CHAPTER 2

LITERATURE SURVEY

2.1 Introduction

Performing a research in a selected area requires ultra structured review and study of literature related to that area. An analysis of the literature is nothing but the collection of information regarding what has been done in the same area and what kind of modifications are required for a significant investigation. This chapter of the thesis provides a detailed review of the literature, the theory and related topics in the field of data mining, GDM, frequent subgraph mining, interestingness measures and applications.

This chapter explains some fundamentals about frequent subgraph generation which helps to develop a new algorithm for ranking that subgraphs. This chapter also describes (section 2.2) about Association Rule Mining (ARM) algorithms regarding the frequent itemset mining since these are the base for the frequent subgraph mining and initiated to develop new algorithms for handling similarity rule and ranking those rules.

Many papers on data mining of semi-structured data have been presented in major international journals and conferences.

The research on data mining and machine learning of symbolic sequences also became active in the last several years. Many papers appeared in ILP, ALT and DS conferences.

Furthermore, in the last few years, many researchers started working on mining ordered tree structures.
One of the most recent research topics associated with structured data is multi-relational data mining whose main scope is to find patterns in expressive logical and relational languages from complex, multi-relational and structured data [86].

The main aim of mining semi-structured data, symbolic sequences and ordered trees is to extract patterns from structured data. Within this framework, the patterns mined are characterized by some measures such as frequency and information entropy is mined. The classes of the patterns handled in the multi-relational data mining are more expressive than the aforementioned data structures.

Recently, a novel field of data mining emerged from a topological view of the data structure. In mathematics, one of the most generic topological structures is graphs.

Regarding the graph structure, the theoretical basis of GDM is not limited to one principle although the history of this research field is still young. This is because research on graphs has a long history in mathematics.

In recent years, the use of graph representations has gained popularity in pattern recognition and machine learning [102] [79] [98]. The main advantage of graphs is that they offer a powerful way to represent structured data.

Among other applications, attributed graphs have been used to address the problem of graphical symbol recognition [65], character recognition [101] [88], shape analysis [94], biometric person authentication by means of facial images [105] and fingerprints [67], computer network monitoring [61], Web document analysis [89], and data mining [48].

The data mining ability to extract useful information has become one of the most important challenges in many fields. If the mined data represents a set of independent entities and their attributes it is very successful. In most domains, there is an interesting knowledge to be mined from the relationships between entities. This relational knowledge may take many forms from periodic patterns of transactions to complicated structural patterns of interrelated transactions. Extracting such
knowledge requires the data to be represented in a form that not only captures the relational information but supports efficient and effective mining of this data and comprehensibility of the resulting knowledge. Relational databases and first-order logic are two popular representations of relational data, but neither has sufficiently supported the data mining process.

All aspects of the relational data mining process were supported by the graph (collection of nodes and links between nodes). The graph is the one of the most general forms of data representation. One of the best studied data structures in computer science and discrete mathematics are graphs. It can therefore be no surprise that graph based data mining has become quite popular in the last few years. The graph easily represents entities, their attributes, and their relationships to other entities. One entity can be arbitrarily related to other entities, relational databases and logic have difficulty in organizing the data to support efficient traversal of the relational links. Graph representations typically store each entity’s relations with the entity. Relational database and logic representations do not support direct visualization of data and knowledge. In fact, relational information stored in this way is typically converted to a graph form for visualization. Using a graph for representing the data and the mined knowledge supports direct visualization and increased comprehensibility of the knowledge. Therefore, mining graph data is one of the most promising approaches of extracting knowledge from relational data.

2.2 Association Rule Mining (ARM)

The objective of an ARM is the discovery of interesting rules from which new knowledge can be derived. ARM enables the presentation of these rules for subsequent interpretation to determine their usefulness. The process of ARM has been divided into two parts as follows:

Part 1: Identify the set of frequent items/itemsets/pattern within the set of transaction using user-specified support threshold.
Part 2: Generate the rules from these above patterns using user-specified confidence threshold.

