile information regarding gender, location and education for some of its members. Many people in this network may have the same features; this is called the *common feature pattern*, which this study is interested in. Common feature patterns indicate groups that are connected and have common features in a network.

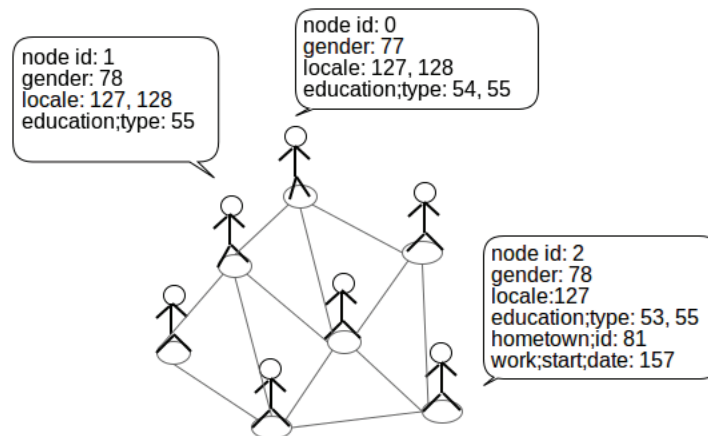


Figure 1.1. Example of attributed graph data.

These complex graphs make it possible to obtain more precious patterns than single-labeled graphs using node attributes or features. For example, it is possible to understand how many groups have the same feature patterns and which nodes create these patterns in an entire network. This information can be used to summarize a network. For example, Figure 1.2 shows an attributed network such as red colored nodes have two common features, and green colored nodes have three common features. It means that these nodes have the same features. Then results show that there are two different feature patterns and red colored patterns are commonly seen in this network.

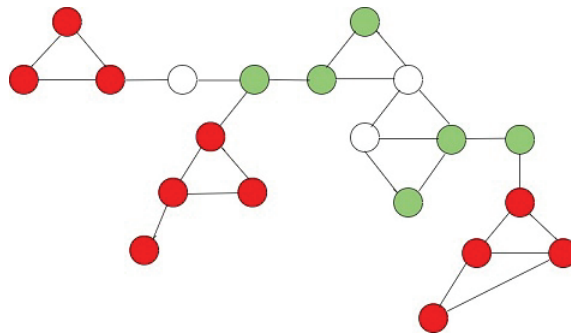


Figure 1.2. Figure shows the nodes that have common features with the same color.

In graph mining, most of the studies focus on single-labeled graphs to obtain patterns. However, finding patterns from multi-labeled graphs are more difficult than single-labeled since such patterns can be found by preserving graph structure and identifying common attributes as itemsets. Therefore, to analyze complex networks, in this study, different approaches are evaluated from graph pattern mining and itemset pattern mining.

1.1. Thesis' Motivation

Frequent pattern mining is a sub-branch of graph mining that finds frequently seen patterns in a graph. To obtain the most common feature patterns, we use frequent pattern mining algorithms. However, in pattern mining, graphs are generally represented using one label for one vertex. For this reason, a whole graph is separated into subgraphs related to each attribute. We obtain many subgraphs and should be able to find the common graph patterns among these subgraphs. This method requires high cost in memory and

CPU time. Therefore, the motivation of this thesis is to use more economic ways to find common feature sets from multi-labeled graphs. The aim, methodology and structure of this thesis are given in the following sections.

1.2. Thesis' Aim and Contributions

The aim of this thesis is to obtain feature patterns that are commonly seen in a multi-labeled graph such as an SN. The most common feature patterns on labeled graph related with frequent pattern mining on graphs. Therefore, frequent pattern mining algorithms are used to find commonly seen feature sets. However, as mentioned above, generally, single-labeled graphs are used for pattern mining algorithms. Therefore, this study evaluates the suitability of attributed network and how it can be applied to find feature sets. Two types of algorithms are investigated in this study to find common features. First, since SNs are represented as graphs, graph mining algorithms are used. Second, since attributes of network members are considered as itemsets, itemset mining algorithms are used after graph transformation. After that, time complexity of algorithms is evaluated. The objectives of this thesis are:

- To understand graph feature pattern mining algorithms.
- To show the problem of feature pattern mining on multi labeled graphs.
- To handle this problem, transformation of graph data to itemsets for using itemset mining algorithms
- Evaluation of these algorithms according to our pattern mining objectives and their complexities.

This thesis contributes to the literature by:

- Demonstrating the problem of obtaining common patterns in large and multi labeled graphs using frequent pattern mining algorithms.
- Modifying Grami algorithm [10] to work on multi-labeled graph data.
- Implementing Apriori [5], Fp-Growth [16] and Max-Miner [6] algorithms on an SN dataset that is transformed from graph data to transactional data.

- Finding common attribute sets by converting graph data to transactional data.
- Showing the similarity of results among analysed and implemented algorithms.

1.3. Methodology of Thesis

In this thesis, we apply both graph and itemset mining approaches to obtain common feature patterns in an SN. gSpan and Grami algorithms are selected from graph pattern mining, Apriori, FP-Growth and Max-Miner algorithms are used from itemset pattern mining.

- For gSpan algorithm, new graphs are created from single large graph according to nodes with a specific single feature. Since gSpan algorithm takes one label for one node, attributed graph dataset is converted for one label, then gSpan algorithm is applied on this input data.
- For Grami algorithm, modification is made on the code implementation because although it can be applied on multiple labeled graph data according to authors [10], currently deployed version does not support multiple labels. Therefore, first, we modified Grami algorithm and then it is applied to selected dataset.
- Subgraph mining algorithms check on graph structure however, itemset mining algorithms do not need to check structure matching and it can be more efficient than subgraph mining algorithms. Therefore, dataset is converted into itemsets and then, itemset mining algorithms Apriori, FP-Growth and Max-Miner are applied.
- Results are compared with regard to patterns found. All experiments are conducted on 2.6 GHz Intel Core i7 PC with 12 GB of RAM and 1 TB HDD.

1.4. Outline of Thesis

The organization of the thesis as follows :

- Chapter 2 shows the used dataset information.

- Chapter 3 explains studies, experimental work and analysis of graph-based feature pattern mining algorithms.
- Chapter 4 shows transformation of graph data to transactional data.
- Chapter 5 explains studies, experimental work and analysis of itemset pattern mining algorithms.
- Chapter 6 evaluates the implemented methods
- Chapter 7 concludes this thesis and it gives a summary and future research.

CHAPTER 2

DATASET

In this thesis, we used Facebook dataset which was obtained from Stanford Network Analysis Platform (SNAP) that includes a collection of large network datasets. In order to get profile and network information from Facebook members, a Facebook application was used by Leskovec et al. from Stanford University [25]. This dataset includes 10 ego-networks for different 10 facebook members as illustrated in Figure 2.1, consist of 4.039 nodes (members and their friends) and 88.234 edges (relations).

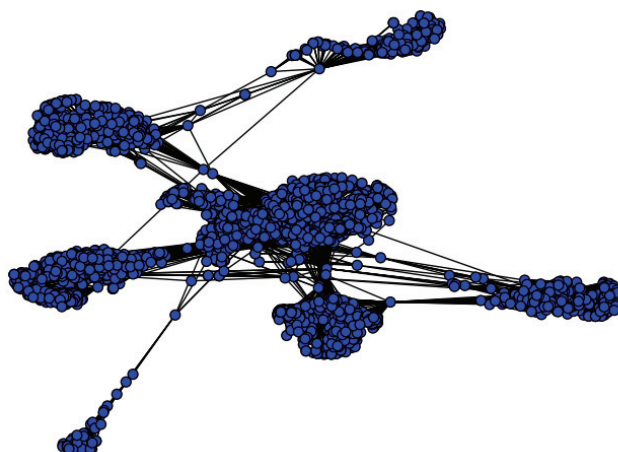


Figure 2.1. Facebook Dataset.

Figure 2.2 shows the smallest network in Facebook dataset in order to understand the ego network structure better. The red color represents ego node and others represent alters. Ego node has the largest friendship connection as a hub point and other nodes (alters) directly connected to ego node. Alters may have a connection to each other. In addition to this, an ego node may have a connection to any node from another ego network.

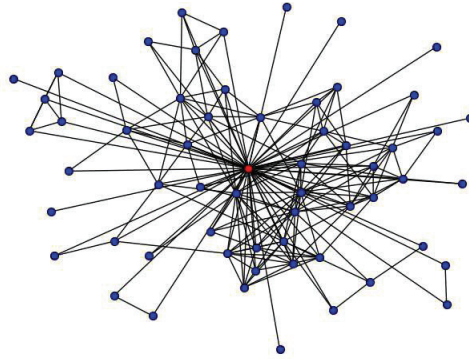


Figure 2.2. Smallest Ego Network from Facebook Dataset.

In Facebook dataset, ego nodes are defined by numbers called node id. For example, ego node id's are 0, 107, 348, 414, 686, 698, 1684, 1912, 3437, 3980. Figure 2.3 shows the degree of each ego node which defines the number of friendship connection. As we understand from Figure 2.3, *ego_network_107* is the largest ego network with 1045 connections and *ego_network_3980* is the smallest ego network with 59 connected nodes.

Table 2.1 shows edges and nodes in ego networks. Using this table we can decide which ego networks can be chosen for the algorithms. For example, gSpan algorithm supports small networks. Hence, we select the smallest networks to run the algorithm.

Table 2.1. The number of nodes and edges of all ego networks are given in the table.

	Ego node id									
	0	107	348	414	686	698	1684	1912	3437	3980
# of nodes	348	1046	230	160	171	69	793	756	548	60
# of edges	2866	27795	3441	1857	1831	367	14817	30780	5360	205

According to given dataset information, feature data contain 26 categories consisting users' gender, education, work location, hometown information etc. Table 2.2 shows all features in Facebook dataset.

Each feature is divided into subcategories for instance, "gender" feature contains

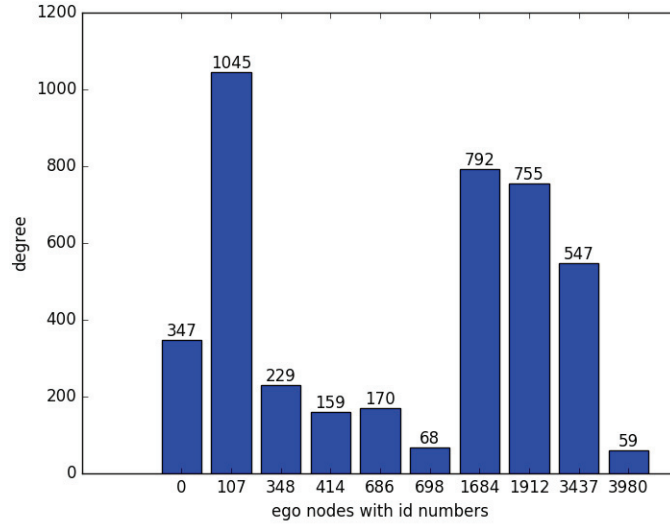


Figure 2.3. Ego nodes with degrees.

two feature ids (Fid) 77 and 78 to represent gender types; male and female. Members may have many different school id. Therefore there may be several subcategories that belong to feature "education;school;id". A few example is given from the dataset: it includes 346 different "education;school;id", 54 different "location;id", 10 "locale" and 3 different "education;type" features. From 26 categories, 1283 different features are obtained. Every node has feature vector which shows whether an information about a category exists or not. For instance, looking at feature vector of a node we can see that this user has specific location information. In this dataset, the mostly seen feature is 127 which specify a locale information. However, we do not know real location information because this datum is anonymous. Nevertheless, using the anonymized data we can understand whether two users have the same location feature.

In order to show the graph representation two smallest ego networks from Facebook dataset are chosen. As shown in Figure 2.4a and 2.4b, the first and second smallest ego networks with ids 3980 and 698 are shown.

Small sample groups are taken from these two networks as illustrated in Figure 2.5a and Figure 2.5b. Each part includes 4 nodes from network 3980 and 698, and node ids which belong to these networks are 3990, 4007, 4016, 4025 and 857, 862, 865, 868 respectively.

Table 2.4. Table shows the some examples from 698.featname file.

0	birthday	anonymized feature 2
1	birthday	anonymized feature 3
2	education;classes;id	anonymized feature 335
3	education;classes;id	anonymized feature 336
4	education;concentration;id	anonymized feature 14
5	education;degree;id	anonymized feature 22
6	education;school;id	anonymized feature 340
7	education;school;id	anonymized feature 341

CHAPTER 3

GRAPH-BASED FEATURE PATTERN MINING

The goal of graph pattern mining is to obtain frequent subgraphs whose occurrences are higher than a minimum support in a single graph or set of graphs [4]. First, basic terms are explained in this section. Then, related work on attributed graph and selected algorithms and experimental work are demonstrated.

3.1. Background

Graphs provide structural representation to analyze real networks. Social Networks are represented as different kinds of graphs. In a social network (SN), nodes are people and links are relations between nodes. A graph $G = (V, E)$ corresponding to this network consists of a set of vertices V , set of edges E that connect the vertices. Therefore, in this study, vertices and edges are used when mentioning a graph, nodes and links are used to describe a network. $L(u)$ and $L(u, v)$ represent label of the vertex u and label of the edge (u, v) respectively. In an undirected graph, edges ("relations" or "friendships") are symmetric. For instance, the edge $(v, u) \in E$ is identical for edge (u, v) such that $E(u, v) = E(v, u)$. On the other hand, in a directed graph, vertices can go in one direction. This means that edges do not have to be symmetrical. Figure 3.1 shows examples of directed and undirected graphs.