The above two parts are generating *strong association rules* from the dataset. The first part is known as *frequent itemset mining*. This part is computationally expensive than the part 2. The second part is called *association rule generation*. This is a straight forward process.

The ARM process is divided into three areas. They are (1) Positive rule mining, (2) Negative rule mining and (3) Interestingness measures. The positive rules are mined from frequent itemsets. Due to frequent itemset mining deficiency, the formats of the frequent itemsets are extended to closed, maximum, sequential, complex frequent itemset. The above types of frequent itemset are supported to constraint based rule mining. The negative relationships between itemset are mined by rule mining process using infrequent itemset. The third category interestingness measures are played important role in the field of ARM similar to the data mining process.

2.3 Frequent Pattern Mining

A dataset consists of patterns which are set of item, sequences, graph or structures. The fundamental role in ARM is finding frequent patterns which are the most important one for classification, clustering, and other data mining tasks. Frequent pattern mining was first proposed by Agrawal et al [1] for market basket analysis in the form of ARM. The fundamental frequent pattern algorithms are classified into three types as follows:

1. Candidate generation approach (E.g. Apriori algorithm)
2. without candidate generation approach (E.g. FP-growth algorithm)
3. Vertical layout approach (E.g. Eclat algorithm)
2.3.1 With Candidate Generation Approach

The first algorithm for frequent itemset mining was developed and is referred as AIS [1]. Later, the algorithm was improved by Apriori and called by the same name. The main improvement developed by the Apriori is the monotonicity property of the support of sets [5] [97]. After the improvement, the monotonicity was further undergone better treatment by Mannila et al [68] and Agarwal et al [2]. The Apriori algorithm was based on the candidate generation approach. The methods involved in this algorithm were discussed in chapter 5.

2.3.2 Without Candidate Generation Approach

An algorithm developed as a first one without candidate generation approach was the algorithm devised by Han et al [43] and named as FP-growth method. This FP-growth method mines the complete set of frequent itemsets without candidate generation. It adopt the divide-and-conquer policy. During the first scan, the database derives a list of frequent items by a frequency descending order. Then the database is compressed into a frequent pattern tree (FP-tree) using the above frequency descending order list. The FP-tree is mined by starting from each frequent length-1 pattern, constructing its conditional pattern base, then constructing its conditional FP-tree, and performing mining recursively on such a tree. The pattern growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree. The concatenation is made by transforming the problem of finding long frequent patterns to searching for shorter ones recursively and then concatenating the suffix. It uses the least frequent items as a suffix, offering good selectivity. The performance studies of FP-Growth exhibit that the method significantly reduces search time.

2.3.3 Vertical Layout Approach

In this vertical layout approach, the first algorithm developed was Eclat (Equivalence CLAss Transformation) algorithm [116]. The approach of this algorithm is to generate all frequent itemsets in a depth-first manner. The technique used in this algorithm is, if the database is stored in the vertical layout, the counting of
support can be done by simply intersecting the covers of two of its subsets that together give the set itself. Eclat algorithm uses this technique inside the Apriori algorithm. Since the total size of all covers at a certain iteration of the local set generation procedure could exceed main memory limits, always this is not possible. It is usually most efficient to find the frequent items and frequent 2-sets separately and Eclat algorithm was used only for all larger sets.

2.3.4 Measures and its Types

ARM is nothing but an algorithmic process in which data are taken as input and producing an output as a discovered pattern. In this scenario, interestingness measures play a very important role to reduce the number of discovered pattern and also to retain only the best ones. In the literature, there are thirty two measures used in data mining to mine interestingness rules [50]. PangNing Tan, Vipin Kumar and Jaideep Srivastava [104] described several key properties one should examine in order to select the right measure for a given application domain. These properties are used with twenty one of the existing measures. It showed that each measure has different properties which make them useful for some application domains, but not for others. There are nine specific criteria used to determine whether a pattern is interesting or not [38]. They are described as follows:

1. **Conciseness**: A concise pattern or set of patterns is relatively easy to understand and remember and thus is added more easily to the user’s knowledge or set of beliefs.