A graph $G = (V', E')$ is isomorphic to graph $G = (V, E)$ if there is a bijective function $f : V' \rightarrow V$:

1. $(u, v) \in E' \Leftrightarrow (f(u), f(v)) \in E$
2. $\forall u \in V', L(u) = L(f(u))$
3. $\forall (u, v) \in E', L(u, v) = L(f(u), f(v))$

There is a mapping from vertex V' to V such that each edge in E' is mapped to an edge in E . A pair of vertices u, v is adjacent in V' if and only if the image pair $f(u), f(v)$ is adjacent in V . For every edge in E' , the label of edge (u, v) is the same as the label of

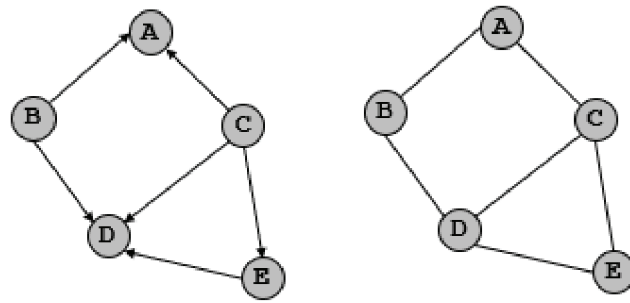


Figure 3.1. Toy examples from directed (left) and undirected (right) graphs. Twitter and Facebook platforms can be given as example for directed and undirected graphs respectively.

edge $(f(u), f(v))$. In other words, if two graphs are isomorphic, they must have the same number of vertices, edges, connected components and the same degrees for corresponding vertices.

Subgraph isomorphism is a generalization of graph isomorphism that checks whether subgraph S exactly matches graph G in terms of vertices, edges and labels [10]. A subgraph $S(V', E')$ of a graph G is a graph whose set of vertices and set of edges are all subsets of $G(V, E)$, $V' \subseteq V$ and $E' \subseteq E$, denoted by $S \subseteq G$. Figure 3.2 shows examples of graph and subgraph isomorphism.

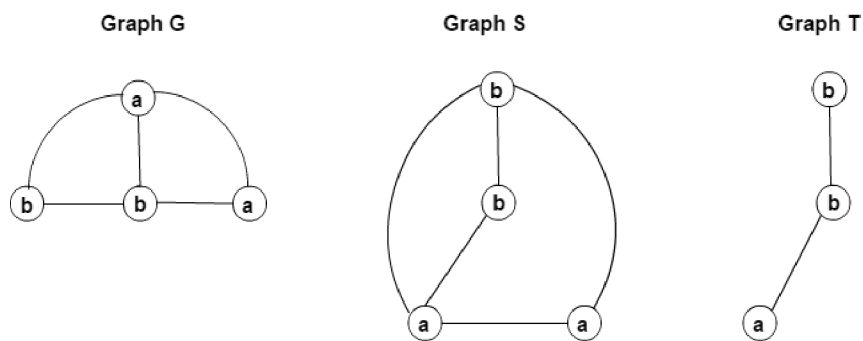


Figure 3.2. Graph G and S are isomorphic graphs because they include same vertices and edges with the same connections. Graph T is given an example for subgraph isomorphic for Graph G and S.

The primary processes of pattern mining are generation and frequency computa-

tion of patterns. When generating new candidates, the same patterns can be obtained. To check duplicate patterns, graph isomorphism should be controlled and to find the frequency of patterns, subgraph isomorphism should be checked [36]. As we see, *isomorphism checking* is required in graph pattern mining. However, it is considered one of the major problems, since it is known to be *NP – Complete* [2].

3.1.1. Candidate Generation

Before examining support computation of patterns, candidate patterns should be determined. Candidate generation has two approaches: Apriori-based and Pattern-growth.

3.1.1.1. Apriori-based Approach

The general approach is to begin with small-sized subgraphs and explore new, larger subgraphs by joining them. To generate new candidates, frequent subgraphs of the same sizes are joined. Next, the frequency of new candidates is checked. The Apriori-based approach provides downward closure property (also called anti-monotonicity) [13] which claims that if a subgraph is infrequent, all of its supersets are infrequent. This property allows for the pruning of redundant candidate generation. Apriori-based algorithms use Breadth First Search (BFS), and candidates are generated level by level. It means that all k-sized subgraphs are generated first, and then all k+1 sized subgraphs are constructed. This process is explained in Figure 3.3.

3.1.1.2. Pattern-Growth Approach

In the pattern-growth approach, new patterns are constructed directly by adding a new edge or a new vertex to a frequent subgraph. This approach uses Depth First Search (DFS) strategy. Hence, edges or vertices are added recursively. Figure 3.4 provides a small example of pattern-growth approach.

Although this approach does not include a join operation like Apriori does, it still has some problems with extension. Namely, the same graphs can be detected many times

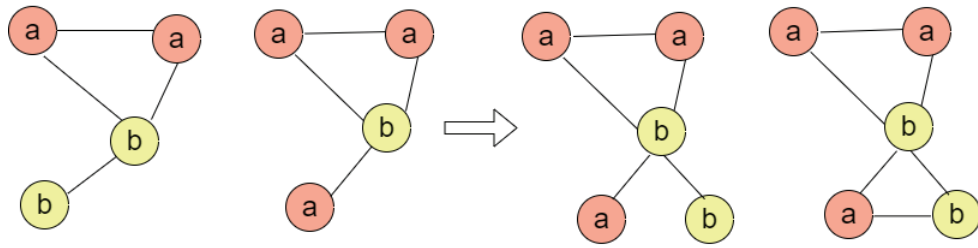


Figure 3.3. Two k -size similar graphs join to generate candidate $k+1$ size graph. In this example, k represents vertex number therefore this is a vertex growing example.

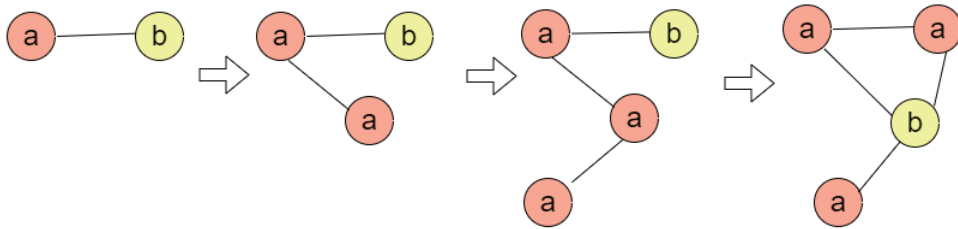


Figure 3.4. Pattern-growth approach starts with small size graph. In each iteration an edge is added and graph is extended.

in an edge extension. Existing approaches try to address this reproduction. For example, Graph-based Substructure Pattern Mining (gSpan) algorithm [35] uses right-most path extension technique to prevent this problem. Briefly, it traverses a graph with DFS and constructs a DFS Tree. But one graph may have many DFS Trees, and only one of them should be kept for more extensions. Therefore, the algorithm uses DFS lexicographic order and creates DFS codes of a graph. If a DFS code is not minimum it is pruned. These processes will be explained in Section 3.3.1.

3.1.2. Support Computation

After the new candidates are generated, their support values should be computed. The support value of a subgraph has a different meaning depending on the input graph type. Therefore, support calculations can change based on input graphs.

3.1.2.1. Support Computation in a Set of Graphs

Set of graphs means a graph dataset consists of more than one graph, and generally, these graphs are small. To find the support value of a subgraph, it is controlled whether this subgraph exists in a graph. If it exists in a graph, the count is increased, and the next graph is examined. If the support value is greater than or equal to the threshold, the candidate subgraph is considered frequent. Figure 3.5 shows an example of support computation for a set of graphs. Graph dataset has 3 input graphs and given threshold is 3. Each frequent pattern which is found from input set occurs in all input graphs.

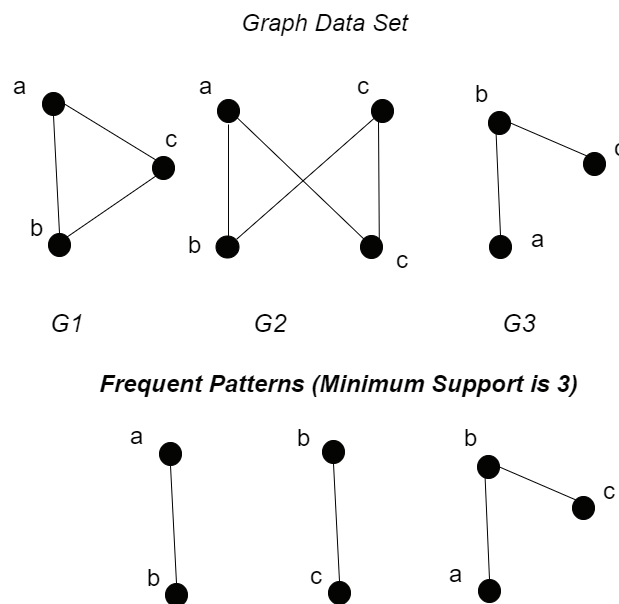


Figure 3.5. Graph dataset consists of three graphs. It is assumed that the minimum support is 3 and frequent patterns are given.

3.1.2.2. Support Computation in a Single Graph

All of the above-mentioned pattern mining algorithms take input as a set of graphs, but in some cases, input can be a single large graph such as an SN. As mentioned earlier, finding the support value of a subgraph in a set of graphs requires simply knowing the number of graphs that include the subgraph. On the other hand, finding the support computation in a single graph is much more difficult. For a subgraph to be frequent, the number of all embeddings in a single graph should exceed a predetermined threshold. When calculating the number of embeddings, overlaps of a subgraph can occur [7]. This can violate the downward closure property, which is satisfied in support computation in a set of graphs [9]. However, the downward closure property provides pruning and decreases search space. Figure 3.6 shows how support values of small and larger subgraphs are computed and how this violates the property.

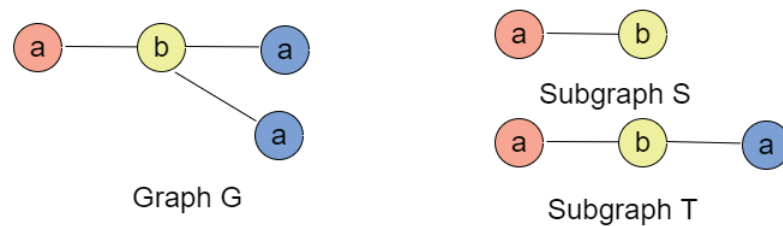


Figure 3.6. Graph G is main graph, S and T are subgraphs. T is larger than S. Occurrence of S in main graph G is 1 but occurrence of T is 2. This situation does not provide the downward closure property.

To preserve this property, three techniques have been proposed [9, 11], which can be categorized in two sections: These methods are summarized in two sections.

1. Overlap-based methods

- Maximal Independent Set (MIS)

This method first finds overlaps and then it counts the number of maximum independent non-overlapping subgraphs. According to MIS, all overlaps are harmful.

- Harmful Overlap (HO)

This technique allows overlapping if it does not violate anti-monotonicity property.

2. Non-Overlap-based methods

- Minimum Image Based (MNI)

The most appropriate method for support computation is minimum image based (MNI) support. Because MIS and HO are known as NP-Complete [7, 11, 10]. MNI checks the number of unique vertices that map to vertices in the graph for each vertex of a subgraph. Then, it takes the minimum number as the support value. Figure 3.7 shows how this method is implemented on a graph.

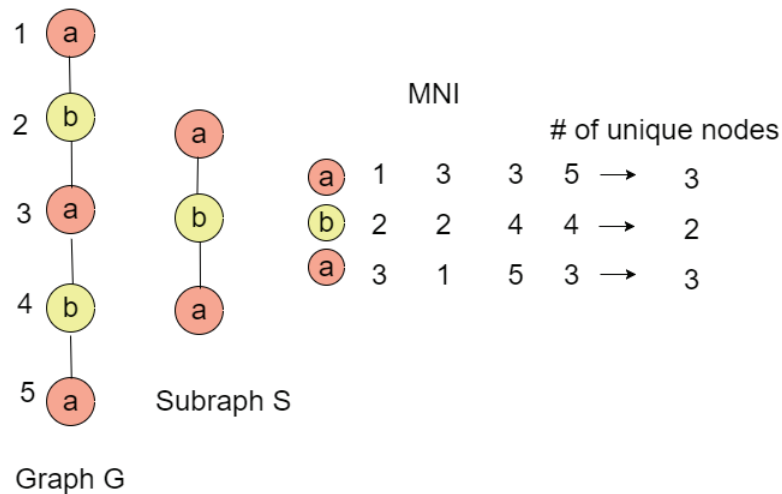


Figure 3.7. To find occurrences of graph S in graph G, MNI technique finds all mappings of nodes in G. Minimum unique number specifies the support value or occurrence value.

3.2. Related Work

There are many algorithms that use Apriori-based approach. The first introduced and well-known algorithms are Apriori-based Graph Mining (AGM) and Frequent Sub-

graph Discovery (FSG) [34] which were proposed in 2000 and 2001 respectively. In AGM [18], candidates are grown by adding a new vertex. In FSG [21], candidates are generated by adding a new edge. Since joining two subgraphs causes overhead, some algorithms have been introduced that use the pattern-growth approach instead.

Graph-based Substructure Pattern Mining (gSpan) algorithm [35], Molecular Fragment Miner (MoFa) and Spanning tree based maximal graph mining (SPIN) are mostly known algorithms that use pattern-growth approach.

When we examine algorithms in terms of input types, two categories are separated. Aforementioned algorithms are generally used as graph pattern mining and take a set of small graphs as input. However, other algorithms that take single graph as input have been proposed as shown in Table 3.1. The SiGram [23] and Grew [22] algorithms use MNI for support computation. Grami algorithm uses MNI [10] benefits from optimizations and can find approximate and exact patterns, and the AGrap algorithm [12] uses MNI and finds inexact patterns. AGrap algorithm utilizes similarity measure to control inexactness, but this algorithm is unable to obtain patterns from dense graphs [9].

Table 3.1. Frequent pattern mining algorithms that takes single large graph with different support counting.

Algorithm	Input Type	Candidate Generation	Support Comp.
SiGram (2005)	Undirected Graph	Apriori-based	MIS
Grew (2004)	Undirected Graph	Apriori-based	MIS
Grami (2014)	Directed or Undirected	Pattern-growth	MNI
AGrap (2014)	Directed or Undirected	Pattern-growth	MNI

The above algorithms use a single large graph, but they assume that nodes have only one label (except for the Grami algorithm). Since the aim of this thesis is to obtain common features from Facebook dataset whose vertices have more than one label, we evaluated previous studies. Although we work on a single large graph, we try to examine how to get patterns using two different kinds of algorithms. Therefore, first, we select gSpan algorithm from transactional graph settings because it is the basis of other graph mining algorithms. Moreover, it is the most well-known and studied algorithm in graph mining. Second, Grami algorithm is selected from single graphs because authors specify that it supports multi-labeled graphs and it takes single large graph as input.