2. **Generality / Coverage**: A pattern is general if it covers a relatively large subset of a dataset.

3. **Reliability**: A pattern is reliable if the relationship described by the pattern occurs in a high percentage of applicable cases.

4. **Peculiarity**: A pattern is peculiar if it is far away from other discovered patterns according to some distance measure.
5. **Diversity**: A pattern is diverse if its elements differ significantly from each other, while a set of patterns is diverse if the patterns in the set differ significantly from each other.

6. **Novelty**: A pattern is novel to an user if the user did not know it before and is not able to infer it from other known patterns.

7. **Surprisingness**: A pattern is surprising (or unexpected) if it contradicts a person’s existing knowledge or expectations.

8. **Utility**: A pattern is of utility if its use by a person contributes to reach a goal.

9. **Actionability/Applicability**: A pattern is actionable (or applicable) in some domains if it enables decision making about future actions in the domains.

Based on the above nine criteria, the measures are classified as i) Subjective Measures, ii) Objective Measures and iii) Semantic Measures.

**2.3.4.1 Subjective Measures**

Subjective techniques generally operate by comparing the user’s beliefs against the patterns discovered by the algorithm [95]. A subjective measure considers both the data and user of the dataset. A subjective measure will access the user’s background knowledge about the required data. Novelty and surprisingness depend on the user as well as the data and patterns themselves, and hence considered as subjective. The main disadvantage of the subjective measure is that it constraint the discovery process only to find what the user’s anticipation or hypothesis i.e. it cannot discover unexpected or unforeseen patterns because it is entirely goal driven [60].

**2.3.4.2. Objective Measures**

This measure is based only on the raw data. The users are not required of any knowledge about the application. Most of these objective measures are based on theories in probability, statistics, or information theory. But conciseness, generality, reliability, peculiarity, and diversity depend only on the data and patterns, and thus
can be considered objective. Objective measures or data driven measures tend to concentrate on finding patterns through statistical strength or correlations [60].

2.3.4.3. Semantic Measures

A semantic measure considers the semantics and explanations of the patterns. Utility and functionality depend on the semantics of the data, and hence can be considered semantic.

2.3.5 Role of Measures

The role of measures can be explained under three categories as follows:

1. They helped to identify each pattern as interesting or uninteresting.
2. The measures are used to determine one pattern which is more interesting than the other.
3. Also the measure helps to rank the interesting or useful patterns.

2.3.6 List of Measures

Given the set of transactions or subgraphs, some measures are to be satisfied. They are

- Support
- Confidence and
- Lift

The support and confidence values are often used to define rule constraints by bounding them above a pre-specified value known as minsup and minconf respectively and also to define total orders for optimization. Lift is the statistical measure to provide the relationship between the two transactions.
2.3.6.1 Support

It is a basic measure related to probability and set theory. It is defined as the fraction of transactions in the database which contain all items in a specific rule [1]. Support measure concerned with the number of transactions in T that support a rule. The support for a rule is defined to be the fraction of transactions in T that satisfy the union of items in the consequent and antecedent of the rule. Support should not be confused with confidence. While confidence is a measure of the rule's strength, support corresponds to statistical significance. Besides statistical significance, another motivation for support measures comes from the fact that it is usually interested only in rules with support above some minimum threshold for business reasons. If the support is not large enough, it means that the rule is not worth considered or it is simply less preferred. Support is the ratio of the number of transactions that include all items in the antecedent and consequent parts of the rule to the total number of transactions.

Usually in data mining tasks, a number $0 \leq \sigma \leq 1$, called the minimum support threshold is provided to the system. An itemset $I$ that has $S(I) \geq \sigma$ is called frequent. Defined in this manner, the support of an item set $I$ is always not greater than the support of any of the subsets of $I$. This is the fundamental property of the support measure.