3.3. Algorithms

In this section, gSpan and Grami algorithms are explained with small examples to understand clearly.

3.3.1. gSpan Algorithm

gSpan algorithm uses a set of graphs whose nodes are single-labeled to mine subgraphs. However, in studied dataset, nodes have ids and attributes (profile information). In this case, we employed an intuitive way; we try to generate new graphs whose nodes have only one and the same feature. In Figure 3.8, graph nodes have 5 features totally: i_1, i_2, i_3, i_4, i_5 .

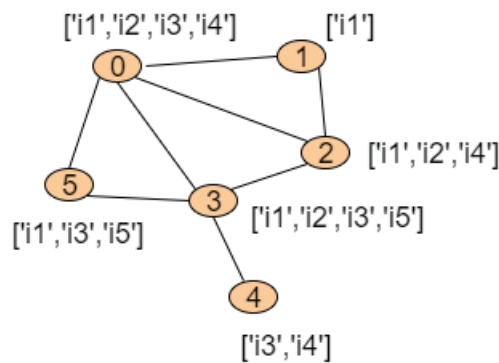


Figure 3.8. Example of attributed graph data.

Main graph is separated into subgraphs and we obtain five subgraphs whose nodes have only one feature: i_1, i_2, i_3, i_4 and i_5 as we see in Figure 3.9. Since all nodes have the same label in a subgraph, we use node ids as their labels. Now we can see easily that a pattern which consists of nodes 0 and 2 has common features: i_1, i_2, i_4 because this pattern is seen in three generated subgraphs. As a result of the algorithm, we can see which nodes have common features.

To generate these subgraphs, we traverse main graph with Breadth First Search (BFS) starting from a random node and checking whether this node has a specific feature. Then if a feature does not exist in a node, this node is removed from graph otherwise it is

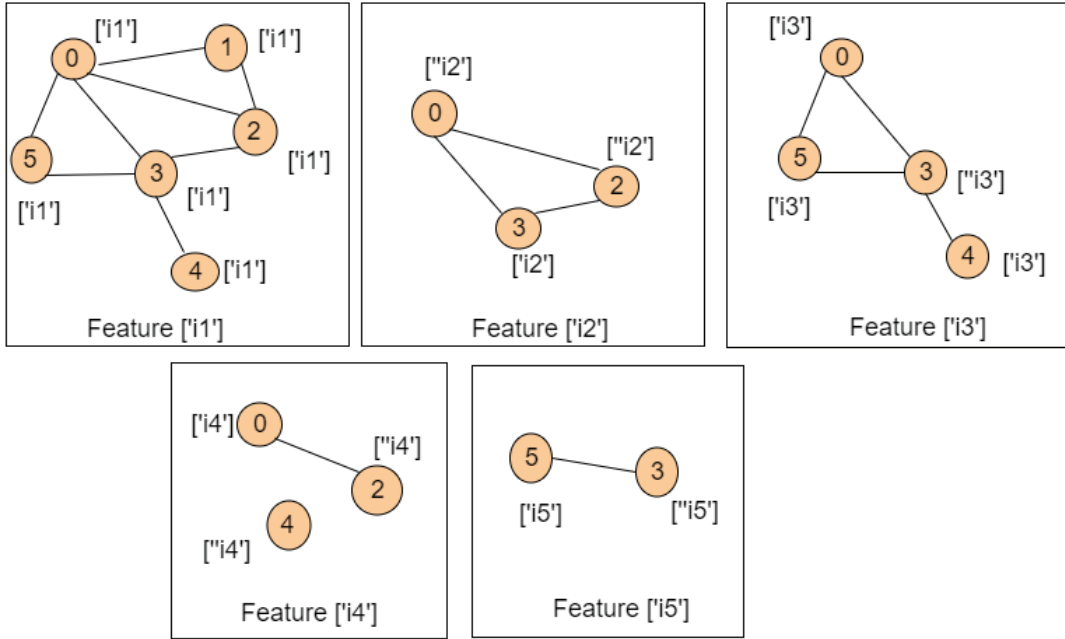


Figure 3.9. Separated subgraphs according to node features.

kept in graph and labeled with its own node id. This process is executed for all features. In this way, new graphs are generated from the dataset. This pre-process takes $O(E + V \cdot w)$ time where V is the number of vertices, E is the number of edges, and w refers to average attribute number of nodes. Using this labeling, gSpan algorithm can be applied, and this set of subgraphs is given as input and new frequent subgraphs are extracted.

gSpan algorithm is based on pattern-growth approach. That is, there is no candidate generation like in Apriori-based algorithms. It uses right-most path extension technique to construct new subgraphs. According to right-most path extension technique, new edges can be added following the right-most path. It constitutes the DFS Code of a subgraph. The same subgraphs can be generated because of isomorphic candidates. Therefore, we only need to keep one of them for extension. To avoid this duplication, among DFS Codes, the minimum code is selected which is called canonical. Minimum DFS Code has the smallest lexicographic order.

Example in the Figure 3.10 [36] which includes only two small graphs is given to understand the gSpan algorithm better. We assume minimum support is 2. Ids of vertices are 1,2,3,4 and labels are a and b.

Edges are represented as 5-tuple: $(v_i, v_j, L(v_i), L(v_j), L(v_i, v_j)) = (0, 1, a, a, -)$

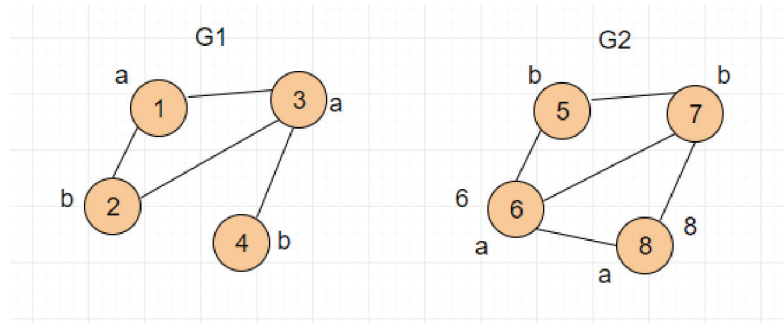


Figure 3.10. Two simple graphs for gSpan example.

where v_i , and v_j are discovery times of nodes, $L(v_i)$, $L(v_j)$, and $L(v_i, v_j)$ denote the label of v_i , label of v_j and label of edge between them respectively. First, all possible edges are found from these two graphs as we see in Figure 3.11. These are shown as candidates C_1, C_2, C_3 and C_4 .

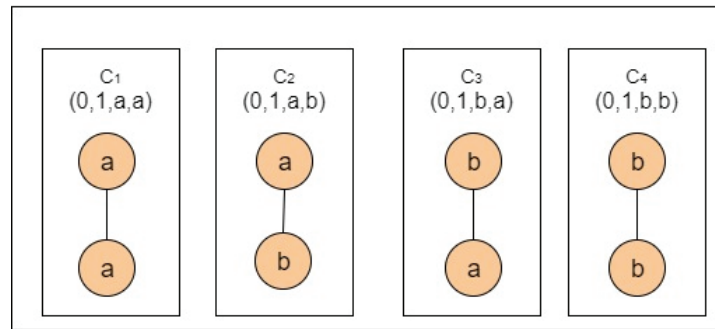


Figure 3.11. Possible frequent edges are generated from two graphs.

If any edge is not canonical or not frequent, it is eliminated. To find canonicity, it is checked whether a code is minimum or not. DFS Code of a pattern is a sequence of edges such as $(0,1,a,a)$. A pattern may have many DFS Codes. Extension from the same DFS Code which was seen in previous step is redundant because same patterns are found. Therefore, to eliminate redundant DFS codes, lexicographic ordering is used. For example DFS Code of C_2 and C_3 are $(0,1,a,b)$ and $(0,1,b,a)$ respectively. These are the same graphs and one of them should be excluded. So, the minimum one is kept extending and another one is removed. According to lexicographic order, DFS Code of C_2 is minimum which is called canonical so C_3 is eliminated. In this example, C_1 and

C_2 is canonical but C_3 is not canonical and C_4 is not frequent because this edge does not exist in two graphs. Therefore, C_3 and C_4 are excluded. Then, right-most path extension continues from C_1 because of DFS as shown in Figure 3.12. This process continues until no new candidate is found. The algorithm works in this process.

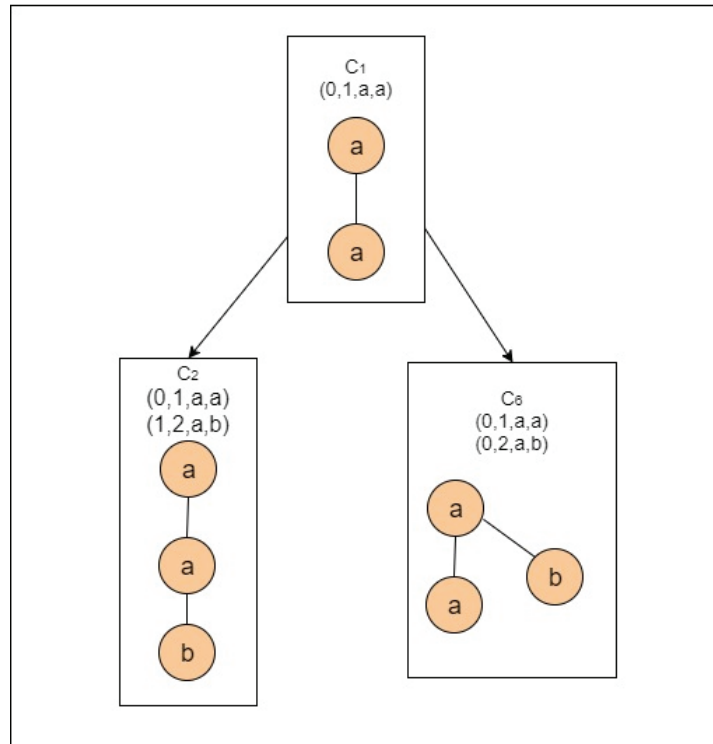


Figure 3.12. Right most path extensions from C_1 .

3.3.2. Grami Algorithm

In Section 3.3.1, massive input subgraphs can be constructed because of the number of nodes and features. Unlike most subgraph mining algorithms, GRAMI is applied for single, large and multi-labeled graphs. However, currently deployed version of Grami algorithm does not support multiple labels and it does not find frequencies of multiple labels accurately. Therefore, the algorithm is modified to work on multi-labeled graph. In addition to this, it is modified to find only common feature patterns.

First, Grami algorithm and how it works on single-labeled and multi-labeled dataset

is explained. Then how we modify it to obtain the correct results is shown.

3.3.2.1. Explanation of Grami Algorithm

Grami algorithm uses MNI support computation metric [10] to find frequencies of patterns. For example, in Figure 3.13 MNI calculation example is given. We want to find the frequency of pattern $P = B - A - C$ whose vertices v_1, v_2, v_3 correspond to variables x_1, x_2, x_3 in a variable set X . Support value of pattern P is obtained by looking for domain of each variables on graph G . Embedding ("instance" or "subgraph isomorphism") B_1, A_1, C_1 and B_1, A_1, C_2 can be found for domain of all variables. All images (number of unique nodes) are found for x_1, x_2, x_3 and minimum number of unique node images gives support value of pattern P which is equal to 1 for this example.

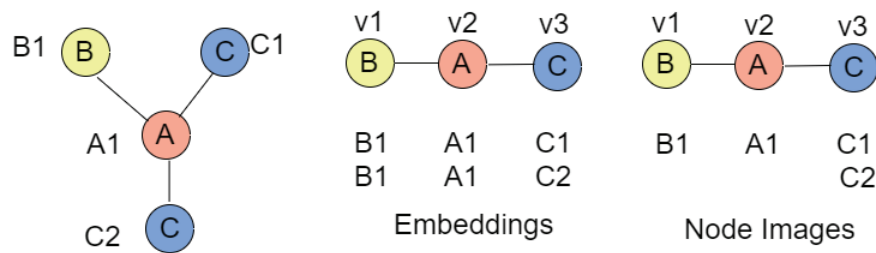


Figure 3.13. Node images.

Grami algorithm converts frequent subgraph problem to Constraint Satisfaction Problem (CSP). It aims to provide all constraints and shown as a tuple (X, D, C) : where X is a set of variables, D is a set of domains and C is a set of constraints between variables in X . The aim of CSP is to find a solution that provides all constraints. The problem can be mapped to a CSP as follows:

Assume $S(V_S, E_S, L_S)$ is a subgraph of G ,

1. There is a variable $x_v \in X$ for each vertex in subgraph S .
2. There is a domain set D for each variable $x_v \in X$.
3. C contains constraints as follows:

- All variables x_v, x_j in X is distinct, $x_v \neq x_j$.
- For each variable x_v in X , label of vertex v and label of variable x_v are the same.
- For all variables that has an edge in graph, labels of vertices and variables are the same.

In the preparation phase, vertices are grouped according to their labels, then whose labels are greater than threshold value marked as frequent. For each frequent vertex, reachable nodes are found within the given distance threshold δ which is equal to 1 for subgraph mining. Algorithm consists of two functions: Search and ISFrequent. Search function starts with all frequent edges. New candidate subgraphs are found by using right-most path extension. The algorithm uses the CSP (Constraint Satisfaction Problem) model to evaluate the frequencies of the candidate subgraphs in ISFrequent function. If the number of assignments in each domain is at least the threshold value, candidate subgraph is frequent. The algorithm is explained with single and multi-labeled graph examples in subsections.

3.3.2.2. Grami Algorithm using Single-label Graph

Figure 3.14 shows an example from a collaboration network that nodes represents author and edges refer to co-authorship. Research interests of authors are software engineering (SE), computer vision (CV) and data mining (DM). P is a pattern which wanted to find the frequency, Graph is G and threshold is selected as 3. Frequent labels are found using support values of SE, CV and DM which are 2, 5, and 3 respectively. SE is eliminated and frequent labels are CV and DM whose nodes are $U_1, U_4, U_5, U_9, U_{10}$ and U_2, U_6, U_7 respectively. For each node, reachable nodes are found. For example, node U_2 is reachable for node U_1 . Then, using frequent nodes, initial candidate edges are generated: CV-CV, CV-DM and DM-DM. Since all of them are frequent, new patterns are extended. Assume one of the extensions is pattern P (DM-CV-CV). To find the frequency of P , it is mapped to CSP.

Vertices in patterns consist of variables. Each variable x in X set corresponds to vertices v_1, v_2, v_3 in pattern. Each domain in D contains vertices in Graph G $u_1, u_2, u_3 \dots u_{10}$. Constraints include that all vertices and edges are different such that $v_1 \neq v_2 \neq v_3$ and

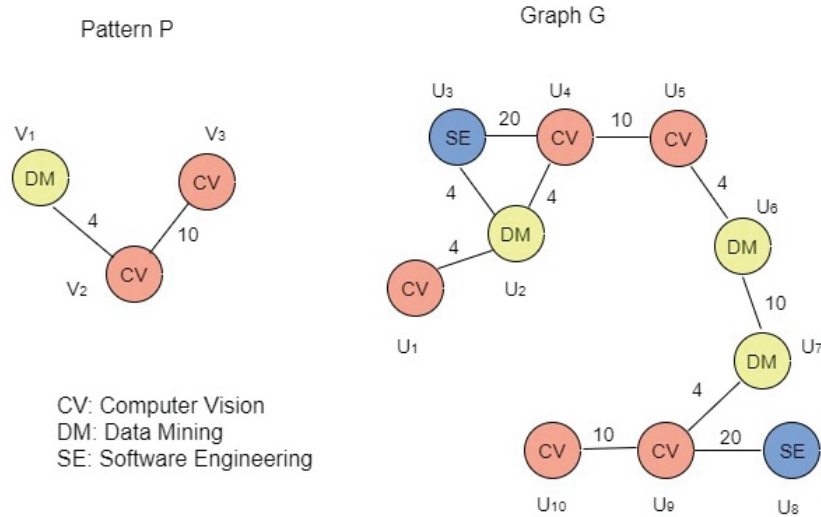


Figure 3.14. Grami single-labeled graph and pattern example.

$L(v_1) = DM$ $L(v_2) = L(v_3) = CV$. A solution ("subgraph" or "embedding") for this CSP problem can be $(v_1, v_2, v_3) = (u_2, u_4, u_5)$. The domains that are constructed for each variable x corresponding to a vertex in pattern P are as follows:

- For vertex v_1 , domain set is (u_2, u_6, u_7) .
- For vertex v_2 , domain set is (u_4, u_5, u_9) .
- For vertex v_3 , domain set is (u_5, u_4, u_{10}) .

Since minimum unique node number is 3 for all domains, according to MNI support value of pattern P is 3. Therefore, it is frequent. New candidate patterns are generated by extending frequent patterns until no new candidates are generated.

3.3.2.3. Grami Algorithm using Multi-label Graph

Grami algorithm works properly on single labeled dataset. This section shows how it works on multi-labeled graphs. Since original implementation does not give correct results, our contribution includes this part. An example is given in Figure 3.15.

Two of the four vertices in the graph are single labeled and two of them have more than one label and we assume that threshold is 2. According to Original Grami Algorithm,

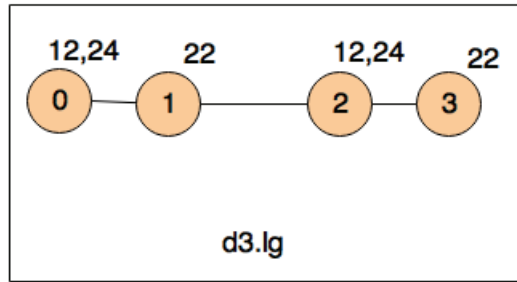


Figure 3.15. Graph includes nodes associated with multiple labels. Node ids are 0,1,2,3 and node labels are specified above the nodes.




all labels are extracted as 12, 22, 24, 100 (new label for 12 and 24) and their frequencies are found as 2. As we see, if there is a multi-labeled node in the graph, new label is generated starting at 100 which corresponds to the combination of labels such as label 100 is generated for combinations of labels 12 and 24. Then, reachable nodes of vertices that have frequent label are found:

- for node 0 which has [12, 24] labels, reachable node is (1:[22]) means that node 1 with label 22.
- for node 2 (2:[12, 24]), reachable nodes are (1:[22]) and (3:[22]).
- for node 1 (1:[22]) reachable nodes are (0:[24, 12]) and (2:[24, 12]).
- for node 3 (3:[22]), reachable node is (2:[24, 12]).

For nodes 1 and 3, label 100 is not added to reachable node. Therefore, initial edges (0,1,100,22), (0,1,22,24), (0,1,22,12) and their support values are constructed as 0, 2, 2 respectively as shown in Table 3.2. However, support value of edge 100-22 is not computed correctly because label 100 does not exist in reachable node sets. This is the first part that we modified. We added multiple label ids to reachable node sets if it requires. After this modification, frequency of all initial edges becomes 2 and algorithm extend these edges. For example, extension 100-22-24 is generated from initial edge 100-22. As Figure 3.16, extension 100-22-24 is given and we expect that the support value of this pattern should be 1 because minimum number of unique nodes is 1.

However, when a vertex has multi attributes, original algorithm assumes that there are different vertices for each attribute. So, it does not find support values correctly. This

Table 3.2. Support values of initial edges in Graph G.

Initial edges	Support value
	0
	2
	2

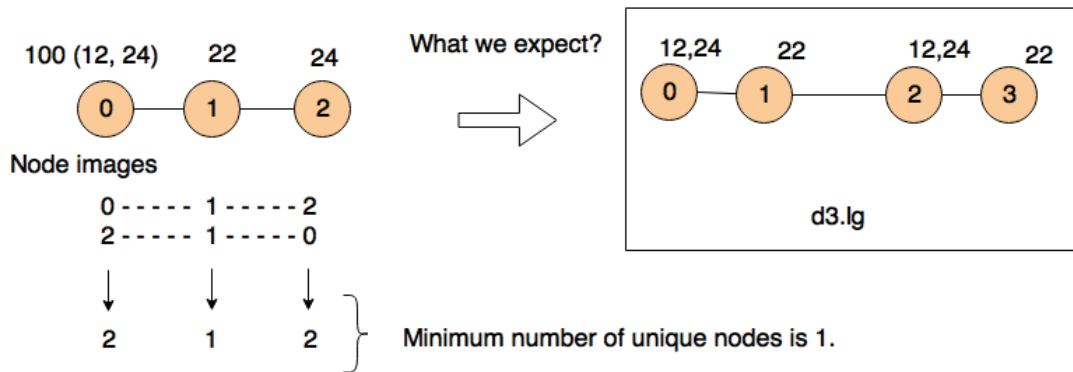


Figure 3.16. Grami support computation of a multi-labeled pattern

is the second part that we modified. To solve this problem, we use hash table to store domain sets of variables. We store label as key of the table and domain set as value such as for extension 100-22-24, keys are 100, 22 and 24, values are $\{0, 2\}$, $\{1, 3\}$, $\{0, 2\}$ respectively. Then we check whether frequency of a value is greater than 0 or not. If it is greater than 0, there are the same nodes in this pattern. For example, $\{0, 2\}$ occurs two times and it means that, the same nodes are given as different nodes. So, we can eliminate this pattern and continue from other patterns. Finally, after this modification, we obtain the correct results. For graph in the Figure 3.15, we obtain patterns (100-22), (22-24),(22-12). After that, we add a pruning phase to find only common features.

3.4. Experimental Work

3.4.1. Complexity Analysis of gSpan Algorithm

Since subgraph isomorphism is NP-Complete, at worst case, run time of gSpan algorithm should be exponential. However, it can be measured by taking the number of subgraph isomorphism tests. $O(kFD)$ gives the number of isomorphism tests to obtain frequent subgraphs from discovered frequent subgraphs where F is the number of frequent subgraphs, D is the dataset size which means how many graphs exist in this dataset and k is the maximum number of subgraph isomorphism that occur between a frequent subgraph and a graph [35]. If we assume that frequent subgraph has n number of nodes and graph has N number of nodes, k takes $O(N^n)$ times. If dataset and number of frequent subgraphs are small then number of subgraph isomorphism become small. It decreases the execution time.

3.4.2. Complexity Analysis of Grami Algorithm

Grami algorithm consists of two functions which are given in Appendix A. It computes all subgraphs and takes a subgraph as input and extend it with frequent edges. it requires $O(2^N)$ times where N shows the number of nodes in graph G . Then it finds the frequency of subgraphs which require subgraph isomorphism that takes $O(N^n)$ times where n is the number of a subgraph. Because each node in a subgraph should be checked that it exactly the same node in a Graph G . Total computational complexity is $O(2^N . N^n)$ which is exponential. Therefore, Grami algorithm uses some optimizations to improve performance [10].

If we calculate the performance of this algorithm on our computer, we should take the number of operations performed into consideration. Our computer performs $(2.6 \times 10^9) \times 8$ operations per second because it is influenced by Clock frequency (GHz) which is equal to 2.6 GHz with 8 CPUs. Total operation of algorithm with real data is calculated as $O(2^{60} \times 60^1) = 6.9712755e+35$ because N is equal to the number of nodes of the smallest ego network and n number of subgraph that we assume it is equal to 1. Then we divide

these two numbers to find the time which requires. $(6.9712755e + 35)/((2.6 \times 10^9) \times 8)$ gives approximately 38.492 days.

Since graph mining algorithms are not scalable for the large graphs [20], to evaluate them, we use two smallest ego networks 3980 and 698 from Facebook dataset which is explained in Section 4. Also, we use 8 features [14, 22, 53, 54, 55, 77, 78, 127] for evaluation since i) the algorithm does not scalable for large feature set and ii) these features are mostly seen in networks that we will see in Section 5.

3.4.3. gSpan Algorithm Results from ego_network_3980

To evaluate gSpan algorithm, ego network is traversed and 8 different subgraphs are generated according to feature set as explained in Section 3.3.1. These graphs consist of input dataset for gSpan algorithm. According to Table 3.3, in the feature set, a feature shows a subgraph whose nodes have only this feature. Therefore, if a feature set has 4 features, this input dataset includes 4 graphs. Support value is specified according to feature set size. If support value is equal to feature set size, the pattern found indicates that all nodes in this pattern have common features which is feature set.

Table 3.3. gSpan algorithm results on the smallest ego network 3980.

Feature Set	Support	# of Patterns found
[14, 22, 53, 54, 55, 78, 127]	7	807
[22, 53, 54, 55, 78, 127]	6	25756
[14, 22, 53, 54, 55, 77, 127]	7	0
[14, 22, 53, 54, 55, 127]	6	807
[22, 53, 54, 55, 77, 127]	6	0
[22, 53, 54, 55, 127]	5	25756

Number of patterns found shows that how many subgraph patterns are generated from gSpan algorithm. Patterns found from the subgraphs are like (0, 1, '3980', '4038', ...) means that in this ego network nodes with id 3980 and id 4038 have common features and these nodes are connected. We observe from the Table 3.3, gender information (77, 78) makes a difference among results. It shows that ego node has gender 78 and most of his/her connections have gender 78 because when feature set includes gender 77, no

pattern is found.

3.4.4. gSpan Algorithm Results from ego_network_698

Table 3.4 shows that nodes in the ego_network_698 do not have features in the feature set as common attributes. Because the number of patterns found is 0 or 1.

Table 3.4. gSpan algorithm results on the smallest ego network 698.

Feature Set	Support	# of Patterns found
[14, 22, 53, 54, 55, 78, 127]	7	0
[22, 53, 54, 55, 78, 127]	6	1
[14, 22, 53, 54, 55, 77, 127]	7	0
[14, 22, 53, 54, 55, 127]	6	0
[22, 53, 54, 55, 77, 127]	6	0
[22, 53, 54, 55, 127]	5	1

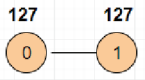
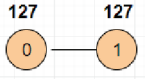
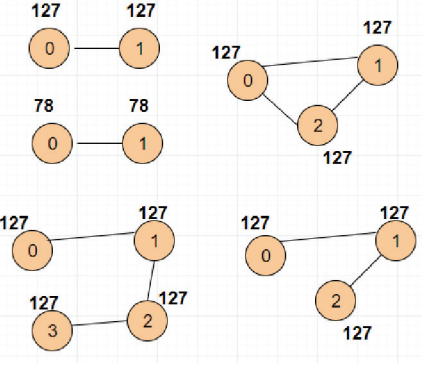
To obtain patterns, all combinations of features should be evaluated. For example 26 features require $C(26,26) + C(26,25) + \dots + C(26,2)$ combinations and it means high complexity.

3.4.5. Grami Algorithm Results from ego_network_3980

gSpan algorithm finds frequent subgraph whose nodes have common features and this subgraph includes their node ids because of our input type. However, Grami algorithm only finds frequent patterns. It means that it could not find node ids in a frequent pattern. Also in gram algorithm, dataset is taken as a single graph therefore, support calculation is different from gSpan algorithm. As the result is shown in Table 3.5, we give different support values to get patterns. At supports 51 and 47, a pattern whose nodes have a feature (127) is found. It means that this pattern is seen greater or equal to threshold times in this graph. But algorithm does not give the exact support numbers. If support value is equal to 43, 5 patterns are generated and visual results are seen in table. If support value is smaller than or equal to 40, algorithm takes long time and it does not give any

result because feature numbers of each node increase the number of possible candidate graphs.

Table 3.5. Grami algorithm results on the smallest dataset 3980.

Support	# of Patterns	Patterns	Visual Results
55	-	-	no pattern found
51	1	127-127	
47	1	127-127	
43	5	127-127-127 78-78	
40	-	-	-

3.5. Results

In this section, the study of graph mining algorithms on an attributed graph which takes two different input sets has been examined. The graph dataset for the gSpan algorithm is separated for each feature. If all features are taken into consideration, the algorithm will fail because of feature combinations. gSpan algorithm is suitable for small datasets.

Grami algorithm works on graphs with a single label. However, it is exponential in the worst case. Increasing the number of labels causes overhead because it increments the label combination.

CHAPTER 4

TRANSFORMATION OF GRAPH DATA TO TRANSACTIONAL DATA

Given subgraph mining algorithms require computationally expensive operations due to i) having lots of features for one node, ii) searching for subgraph isomorphism or iii) constructing many graphs for input. From a different point of view, graphs can be represented in a different format to reduce these problems.