Given a graph $G$, the support of $G$ is defined as

$$\text{sup}(G) = \frac{\text{number of graph transactions } G, \text{ where } G, \subset G \in GD}{\text{total number of graph transactions } G \in GD}$$

In other words, The support of a condition $A$ is equal to the number of records in the data-set for which $A$ evaluates to true, and this value is denoted as sup($A$). The support of a rule $A \rightarrow C$, denoted similarly as sup($A \rightarrow C$), is equal to the number of records in the data-set for which both $A$ and $C$ evaluate to true. The antecedent support of a rule is the support of its antecedent alone.
2.3.6.2 Confidence

Another measure of the association rules is confidence [1]. This is the strength of the implication of a rule and can be represented as a ratio between the transaction numbers, including \( x \) and \( y \) and those including \( X \), and \( X \) means that \( x \cup y \). That is, confidence is the ratio of the number of transactions that include all items in the consequent as well as antecedent to the number of transactions that include all items in antecedent. Given two induced subgraphs \( G_b \) and \( G_h \), the confidence of the association rule \( G_b \Rightarrow G_h \) is defined as

\[
\text{conf}(G_b \Rightarrow G_h) = \frac{\text{number of graphs } G \text{ where } G_b \sqcup G_h \subseteq G \in GD}{\text{number of graphs } G \text{ where } G_h \subseteq G \in GD}
\]

Confidence can also be evaluated in terms of support as

\[
\text{conf}(A \rightarrow C) = \frac{\text{sup}(A \rightarrow C)}{\text{sup}(A)}
\]

2.3.6.3 Lift

“Lift” is the most commonly used metric to measure the performance of targeting models. The purpose of targeting model is to identify a subgroup from a larger population. Generally, lift can be calculated by looking at the cumulative targets captured up to \( p\% \) as a percentage of all targets and dividing by \( p\% \) ie. Lift is simply the ratio of target response divided by average response[12].

Lift measures show the relationship between two or more items when they occur together more often than expected and if they were statistically independent. Lift originally called Interest, was first introduced by Motwani, et al., (1997) [12]. It measures the number of times \( A \) and \( B \) occur together compared to the expected number of times if they were statistically independent and is given by

\[
Lift = \frac{P[A \cap B]}{P[A]P[B]}
\]
with the obvious extensions to more than two sets[10].

Lift can also be framed as a function of confidence [32]

\[
\text{lift}(A \rightarrow C) = \frac{|D|_{\text{conf}}(A \rightarrow C)}{\text{sup}(C)}
\]

Where, the support and confidence measures were already discussed in the previous sections.

By this conviction, Lift is obviously monotone in confidence and unaffected by rule support when confidence is held fixed.

It’s hard to say what level of lift represents "good" model performance, because the potential for predictive targeting varies widely between applications. Rather than using lift to evaluate a model in isolation, it is often more reliable to use lift to evaluate the relative performance of alternative models. If a tree model provides higher lift than a neural net on the same data, this provides a key factor in choosing between the models.

In the case of subgraph architecture, the above definition of the lift is considered and that can be defined as

\[
\text{Lift} = \left( F_{\text{lift}}^G \right) = \frac{\text{number of graph of } F_a \text{ and } F_b}{\text{number of } F_b \times \text{number of } F_a} = \frac{P(XUY)}{P(X)P(Y)}
\]

The relationship of \( X \) and \( Y \) are defined by the lift as

i) Lift value > 1 then \( X \) and \( Y \) depend on each other
ii) Lift value < 1 then \( X \) depends on the absence of \( Y \) or vice-versa
iii) Lift value close to 1 then \( X \) and \( Y \) are independent.
2.4 Graph Mining

Mining knowledge from structured data is a major research topic in recent study of Data Mining. In structured data, graph structure is one of the representations, because it frequently appears in real-world data. Graph is an alternate way of modeling the objects. Modeling objects using graphs allows us to represent arbitrary relations among entities. Mining of graph data is the visualization of the discovered knowledge. *Graph visualization* is the rendering of the nodes, links, and labels of a graph in a way that promotes easier understanding of human beings from the concepts represented by the graph. Graphs have been used for modeling entities and their relationships such as the Internet, the web, social networks, metabolic networks, protein interaction networks, food webs, citation networks, and many more. Graph mining refers to the process of identifying interesting patterns and extracting knowledge from large graph databases.