4.1. Background and Related Work

Graphs can have lots of information about nodes and edges. To obtain precious information from graph data in an efficient way, graph transformation can be used. In a study [19], frequent itemset mining is used for learning from graph data. Authors convert graph data to itemset data because of the graph mining computational complexity. After converting itemset data they can use itemset mining algorithms. Edge lists represent graph edge information and $L = (\lambda(V_i), \lambda(V_j), \mu(e_k))$ is an element of edge list which has two node labels with an edge label between them. Edge lists allow to represent graph edge information as a list but this list can include more than one node that has the same label. However, an itemset does not contain the same item. Therefore, each element in the edge list is represented as a tuple (W, i) , W is an element and i is the number of occurrence of W in L . This list does not contain elements which have the same labels and this is called edge set. At this point, different frequent pattern mining algorithms can be applied. Using these methods, they showed that itemset mining algorithms are computationally more straightforward than graph mining algorithms. However, they only consider simple labeled graph not attributed graph and it is not suitable for a single graph input.

Frequent subgraph mining looks for the same structure means that there are the same node labels with the same edges between subgraphs. Cule et al. [8] have not look for exactly the same connections between nodes and edges. They give the definition of interesting itemsets in graph datasets. If nodes with the same labels frequently occur

near each other in a graph dataset, these labels called interesting. It is not necessary to make exactly the same connections for becoming interesting. To mine interesting itemsets authors proposed two methods: traditional itemset mining and cohesive itemset mining. In first approach, graph dataset is transformed into itemsets. Labels of neighbors for each node is written as itemsets. These itemsets consist of transactions. After this step, one of the existing frequent itemset mining algorithms, FP-Growth, is applied. In second approach, to find interestingness of an itemset X , cohesion and coverage values of X should be calculated. Number of nodes has an item from itemset X $|N(X)|$ is divided by number of total nodes $|G|$ and coverage is calculated as $P(X) = \frac{|N(X)|}{|G|}$. To find cohesion of itemset X , we look total distance (\overline{W}) of a node that has an item from itemset X to neighbors which has other items in X . $C(X) = \frac{|X|-1}{\overline{W(X)}}$ gives the cohesion of an itemset X where $|X|$ is itemset size. Finally, interest of X is calculated as product of its coverage and cohesion $I(X) = P(X)C(X)$. Although, first approach, traditional itemset mining, is not available for multiple graph setting, cohesive mining is applied for single and multiple graph datasets. Experiments show that frequent itemset mining is better than cohesive mining in terms of time efficiency but they find same itemsets for single graph. In multiple graph setting cohesive algorithm is compared with GASTON algorithm because for multiple graph setting frequent itemset mining cannot be applied. While GASTON algorithm discovers subgraphs, Cohesive algorithm extracts frequent itemsets. Although this method finds patterns which are not discovered by subgraph mining, there are some drawbacks. An interesting itemset can include items that infrequently occur in the dataset. In study [17], frequent cohesive itemsets are found using algorithm in the previous study by the same authors. Because in the previous study, interesting itemsets are found but each item in an interesting itemset does not controlled whether they are frequent or not. Therefore, it should be checked that each item in a discovered itemset must be frequent. To find frequent cohesive itemsets, firstly, infrequent items are excluded then candidate itemsets are generated using depth first search. Cohesions of itemset are found like in the previous study. It is more efficient than subgraph mining because costly isomorphism testing does not perform.

Apart from previous studies, some studies use more than one conversion. For example, Thomas et al. proposed Itemset-based Subgraph Mining (ISG) algorithm [33] to obtain subgraphs using frequent itemset mining algorithms. ISG can be summarized in five steps:

- Graph data are converted to itemset data.

- Maximal frequent itemsets are found.
- Each maximal itemset is converted to graph data.
- If these converted graphs are ambiguous then preprocessing is done.
- Itemsets are converted graphs and maximal subgraphs are found.

Some graphs have the same converted itemsets therefore they cannot be transformed to a graph uniquely and they cause ambiguity. To handle this ambiguity item ids of edges are added to this conversion. After all, using itemset mining, maximal subgraphs are found from graphs with unique edges. This algorithm is compared with SPIN, MARGIN and gSpan algorithms in terms of time-efficiency by this study. But they took into consideration unique edge labeled graphs. Our graph datasets do not have edge label because all edges are friendship relation and same.

Many of the above studies [19, 33, 17] input graph is considered as set of small graphs and some of them [19, 33] take graphs which has nodes only one labeled. The study [17] claims that they handle with multiple labels using transformation on the given graph. Each node is replaced with a dummy node which has a unique label. Then this node is connected to a new set of nodes. But the details and implementation are not shown in the paper.

4.2. Experiment

As we see from background information, graph transformation can be more efficient than graph mining algorithms. Therefore, this section explains how we represent graph data as transactional data. Features of sample nodes are given in Table 4.1. As we see from the table, some nodes have many features, while others have few. In addition, the same features are seen in many nodes in a network.

According to transformation algorithm which is given in Appendix A, edges of a network are considered as transactions. Therefore, every edge between nodes is written as transaction id, and common features of nodes represent items. For this reason, all edges are traversed using breadth first search and common features are getting from the end nodes of this edge. Common features are intersection of two node features and they constitute of transactions.

Table 4.1. Some nodes and their feature ids are shown in the table.

Node id	Features
857	[78, 127]
862	[14, 22, 53, 54, 55, 78, 127, 154, 216, 253, 296, 341, 720, 1065, 1191]
865	[53, 78, 127, 154, 341]
868	[22, 54, 78, 127, 341]
3990	[6, 53, 55, 77, 125, 1277]
4007	[6, 78, 125, 1275, 1276]
4016	[55, 78, 125, 1276]
4025	[77, 125, 1276]

After graph transformation we obtain edges in a graph and common features of these nodes as shown in Table 4.2. For itemset mining algorithms, only common features column will be used.

Table 4.2. This table shows common features of nodes. Edges are written as Tid and common features are items.

Edge	Tid	Common Features (items)
857-862	1	78,127
857-865	2	78,127
857-868	3	78,127
862-868	4	22, 54, 78, 127, 341
862-865	5	53, 78, 127, 154, 341
865-868	6	78, 127, 341
3990-4007	7	6,125
3990-4016	8	55, 125
3990-4025	9	77, 125
4016-4025	10	125, 1276
4007-4025	11	125, 1276
4007-4016	12	78, 125, 1276

4.3. Result

Properties of acquired transactional dataset are given in Table 4.3. According to table, given properties show that these two datasets are highly similar.

Table 4.3. After transformation, transactional dataset properties of two smallest ego networks

Ego Network	# of Transactions	# of items	# of Average Transaction Size
3980	204	40	3
698	289	44	2

Since every edge and node is traversed, and all features of two nodes are compared, edge number E , vertex number V and average attribute number w of each node become important for this pre-process. Time complexity of this pre-process is $O(E * w^2 + V)$.

CHAPTER 5

ITEMSET PATTERN MINING ON TRANSACTIONAL DATA

In Chapter 4, multi-attributed graph dataset is transformed into transactional data. Therefore in this chapter, itemset pattern mining algorithms are examined on this transformed data. First, background information about itemset mining is given and then selected algorithms are applied and the results are compared with regard to complexity, performance and the number of patterns which is found.

5.1. Background

Association rule problem which was proposed by Agrawal et al. [4] tries to understand buying patterns that increase supermarket sales. Transaction database T consists of transactions $T = \{T_1, T_2, T_3 \dots T_n\}$. Each transaction refers to a set of items bought by customers. If a set consists of l items, it is called l - *itemset*. The objective of association rule mining is to find all frequent itemsets and to generate useful rules from these itemsets which consist of large amount of data. For example, rule $X \rightarrow Y$ where X and Y are frequent itemsets denotes that if X is purchased, then Y is purchased [1]. Two metrics are used to evaluate the results, namely support and confidence [24].

The support of an itemset X means the number of transactions that include X in a dataset D . It is calculated as ratio of transactions including item to all transactions in the database $sup(X) = \frac{X_{count}}{|D|}$ and is used to find frequent patterns. If $sup(X)$ is greater than or equal to the minimum support value, it is said that this itemset is frequent. To obtain frequent itemsets candidates are generated then support values are computed.

Each generated itemset I is potentially a candidate frequent pattern. Therefore, the candidate itemset search space is exponential because there are $2^{|I|}$ potentially frequent itemsets. For example, if set of items $I = \{A, B, C, D, E\}$ and $2^{|I|} = 2^5 = 32 - 1$ (except empty set) possible itemsets are generated. However, many of these candidates may not be frequent. Apriori property (also called downward closure property) provides

the concept that a subset of a frequent itemset is also a frequent itemset. For instance, if itemset $\{AB\}$ is a frequent itemset then $\{A\}$ and $\{B\}$ should also be frequent. If $\{AB\}$ is not frequent, then any superset of it can not be frequent. Thanks to this property, search space is decreased.

Apriori and FP-Growth algorithms are well known itemset mining algorithms and each has different approaches to find frequent itemsets [5, 16].

5.2. Algorithms

After converting graph data to transactional data, itemset pattern mining algorithms can be applied. Selected itemset mining algorithms are Apriori, FP-Growth and Max-Miner algorithms. In this section algorithms are explained using the same example.

5.2.1. Apriori Algorithm

Apriori Algorithm [5] runs on our small dataset and steps are shown briefly as follows:

- Initially, support counts of k-itemsets are found, k is 1 at first step.
- k+1 candidate itemsets are generated from k itemsets.
- According to Apriori principle, some itemsets are pruned.
- Support counts of itemsets are found, and then some of the candidates are eliminated because of insufficient support.
- The procedure is repeated until the candidate set is empty.

As described in above, support counts of all 1-itemsets are calculated using transformed data in Table 4.2. Results are shown as "one-item itemset key and their support values such as; '78': 7, '127': 6, '125': 6, '341': 3, '1276': 3, '154': 1, '22': 1, '55': 1, '54': 1, '6': 1, '53': 1, '77': 1". Minimum support value ("threshold") is specified by user. For this example, minimum support value is determined as 3. That is, if an itemset is shown more than or equal to 3 transactions, this itemset is taken as candidate for the next step.

Items whose support value is smaller than minimum support are eliminated and remaining items are '78': 7, '127': 6, '125': 6, '341': 3, '1276': 3. Itemsets which have 2-items are created from remaining items. In this step, algorithm checks if Apriori property is required. It means that, all k items which is used to create k+ 1 items are checked whether they are frequent because according to apriori property a subset of a frequent itemset must also be a frequent. But this property does not occur for this example. That is, all subsets satisfy the Apriori property. All these phases are shown in Figure 5.1 step by step.

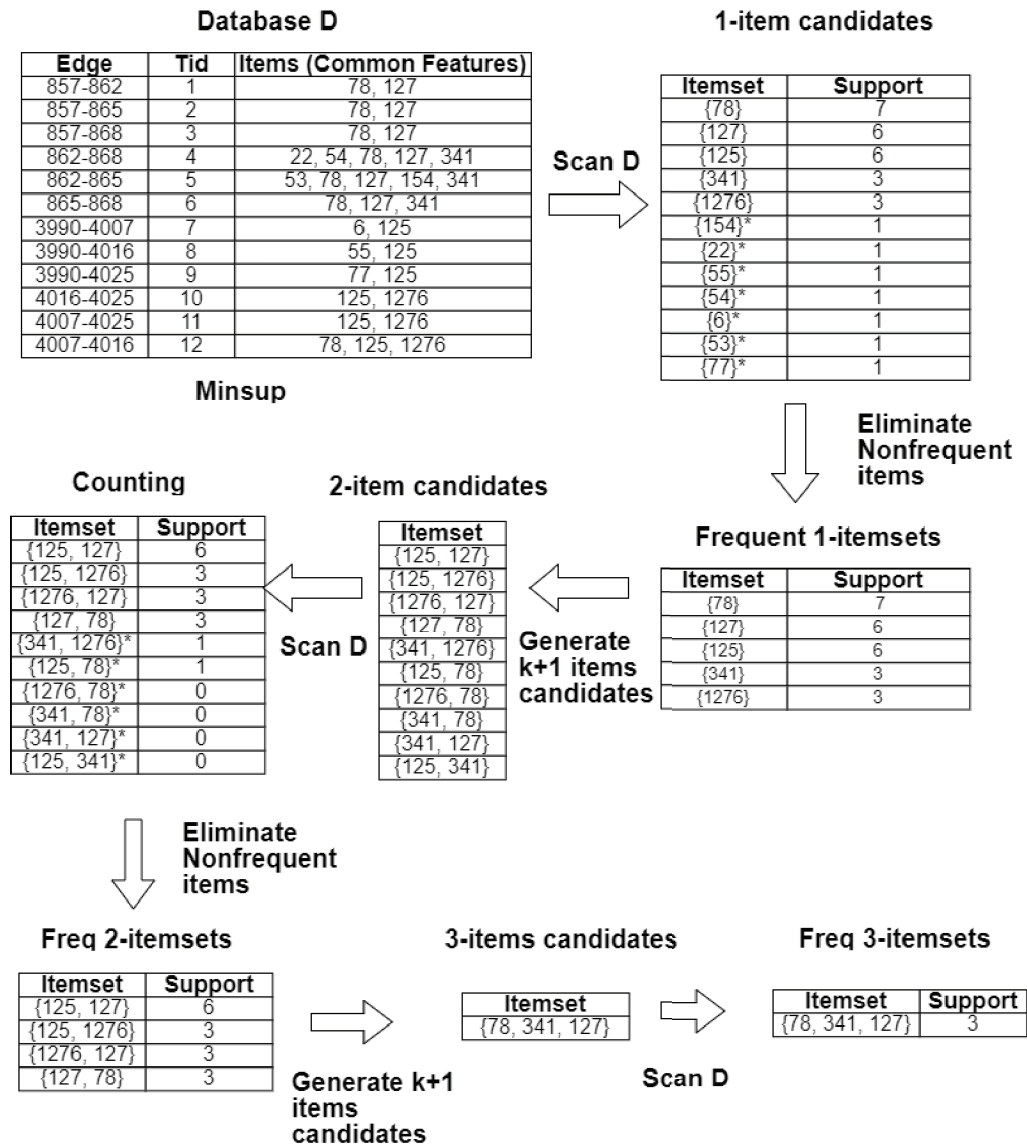


Figure 5.1. The Apriori Algorithm on sample dataset

As shown in the figure, from 1-item to 3-items frequent itemsets are found. Re-

sult that include all k-itemsets is [['127', '341', '78'], ['78', '127'], ['341', '78'], ['125', '1276'], ['341', '127'], ['127'], ['341'], ['125'], ['78'], ['1276']]. This means that, features 127, 341 and 78 are the most common patterns for this network in 3-itemsets. These numerical values have feature names like 1276 : *lastname*, 78 : *gender*, 341 : *education; school; id*, 125 : *locale*, 127 : *locale*. Database is scanned for every support counting phase as shown in Figure 5.1. In addition to this, Apriori Algorithm generate infrequent itemsets as candidates and all itemsets are checked.