2.4.1 Graph Database

The key advantage of graph modeling is that it allows solving problems which could not be solved previously. For instance, consider a problem of mining chemical compounds to find recurrent substructures. This can be achieved by using a graph-based pattern discovery algorithm by creating a graph for each one of the compounds for which vertices correspond to different atoms, and whose edges correspond to bonds between them. Assign to each vertex a label corresponding to the atom involved (and potentially its charge), and assign to each edge a label corresponding to the type of the bond (and potentially information about their relative 3D orientation). Once these graphs have been created, recurrent substructures across different compounds become frequently occurring subgraphs.

The graph representation, that is, a collection of nodes and links between nodes, does support all aspects of the relational data mining process. As one of the most general forms of data representation, the graph easily represents entities, their attributes, and their relationships to other entities. There are many descriptions for several diverse domains and how graphs can be used to represent the domain. Because one entity can be arbitrarily related to other entities, relational databases and logic
have difficulty in organizing the data to support efficient traversal of the relational links.

Graph representations typically store each entity’s relations with the entity. Finally, relational database and logic representations do not support direct visualization of data and knowledge. In fact, relational information stored in this way is typically converted to a graph form for visualization. Using a graph for representing the data and the mined knowledge supports direct visualization and increased comprehensibility of the knowledge. Therefore, mining graph data is one of the most promising approaches to extracting knowledge from relational data.

2.4.1.1 Conversion of Graph Database

It is very important to represent every data set as a graph database, the structure of a graph dataset would follow some conversions and to clarify some issues. Those conversion methods and issues are discussed in the following sections.

Sparse Graph Representation: There is no strict distinction between sparse and dense graphs. The advantage of the sparse representation is for graph algorithms which are linear in the number of edges, and more generally, for those algorithms based on the adjacency list representation (the flow-based algorithms and the shortest-paths algorithms). Further, the sparse representation is required when creating multi graphs since they may have multiple edges.

Sparse Graph Definition: A graph in which the number of edges is lesser than the possible number of edges. Any graph which is represented by means of an adjacency list is called as Sparse Graph.

The FSG algorithm uses sparse graph representation to store input transactions, intermediate candidates and frequent subgraphs. Each one of the transactions, candidates and discovered frequent subgraphs are stored using adjacency-list representation, while the canonical labeling is described in next section which is based on adjacency matrix representation. Thus, after determining canonical label for a subgraph, then the canonical adjacency matrix is converted back into
adjacency lists. This adjacency-lists representation saves memory when input transaction graphs are sparse, and speeds up computation.

**Canonical Labeling:** There are many differences between this algorithm and the traditional frequent itemset discovery. A difference appears when sorting of an object is attempted. In the traditional frequent itemset discovery, itemsets can be sorted by lexicographic ordering. Clearly this is not applicable to graphs. To get total order of graphs, canonical labeling is used. A canonical label is a unique code of a given graph [11] [46]. A graph can be represented in many different ways, depending on the order of its edges or vertices. Nevertheless, canonical labels should be always the same no matter how graphs are represented, as long as those graphs have the same topological structure and the same labeling of edges and vertices. By comparing canonical labels of graphs, they can be sorted in a unique and deterministic way, regardless of the representation of input graphs. The canonical label of a graph g is denoted by by \( cl(g) \). It is easy to see that computing canonical labels is equivalent to determining isomorphism between graphs, because if two graphs are isomorphic with each other, their canonical labels must be identical. Both canonical labeling and determining graph isomorphism are not known to be either in P or in NP-complete [46].