5.2.2. FP-Growth Algorithm

Apriori algorithm is easy to understand and efficient when database is small. However, it generates lots of candidates and scans the database many times. On the other hand, FP-Growth algorithm [16] finds frequent itemsets without candidate generation. This algorithm has two approaches:

1. A data structure called FP-Tree is built.
2. Frequent itemsets are obtained from FP-Tree.

To construct FP-Tree, database is scanned once and frequent 1-itemsets are found. Then, Frequent itemsets are sorted in descending order called priority list. In transactions, infrequent items are removed and transactions are sorted according to the list. Root of FP-Tree is created and labeled as null. The database is scanned for the second time to add items to the tree. After constructing FP-Tree, for each item conditional FP-Trees are created directly on FP-Tree. After that conditional patterns are found. Same toy example explains how FP-Growth algorithm works in detail.

In Figure 5.2 database is scanned and frequent 1-itemsets are found. Items are sorted by support count and this created list is called priority list. According to this list, items in the transactions are sorted.

After that, transactions are scanned and all items in the transaction are added recursively to the FP-Tree as shown in Figure 5.3.

After constructing FP-Tree, frequent itemsets are found directly from this tree as shown in Figure 5.4. Items are extracted from bottom, for instance, item 1276 is extracted and items on this path are written as conditional FP-Tree items such as item 125 occurs

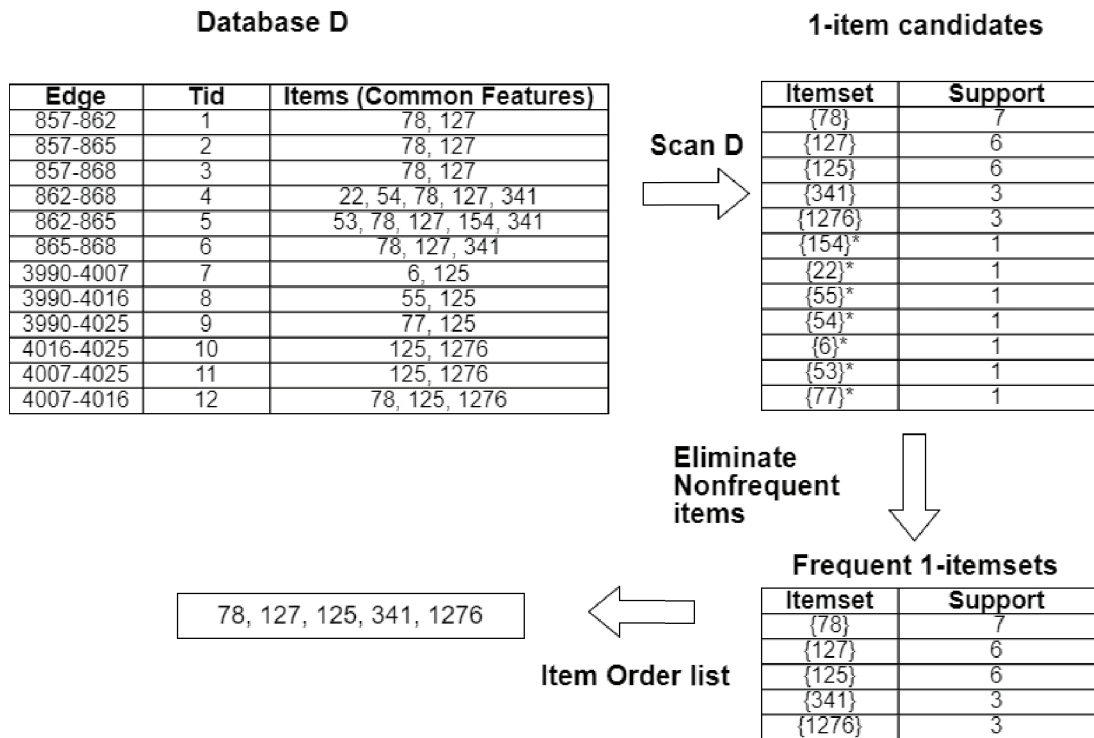


Figure 5.2. Frequent 1-itemsets are found and sorted

on this path two times, 78 and 125 are on this path one time. Using conditional FP-Tree items, according to minimum support some items are eliminated and others create frequent patterns. In previous example, support count of item 125 is three but support of item 78 is one. For this reason, item 78 is eliminated, then for item 1276, pattern 125, 1276 is created as shown in Figure 5.5.

To show final results 1-item frequent sets should be added to this table. Results are the same with Apriori Algorithm. (125, 1276) : 3, (341) : 3, (127, 78) : 6, (127) : 6, (125) : 6, (78) : 7, (341, 78) : 3, (127, 341) : 3, (127, 341, 78) : 3, (1276) : 3 are frequent itemsets for this example.

5.2.3. Max-Miner Algorithm

The aim of Max-Miner algorithm [6] is to find maximal patterns instead of finding of all frequent patterns. This algorithm uses both the superset-frequency and the subset-

Step 1 - FP-Tree Construction

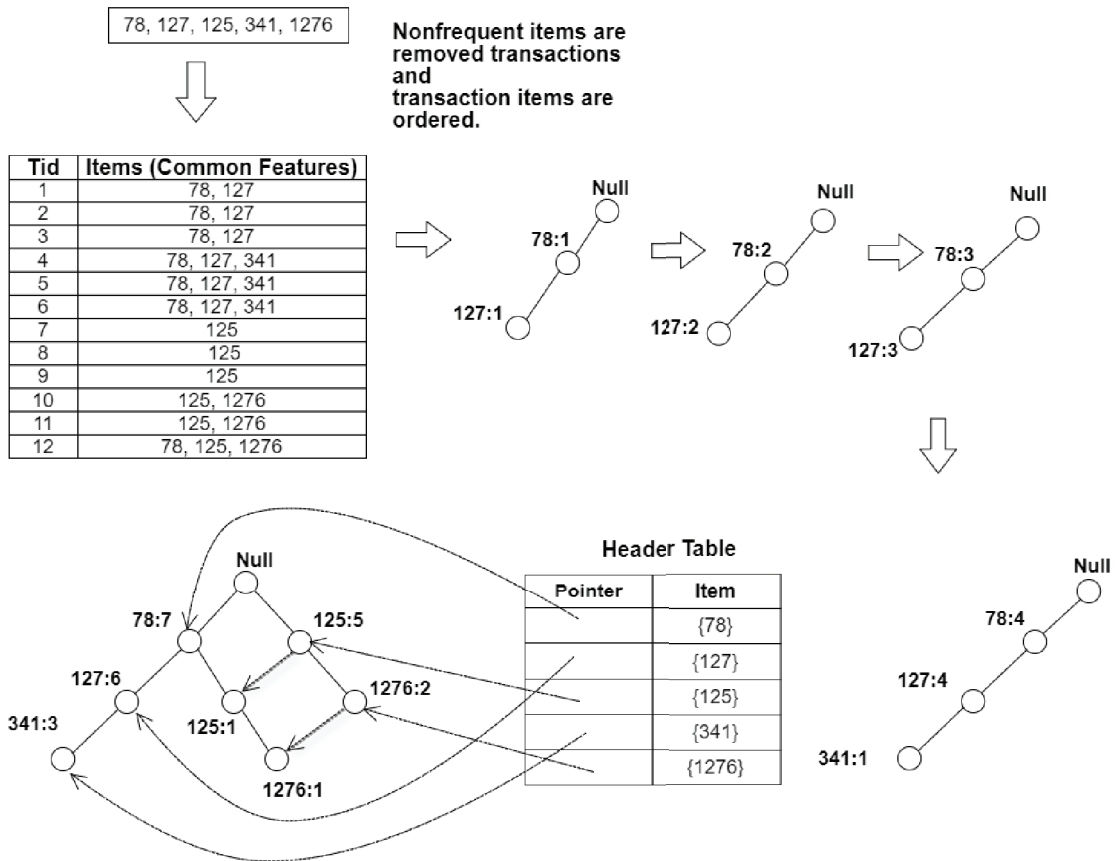


Figure 5.3. Step 1. FP-Tree Construction

infrequency for pruning.

Algorithm initially scans database and gets the $1 - itemsets$ which are initial candidate groups. After finding $1 - itemsets$, in main function, for every candidate, dataset is scanned and supports of all candidate groups are found. A candidate itemset group g is shown as head and tail groups. First part of the group g shows the head $h(g)$ and second part which means that all items not in $h(g)$ shows the tail $t(g)$. For example, for node a , $h(g) = a$, $t(g) = b, c, d$. According to the algorithm, for each g in candidates, find the support of $h(g) \cup t(g)$ and if it is frequent then any subset of this itemset be frequent but not maximal. Therefore, sub-nodes are not found and all of them are pruned. This operation is called as superset-frequency pruning. For each g in if $h(g) \cup t(g)$ is not frequent then new sub nodes are generated. if $h(g) \cup \{i\}$ is not frequent, item i is removed from $t(g)$. This is called subset infrequency. Example is shown in Figure 5.6

Step 2 - Frequent Itemset Generation

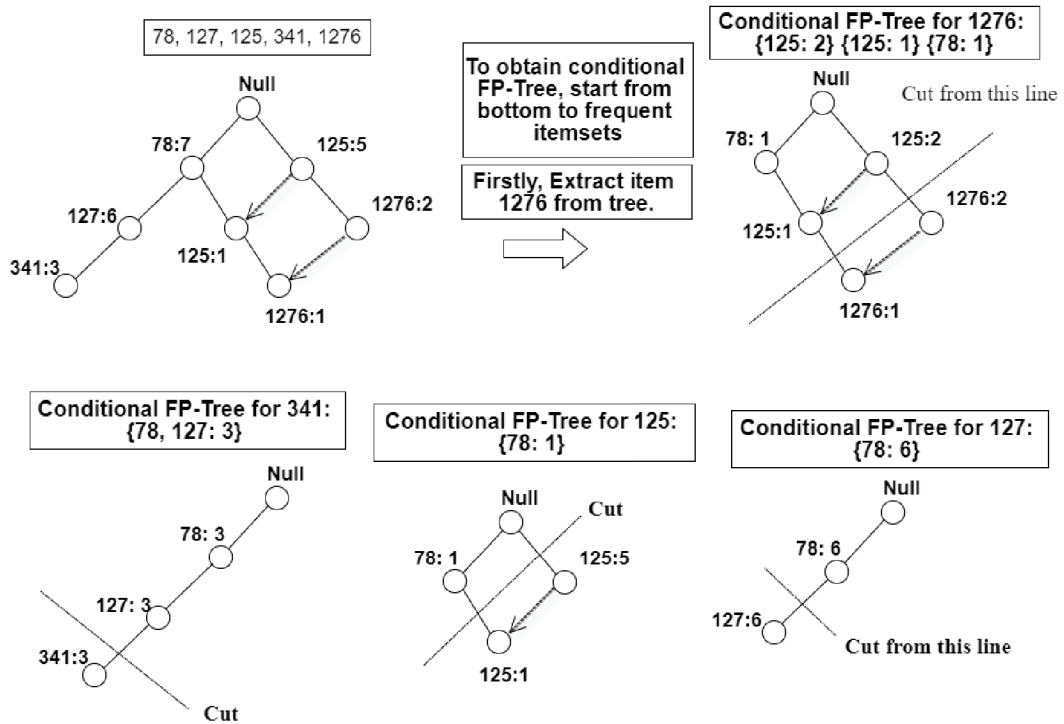


Figure 5.4. Step 2. Constructing Conditional FP-Trees to generate itemsets

with the same dataset. Initially, database is scanned and all frequent 1-itemsets are found. New candidates are generated using frequent items.

For example in this figure, new candidates are generated from (78, 127, 125, 341, 1276) such as 78(127, 125, 341, 1276) where $h(g) = 78$ and $t(g) = (127, 125, 341, 1276)$. For each candidate group check whether $h(g) \cup t(g)$ is frequent. if $h(g) \cup t(g)$ is frequent, then node g is not expanded and all subsets are pruned. if not, new candidates are generated such that $h(g) \cup \{i\}$ where $i \in t(g)$. If $h(g) \cup \{i\}$ is not frequent, then item i is removed from parenthesis. For example, (78, 125) and (78, 341) are not frequent then items 125 and 341 are removed from parenthesis. After whole processes, the result is [(127, 341, 78), (125, 1276)].

Suffix	Frequent itemsets
1276	{125, 1276}:3
341	{78, 341}:3, {127, 341}:3, {78, 127, 341}: 3
125	-
127	{78, 127}:6

Figure 5.5. Frequent Itemsets

5.3. Experimental Work

In this section, we give the results of itemset mining algorithms for transformed data. Itemset mining algorithms Apriori, FP-Growth and Max-Miner are implemented in Python using Networkx library. Like in graph approach, for the experiments, two smallest ego networks (3980 and 698) are selected. Minimum support threshold is given from 10% to 90%. For instance, threshold is 10% for the dataset means that if an itemset is frequent, this itemset is seen at least in $(67.557 \times 10) \div 100 = 6.755$ transactions. Frequent patterns found from itemset mining algorithms are given in Table 5.1 and Table 5.2 for ego network 3980 and 698 respectively.

As shown from these tables that Apriori and FP-Growth algorithms find the same frequent itemsets according to different thresholds. For instance, at 0.1 threshold in Table 5.1, number of patterns found from Apriori and FP-Growth algorithms is 53. As we see pattern '127' is the most commonly seen feature. Looking these result, we make an inference that nodes have some common attributes with their neighbors such as at threshold 0.1 we obtain ['54', '78', '53', '22', '127']. It means that this pattern has at least 6.755 relations among people. Also we know the meaning of these features such that "78" is a gender feature, ["53", "54"] education_type features, ["22"] refers to education_degree and ["127"] refers to locale information. In addition, itemset mining algorithms show the number of frequent itemsets. For example, pattern ["54", "78", "53", "22", "127"] is shown in 21 transactions and it means 21 relations between nodes have this feature pattern in the network.