A straightforward way of determining a canonical label is to use a flattened representation of the adjacency matrix of a graph. Namely, by concatenating rows or columns of an adjacency matrix one after another a list of integers are constructed. By regarding this list of integers as a string, total order of graphs by lexicographic ordering can be obtained. To compute a canonical label of a graph, all the permutations of its vertices are tried to see which order of vertices gives the minimum adjacency matrix. To narrow down the search space, first partition was done with the vertices by their degrees and labels, which is a well-known technique called vertex invariants[11]. Then, all the possible permutations of vertices inside each partition are tried. Let an example be considered to see how the search space of canonical labeling with vertex invariants is reduced.
An example is shown here to see how the search space of canonical labeling with vertex invariants can be reduced. The graph of size 3 ($g_3$) is considered as shown in figure 2.1 below. $A, B, C$ and $D$ are the vertex identifiers. Label $e_a$ is given for two edges of $g_3$, and the other edge of the graph has a label $e_b$. Vertices label for $A, B$ and $D$ is $v_a$, and only $C$ is labeled with $v_b$. A canonical label is formed by concatenating column 7 in the upper triangle of an adjacency matrix from left to right and is a string.

![Figure 2.1: Graph of size 3](image)

The initial adjacency matrix of the graph $g_3$ is considered as below:

<table>
<thead>
<tr>
<th>label id</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$v_a$</td>
<td>$v_a$</td>
<td>$v_b$</td>
<td>$v_a$</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>$e_a$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>$e_a$</td>
<td>0</td>
<td>$e_a$</td>
<td>$e_b$</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>$e_a$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>$e_b$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
The vertex degrees are partitioned into two groups. One group belongs to degree 1 and the other for degree 2. Vertices \( A, C \) and \( D \) belong to the degree 1 and \( B \) to the degree 2.

**Table 2.2: Partitioned Vertex**

<table>
<thead>
<tr>
<th>label</th>
<th>A</th>
<th>C</th>
<th>D</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>id partition</td>
<td>( v_a )</td>
<td>( v_b )</td>
<td>( v_a )</td>
<td>( v_a )</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( e_a )</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( e_a )</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( e_b )</td>
</tr>
<tr>
<td>B</td>
<td>( e_a )</td>
<td>( e_a )</td>
<td>( e_b )</td>
<td>0</td>
</tr>
</tbody>
</table>

Because \( v_a < v_b \), the first partition is again divided into two.

**Table 2.3: Further Partitioned Vertex**

<table>
<thead>
<tr>
<th>id label partition</th>
<th>D</th>
<th>A</th>
<th>C</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( v_a )</td>
<td>( v_a )</td>
<td>( v_b )</td>
<td>( v_a )</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( e_b )</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( e_a )</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( e_a )</td>
</tr>
<tr>
<td>B</td>
<td>( e_b )</td>
<td>( e_a )</td>
<td>( e_a )</td>
<td>0</td>
</tr>
</tbody>
</table>
At this stage, there will be no possibility of further partitioning. Thus, all the possible permutations of vertices within each partition exhaustively tested, and two different permutations of the vertices as shown below were obtained.

**Table 2.4: Permutation-1 of the vertices**

<table>
<thead>
<tr>
<th>label</th>
<th>D</th>
<th>A</th>
<th>C</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>$v_a$</td>
<td>$v_a$</td>
<td>$v_b$</td>
<td>$v_a$</td>
</tr>
<tr>
<td>partition</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$e_b$</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$e_a$</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$e_a$</td>
</tr>
<tr>
<td>B</td>
<td>$e_b$</td>
<td>$e_a$</td>
<td>$e_a$</td>
<td>0</td>
</tr>
</tbody>
</table>

**Table 2.5: Permutation-2 of the vertices**

<table>
<thead>
<tr>
<th>label</th>
<th>A</th>
<th>D</th>
<th>C</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>$v_a$</td>
<td>$v_a$</td>
<td>$v_b$</td>
<td>$v_a$</td>
</tr>
<tr>
<td>partition</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$e_a$</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$e_b$</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$e_a$</td>
</tr>
<tr>
<td>B</td>
<td>$e_a$</td>
<td>$e_b$</td>
<td>$e_a$</td>
<td>0</td>
</tr>
</tbody>
</table>