Table 5.3 shows the execution time of the algorithms. If the threshold is low, number of produced patterns increases. However, algorithms have small execution times.

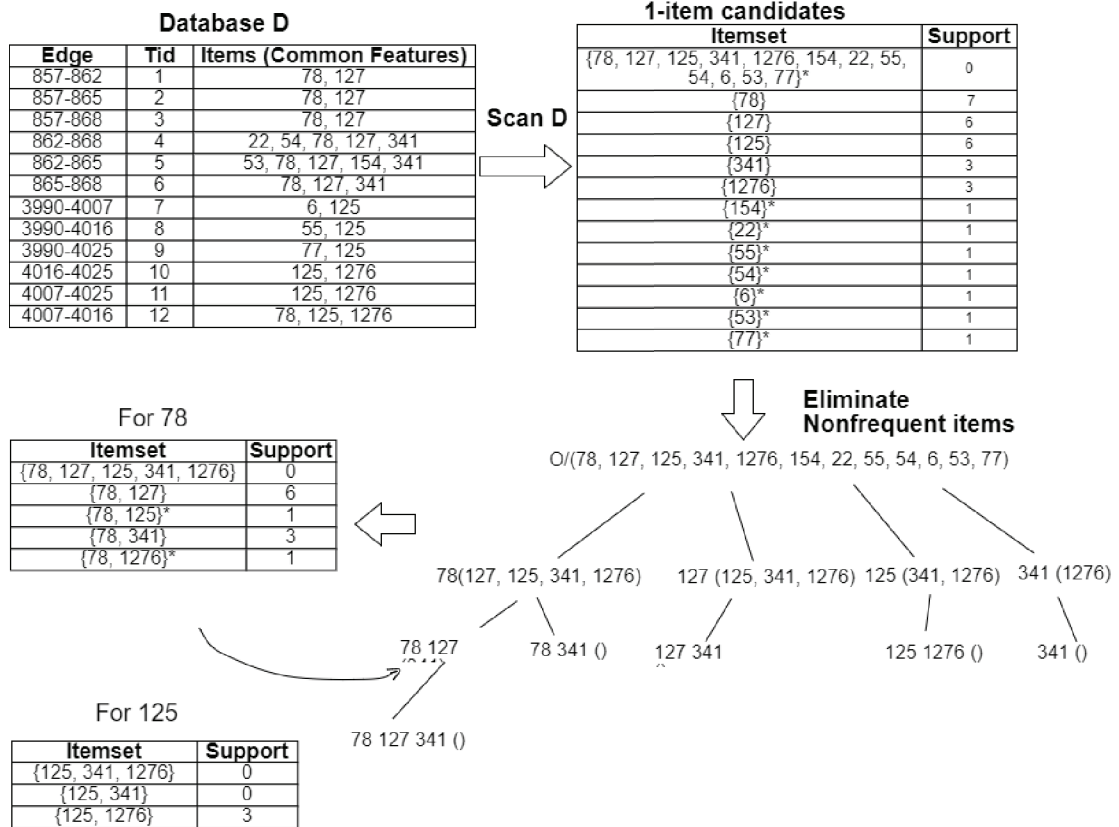


Figure 5.6. Working procedure of MaxMiner algorithm.

Performance of Apriori algorithm is worse than other algorithms since it requires many scanning from database.

5.3.1. Analysis of Itemset Mining Algorithms

This section provides analysis of Apriori, FP-Growth and Max-Miner algorithms. Algorithm pseudocodes are given in Appendix A.

5.3.1.1. Analysis of Apriori Algorithm

Algorithms takes transaction database and threshold as input and gives frequent itemsets as output. To find frequent $\{large\ 1 - itemsets\}$, $F_k = \{i | i \in I \wedge \sigma(i) \geq$

Table 5.1. Result of the smallest ego network (3980) according to itemset mining algorithms

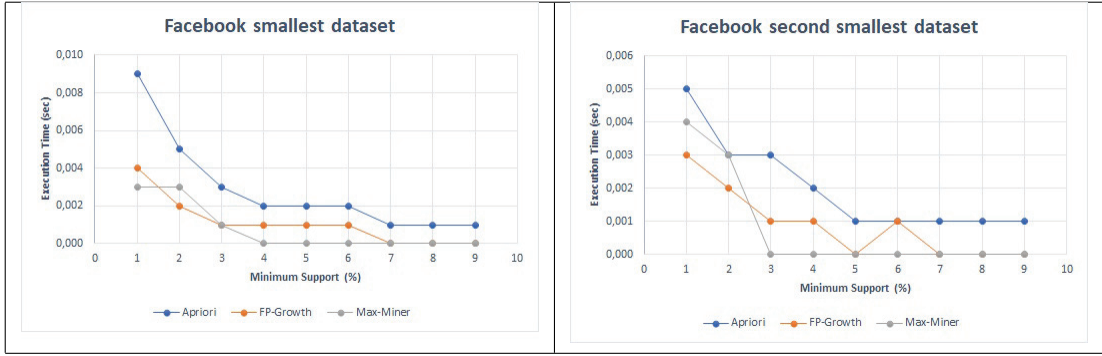
Threshold	Algorithms					
	Apriori		FP-Growth		Max-Miner	
	#	Patterns	#	Patterns	#	Patterns
0.1	53	['54', '78', '53', '22', '127'] ['55', '54', '78', '53', '127'] ['55', '54', '53', '127'] ['54', '53', '22', '127'], ['78'], ['127'], ['53'] ['55'], ['22'], ['14']	53	['54', '78', '53', '22', '127'] ['55', '54', '78', '53', '127'] ['55', '54', '53', '127'], ['54', '53', '22', '127'] ['78'], ['127'], ['53'] ['55'], ['22'], ['14']	4	['127', '53', '54', '55', '78'] ['127', '22', '53', '54', '78'] ['127', '14', '54'] ['127', '14', '53']
0.2	21	['54', '78', '53', '127'] ['127', '53', '78'], ['54', '53', '78'], ['54'], ['55'], ['78'], ['127'], ['53']	21	['54', '78', '53', '127'] ['127', '53', '78'], ['54', '53', '78'], ['54'], ['55'], ['78'], ['127'], ['53']	3	['127', '53', '54', '78'] ['127', '53', '55'] ['53', '54', '55']
0.3	9	['127', '78'], ['54', '53'], ['54', '78'], ['53', '127'], ['54', '127'], ['54'], ['78'], ['127'], ['53']	9	['127', '78'], ['54', '53'], ['54', '78'], ['53', '127'], ['54', '127'], ['54'], ['78'], ['127'], ['53']	5	['54', '78'], ['127', '78'], ['53', '54'], ['127', '53'], ['127', '54']
0.4	3	['127', '78'], ['78'], ['127']	3	['127', '78'], ['78'], ['127']	1	['127', '78']
0.5	3	['127', '78'], ['78'], ['127']	3	['127', '78'], ['78'], ['127']	1	['127', '78']
0.6	3	['127', '78'] ['78'], ['127']	3	['127', '78'], ['78'], ['127']	1	['127', '78']
0.7	2	['78'], ['127']	2	['78'], ['127']	1	['127']
0.8	1	['127']	1	['127']	0	-
0.9	0	-	0	-	0	-

$N \times minsup$ formula is used (step 2 in pseudocode). This algorithm is affected by some factors such as minsup threshold, number of items, and number of transactions etc. To obtain frequent 1-itemsets support count of every item in database is found (step 2). To find support counts, every transaction is traversed so assuming that total number of transactions is N and average transaction width is w , $O(Nw)$ time is required. Candidate generation is implemented using a function called *apriori_gen* (step 5). This function follows two operations: Candidate Generation and Candidate Pruning. Several methods are used to generate candidates. The most available method is $F_{k-1} \times F_{k-1}$ which propose if first $(k - 2)$ items are identical of two $(k - 2)$ itemsets, these itemsets can be merged. For instance, *Bread, Diaper* and *Bread, Milk* are merged to form a 3-itemset *Bread, Diaper, Milk*. At worst case scenario, algorithm must merge every frequent $(k - 1) - itemsets$ to find $k - itemsets$. Therefore, cost of merging itemsets is: cost of merging $< \sum_{k=2}^w (k - 2) |F_{k-1}|^2$

Table 5.2. Result of the second smallest ego network (698) according to itemset mining algorithms.

Threshold	Algorithms					
	Apriori		FP-Growth		Max-Miner	
	#	Patterns	#	Patterns	#	Patterns
0.1	10	['127','78'], ['55', '340'], ['55', '53'], ['78'], ['126'], ['77'], ['127'], ['53'], ['55'], ['340']	10	['127','78'], ['55', '340'], ['55', '53'], ['78'], ['126'], ['77'], ['127'], ['53'], ['55'], ['340']	5	['127', '78'], ['340', '55'], ['53', '55'], ['126'], ['77']
0.2	6	['78'], ['126'], ['77'], ['127'], ['53'], ['55']	6	['78'], ['126'], ['77'], ['127'], ['53'], ['55']	5	['126', '127'], ['53'], ['55'], ['77']
0.3	2	['127'], ['78']	2	['127'], ['78']	1	['127']
0.4	1	['78']	1	['78']	0	-
0.5	0	-	0	-	0	-
0.6	0	-	0	-	0	-
0.7	0	-	0	-	0	-
0.8	0	-	0	-	0	-
0.9	0	-	0	-	0	-

Table 5.3. Execution time of algorithms according to given thresholds



After this step, It should be controlled whether created new itemsets have frequent subsets. Candidate Pruning step checks all frequent subsets. Hash tree is used to keep candidate itemsets and depth of the tree is k maximum. Populating the hash tree of the itemsets is $O(\sum_{k=2}^w k |C_k|)$ check that $k-2$ subsets are frequent then $O(\sum_{k=2}^w k(k-2) |C_k|)$ time is required. Now we have candidate itemsets and we should find support counts of itemsets. To count the support of the candidates, the algorithm needs to make an additional pass over the data set (steps 6-10). The *subset* function is used to determine all the candidate itemsets in C_k that are contained in each transaction t . In this section hash structure is used to keep number of each itemset. Itemsets are hashed into different buckets and using this method instead of comparing every itemset with all transactions, itemsets compared with available buckets. Assume that transactions have length $|t|$ and size of itemsets $|k|$ then $\binom{|t|}{|k|}$ itemsets are produced. The cost for counting is $O(N \sum_k \binom{w}{k} \alpha_k)$ where w width of maximum transaction and α_k is the cost to update the support count of a candidate k - *itemset* in the hash tree [32]. After counting their supports, the algorithm

eliminates all candidate itemsets whose support counts are less than minsup (step 12). The algorithm terminates when there are no new frequent itemsets generated in step 13 $F_k = \phi$ [32].

5.3.1.2. Analysis of FP-Growth Algorithm

This algorithm takes database and threshold and it gives frequent itemsets and it defines an empty list (F list) initially. FP-Growth algorithm uses a data structure called as FP-Tree to store all data. It scans database two times; at first scan support values of items are calculated, at second scan FP-Tree is constructed.

Algorithm consists of three steps; counting support values of items, creating FP-Tree and obtaining frequent itemsets from FP-Tree.

From Step 1 to 4, support values of items are calculated. Like Apriori, every transaction is traversed so assuming that total number of transactions is N and average transaction width is w , $O(Nw)$ time is required. At step 6, ordered items are added to F list. Then, in step 8-10 each transaction is ordered according to F list and meanwhile FP-Tree is constructed with *ConstructTree* function. Tree creation depends on the number of items in the database DB. To insert a transaction into tree, number of frequent items in transaction is considered which is equal to $O(|DB|)$. Then recursive function *Growth* is called to obtain frequent itemsets from tree paths. This operation takes $O(\text{header count}^2 * \text{depth of tree})$ times.

5.3.1.3. Analysis of Max-Miner Algorithm

Max-Miner algorithm extracts maximal frequent itemsets. It takes a transaction set and minimum support specified by the user implicitly as input. Algorithm calls function *GEN-INITIAL-GROUPS* to obtain frequent 1 – *itemsets* as given pseudocode in Appendix A. This function initially scan database and get frequent 1 – *itemsets*. Then it generates new candidates. After that, in main function, for every candidate, dataset is scanned and supports of all candidate groups are found.

Therefore, we start with *GEN-INITIAL-GROUPS* function to analyze algorithm. Assume N shows the number of transactions, w is average transaction width, F_1 denotes

the number of frequent 1-itemsets, l shows the length of the longest frequent itemset. Scanning all transactions T requires $O(Nw)$ like Apriori, then for each frequent item new candidate groups are created. It means *GEN-INITIAL-GROUPS* takes $O(Nw)$ times. This is the first database scan in Max-Miner algorithm. After analysing of *GEN-INITIAL-GROUPS* function, we return main function. It finds support values of all candidates which are obtained from initial function. If candidate is not frequent, according to *GEN-SUB-NODES* function, firstly infrequent items in $t(g)$ are removed. Item g is shown as $h(g) + t(g)$. Then each item i in $t(g)$ is merged with $h(g)$ and check whether items $h(g) + \{i\}$ are frequent.

This operation takes $(l - 1) * N$ times. Number of items in $t(g)$ is equal to $l - 1$ because first element in the longest itemset is in $h(g)$ and others are in $t(g)$ also N denotes the transaction number. Therefore, all transactions are checked number of items in $t(g)$ times means $(l - 1) * N$ times. Other operations take less time than this operation. *GEN-SUB-NODES* takes $O((l - 1) * N)$ times.

At final step, we examine main function Max-Miner, first scanning is performed by initial function. Then while loop is executed until variable C is empty. At worst case C becomes l (longest itemset size) because if longest itemset is frequent set of candidates group C_{new} becomes empty and it terminates the algorithm. In other case, if longest algorithm is infrequent we have to generate itemset longer than n which is impossible. It means that Max-Miner algorithm scans database at most $1+1$ times. Considering all functions, Max-Miner algorithm takes $O(Nw) + O((l - 1) * N)$ times.

5.4. Results

Patterns are found exactly the same from Apriori and FP-Growth algorithms. Max-miner algorithm finds only maximal patterns so it is the most efficient algorithm among these graphs. Apriori scans database many times so it takes a lot of time. FP-Growth scans two times and FP-Tree creation affects the algorithm.

CHAPTER 6

EVALUATION OF IMPLEMENTED METHODS

Our experimental question was finding the most frequent feature patterns on friendship relation of selected SNAP Facebook dataset. Therefore, we examined two different kinds of algorithms which are from graph mining and itemset mining to obtain frequent feature patterns. To evaluate these we modified algorithm or dataset. Both types of algorithms have pros and cons. For itemset mining, graph transformation requires preprocessing and after transformation, algorithms do not give exact graph structure. We extracted frequent itemsets that commonly seen in graph nodes using itemset mining algorithms. On the other hand, graph mining algorithms do not have to preprocess. It gives exact pattern structure. Yet, graph mining algorithms require costly operations. So, if studies do not need to graph structure, itemset mining can be a good option.

CHAPTER 7

CONCLUSION AND FUTURE WORK

Pattern mining algorithms obtain frequent patterns from different type of datasets i.e. transactional or graph. The most efficient way to represent a social network is graph data. However, the existent algorithms generally concentrate on single-labeled graphs. They assume that nodes or edges have only one label. Yet, complex graphs like social networks have more information than simple graphs, additionally, we have to consider the size of these datasets. They are accepted as big data. Therefore, to deal with such graphs become important.

In this thesis, we evaluate graph and itemset pattern mining algorithms on multi-labeled graph data to obtain common feature patterns. To work with gSpan algorithm which takes single-labeled graph as input, we separate entire graph into subgraphs whose nodes become single-labeled. After applying the algorithm we obtain patterns which include graph structure and common attributes. Moreover, we use Grami algorithm that handles multi-labeled graphs to find patterns. This algorithm works well for small number of features but performance of the algorithm is not well when the number of features increases.

While graphs reflect the interaction of networks successfully, from the analytical point of view, they face serious problems with algorithmic complexity. We try to convert graph data to itemset data. After transformation, itemset mining algorithms are applied. The result shows that itemset mining algorithms extract common attributes from all edges in a complex graph. However, graph structure and node id information are not kept. As a result, to benefit from transaction algorithms it needs preprocessing cost to transform graph data and it loses graph structure. For this reason, each approach has advantages and disadvantages according to desired patterns.

The result of experimental works shows that the most common feature patterns in friendship relationships of Facebook dataset are 78-78 and 127-127. It means that feature 127 and 78 are shown frequently in relationships and former indicates the location and latter represents gender information.

As future works; to mine the most frequent patterns;

1. on complex graph datasets; the complexity of existent solutions should be reduced,
 - new graph based algorithms can be explored,
 - existent algorithms work on static data, but SN's environment changes very dynamically, Artificial Intelligence and Machine Learning algorithms can present new ways on this objective. If graph data is transformed into suitable data format Machine learning algorithms can be applied to classify patterns.
2. New data representation alternatives can be studied to support this huge and very dynamic environment with their new algorithms.

REFERENCES

- [1] C. C. Aggarwal. *Data mining: the textbook*. Springer, 2015.
- [2] C. C. Aggarwal and J. Han. *Frequent pattern mining*. Springer, 2014.
- [3] C. C. Aggarwal and H. Wang. *Managing and Mining Graph Data*. Springer Publishing Company, Incorporated, 1st edition, 2010.
- [4] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *Acm sigmod record*, volume 22, pages 207–216. ACM, 1993.
- [5] R. Agrawal, R. Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.
- [6] R. J. Bayardo Jr. Efficiently mining long patterns from databases. *ACM Sigmod Record*, 27(2):85–93, 1998.
- [7] B. Bringmann and S. Nijssen. What is frequent in a single graph? *Advances in Knowledge Discovery and Data Mining*, pages 858–863, 2008.
- [8] B. Cule, B. Goethals, and T. Hendrickx. Mining interesting itemsets in graph datasets. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 237–248. Springer, 2013.
- [9] A. Dhiman and S. Jain. Frequent subgraph mining algorithms for single large graphs a brief survey. In *Advances in Computing, Communication, & Automation (ICACCA)(Spring), International Conference on*, pages 1–6. IEEE, 2016.
- [10] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis. Grami: Frequent subgraph and pattern mining in a single large graph. *Proceedings of the VLDB Endowment*, 7(7):517–528, 2014.
- [11] M. Fiedler and C. Borgelt. Subgraph support in a single large graph. In *Data Mining*

Workshops, 2007. ICDM Workshops 2007. Seventh IEEE International Conference on, pages 399–404. IEEE, 2007.

- [12] M. Flores-Garrido, J.-A. Carrasco-Ochoa, and J. F. Martínez-Trinidad. Agrap: an algorithm for mining frequent patterns in a single graph using inexact matching. *Knowledge and Information Systems*, 44(2):385–406, 2015.
- [13] P. Fournier-Viger, J. C.-W. Lin, B. Vo, T. T. Chi, J. Zhang, and H. B. Le. A survey of itemset mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2017.
- [14] M. Fukuzaki, M. Seki, H. Kashima, and J. Sese. Finding itemset-sharing patterns in a large itemset-associated graph. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 147–159. Springer, 2010.
- [15] E. Gudes. Graph and web mining-motivation, applications and algorithms. *International Journal on Software Bug Management*, 2010.
- [16] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *ACM sigmod record*, volume 29, pages 1–12. ACM, 2000.
- [17] T. Hendrickx, B. Cule, and B. Goethals. Mining cohesive itemsets in graphs. In *Discovery Science*, pages 111–122. Springer, 2014.
- [18] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. *Principles of Data Mining and Knowledge Discovery*, pages 13–23, 2000.
- [19] T. Karunaratne and H. Bostrom. Can frequent itemset mining be efficiently and effectively used for learning from graph data? In *Machine Learning and Applications (ICMLA), 2012 11th International Conference on*, volume 1, pages 409–414. IEEE, 2012.
- [20] V. Krishna, N. R. Suri, and G. Athithan. A comparative survey of algorithms for frequent subgraph discovery. *Current Science*, pages 190–198, 2011.

- [21] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 313–320. IEEE, 2001.
- [22] M. Kuramochi and G. Karypis. Grew-a scalable frequent subgraph discovery algorithm. In *Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on*, pages 439–442. IEEE, 2004.
- [23] M. Kuramochi and G. Karypis. Finding frequent patterns in a large sparse graph. *Data mining and knowledge discovery*, 11(3):243–271, 2005.
- [24] K. Lai and N. Cerpa. Support vs. confidence in association rule algorithms. In *Proceedings of the OPTIMA Conference, Curicó*, 2001.
- [25] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. <http://snap.stanford.edu/data>, June 2014.
- [26] Y. Mitsunaga, T. Washio, and H. Motoda. Mining quantitative frequent itemsets using adaptive density-based subspace clustering. *Transactions of the Japanese Society for Artificial Intelligence*, 21:439–449, 2006.
- [27] Y. Miyoshi, T. Ozaki, and T. Ohkawa. Frequent pattern discovery from a single graph with quantitative itemsets. In *Data Mining Workshops, 2009. ICDMW'09. IEEE International Conference on*, pages 527–532. IEEE, 2009.
- [28] F. Moser, R. Colak, A. Rafiey, and M. Ester. Mining cohesive patterns from graphs with feature vectors. In *Proceedings of the 2009 SIAM International Conference on Data Mining*, pages 593–604. SIAM, 2009.
- [29] C. Pasquier, F. Flouvat, J. Sanhes, and N. Selmaoui-Folcher. Attributed graph mining in the presence of automorphism. *Knowledge and Information Systems*, 50(2):569–584, 2017.
- [30] I. Robinson, J. Webber, and E. Eifrem. *Graph databases: new opportunities for connected data.* ” O’Reilly Media, Inc.”, 2015.

- [31] J. Sese, M. Seki, and M. Fukuzaki. Mining networks with shared items. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 1681–1684. ACM, 2010.
- [32] P.-N. Tan et al. *Introduction to data mining*. Pearson Education India, 2006.
- [33] L. Thomas, S. Valluri, and K. Karlapalem. Isg: Itemset based subgraph mining. Technical report, Technical Report, IIIT, Hyderabad, December2009, 2009.
- [34] G. Xu, Y. Zong, and Z. Yang. *Applied data mining*. CRC Press, 2013.
- [35] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 721–724. IEEE, 2002.
- [36] M. J. Zaki, W. Meira Jr, and W. Meira. *Data mining and analysis: fundamental concepts and algorithms*. Cambridge University Press, 2014.

APPENDIX A

PSEUDOCODE OF ALGORITHMS

Algorithm 1 Frequent Subgraph Mining (Grami)

Input: A graph G and the frequency threshold τ ;

Output: All subgraph S of G such that $S_G(S) \geq \tau$

```
1: result  $\leftarrow \theta$ 
2: Let fEdges be the set of all frequent edges of  $G$ 
3: for each  $e \in fEdges$  do
4:   result  $\leftarrow result \cup SubgraphExtension(e, G, \tau, fEdges)$ 
5:   remove  $e$  from  $G$  and fEdges
6: return result
```

Algorithm 2 SubgraphExtension (Grami)

Input: A subgraph S of a graph data G , the frequency threshold τ and the set of frequent edges *fEdges* of G ;

Output: All frequent subgraphs of G that extend S

```
1: result  $\leftarrow S$ , candidateSet  $\leftarrow \theta$ 
2: for each edge  $e$  in fEdges and node  $u$  of  $S$  do
3:   if  $e$  can be used to extend  $u$  then
4:     Let ext be the extension of  $S$  with  $e$ 
5:     if ext is not already generated then
6:       candidateSet  $\leftarrow candidateSet \cup ext$ 
7: for  $c \in candidateSet$  do
8:   if  $S_G(c) \geq \tau$  then
9:     result  $\leftarrow result \cup SubgraphExtension(c, G, \tau, fEdges)$ 
10: return result
```

Algorithm 3 SubgraphExtension (Modified Grami)

Input: A subgraph S of a graph data G , the frequency threshold τ and the set of frequent edges $fEdges$ of G ;

Output: All frequent subgraphs of G that extend S

```
1: result  $\leftarrow S$ , candidateSet  $\leftarrow \theta$ 
2: for each edge  $e$  in  $fEdges$  and node  $u$  of  $S$  do
3:   if  $e$  can be used to extend  $u$  then
4:     Let ext be the extension of  $S$  with  $e$ 
5:     if ext does not include same node items then
6:       if ext is not already generated then
7:         candidateSet  $\leftarrow candidateSet \cup ext$ 
8:   for  $c \in candidateSet$  do
9:     if  $S_G(c) \geq \tau$  then
10:    result  $\leftarrow result \cup SubgraphExtension(c, G, \tau, fEdges)$ 
11: return result
```

Algorithm 4 BFS Itemset Transformation Algorithm

Input: Graph: Ego Graph;

start: Starting vertex

Output: Text file containing transactions

```
1: for each  $v \in Graph$  do
2:   visited_vertex[v] := False
3: end for
4: Queue :=  $\phi$ 
5: Enqueue(Queue, start)
6: while Queue is not empty do
7:   vertex := Dequeue(Queue)
8:   if vertex not in visited_vertex then
9:     visited_vertex[vertex] := True
10:    for each  $w$  in Graph.Adj[vertex] do
11:      if  $w$  is not in visited_vertex then
12:        node_features := features of vertex
13:        neighbor_features := features of  $w$ 
14:        for each feature in node_features do
15:          if feature in neighbor_features then
16:            add feature to common_features
17:          end if
18:        end for
19:        if common_features > 0 then
20:          write common features to file
21:        end if
22:        Enqueue(Queue,  $w$ )
23:      end if
24:    end for
25:  end if
26: end while
```

Algorithm 5 Apriori Algorithm

Input:

D: transaction database;

min_sup: Minimum support threshold

Output: frequent itemsets

```
1:  $k = 1$ 
2:  $F_k = \{large\ 1 -\ itemsets\}$ ;
3: repeat
4:    $k = k + 1$ 
5:    $C_k = \text{apriori\_gen}(F_{k-1})$ ; // New candidates are generated
6:   for each transaction  $t \in D$  do begin
7:      $C_t = \text{subset}(C_k, t)$ ; // Candidates contained in t
8:     for each candidate itemset  $c \in C_t$  do
9:        $\sigma(c) = \sigma(c) + 1$ ; // Increment support count
10:    end for
11:  end for
12:   $F_k = \{c \in C_k | c.count \geq min\_sup\}$ 
13: until  $F_k = \phi$ 
14: return  $F = \bigcup_k F_k$ ;
```

Algorithm 6 FP-Growth Algorithm

Procedure: FPGrowth(DB, ξ)

Define and clear F-List: F[]

```
1: for each transaction  $T_i$  in DB do
2:   for each item  $a_j$  in  $T_i$  do
3:      $F[a_i] ++$ ;
4:   end for
5: end for
6: Sort F[];
7: Define and clear the root of FP-tree : r;
8: for each transaction  $T_i$  in DB do
9:   Make  $T_i$  ordered according to F;
10:  Call  $ConstructTree(T_i, r)$ ;
11: end for
12: for each item  $a_i$  in I
13:  for each transaction  $T_i$  in DB do
14:    Call  $Growth(r, a_i, \xi)$ 
15:  end for
```

Algorithm 7 Max-Miner Algorithm

Input:

D: transaction database;

min_sup: Minimum support threshold

Output: set of maximal frequent itemsets

Define and clear set of candidate groups C

```
1: Set of itemsets  $F \leftarrow Gen\_Initial\_Groups(D, C)$ 
2: while C is non-empty do
3:   scan D to count the support of all candidate groups in C
4:   for each  $g \in C$  such that  $h(g) \cup t(g)$  is frequent do
5:      $F = F \cup \{h(g) \cup t(g)\}$ 
6:   Set of Candidate Groups  $C_{new} \leftarrow \{\}$ 
7:   for each  $g \in C$  such that  $h(g) \cup t(g)$  is infrequent do
8:      $F \leftarrow F \cup Gen\_Sub\_Nodes(g, C_{new})$ 
9:    $C \leftarrow C_{new}$ 
10:  remove from F any itemset with a proper superset in F
11:  remove from C any group g such that  $h(g) \cup t(g)$  has a superset in F.
12: return F
```