The right label of the matrix gives “000$e_a$eb$e_a$”, while the left “000$e_b$eb$e_a$”. Because $e_a < e_b$ and “000$e_a$eb$e_a$” $< “000$e_b$eb$e_a$” by string comparison, the label at the right becomes canonical and its adjacency matrix is the canonical representation of $g_3$, that is $CL (g_3) = 000e_aeb_a$. By partitioning based on vertex invariants, only 2 permutations were performed in the last step, although the total number of permutations for 4 vertices was $4! = 24$. 
Similarly $N$ partitions of the vertices by vertex invariants in a graph of $M$ vertices can be created. Each partition size is given by $p_i$ for $i = 1, 2... N$. Clearly $\sum_{i=1}^{N} p_i = M$. Then, the reduced search space becomes $\prod_{i=1}^{N} (p_i!)$ although the original was $M!$.

**Isomorphism:** Graph isomorphism is a problem to find out whether given two graphs $g_1$ and $g_2$ are isomorphic, i.e., to find a mapping from a set of vertices to another set. Two isomorphisms are to be solved. One is graph isomorphism and another is subgraph isomorphism. Subgraph isomorphism is to find an isomorphism between $g_1$ and a subgraph of $g_2$. In other words, it is to determine if a graph is included in the other larger graph. The subgraph isomorphism was thoroughly discussed in the section 2.8.2. There exists another isomorphism called Automorphism which is a special case of graph isomorphism where $g_1 = g_2$, which means to find a mapping from a graph to itself.

A well-known algorithm for subgraph isomorphism is proposed in [13]. As suggested in [46], graph isomorphism can be directly solved in practice, although it is not known to be either in P or in NP-complete. On the other hand, subgraph isomorphism has been proved to be in NP-complete [47]. Thus, there is no scalable algorithm to solve it. When the size of graphs is small such as 10 vertices or less, however, it is also known that subgraph isomorphism can be feasible even with a simple exhaustive search [46] [13]. The graph isomorphism is solved by a simple way, which is, starting from a single vertex in one graph, to try to find a mapping to one of the vertices in the other graph that is consistent with the labeling. Then, keep the same process by adding vertices one by one until either reaching a complete mapping or end up with an exhausting the search space. When seeking for the next mapping, careful steps are taken to keep the consistency of edge and vertex labels. The search space can be reduced more if there are more labels are assigned to edges and vertices, which leads to restriction against mapping. This approach can solve both graph and subgraph isomorphism.
2.5 Subgraph Mining

The amount of available data is increasing very fast. With this data, the desire for data mining is also growing. More and larger databases have to be searched to find interesting (and frequent) elements and connections between them. Most often the data of interest is very complex. It is common to model complex data with the help of graphs consisting of nodes and edges that are often labeled to store additional information. Having a graph database, the main goal is to find connections and similarities between its graphs. Based on these connections and similarities, the graphs can be categorized, clustered or changed according to the application area. Regularly occurring patterns in the form of subgraphs — called *fragments* in this context — that appear at least in a certain percentage of graphs, are a common method to analyze graph databases. The actual occurrence of a fragment in a database graph is called *embedding*. Finding the fragments and their embeddings is the goal of subgraph mining [51].

Efficient algorithms for finding frequent patterns—both sequential and nonsequential in very large datasets have been one of the key success stories of data mining research [5] [4] [39] [80] [96] [117]. One way of formulating the frequent pattern discovery problem for graph datasets is that of discovering subgraphs occurring frequently in the given input graph dataset.

2.5.1 Subgraph Categories

Various classes of substructures are targeted in GDM. This is because the graph is one of the most generic data structures and includes characteristic substructures in various views. Mathematically, a graph G is represented as $G(V; E; f)$ where $V$ is a set of vertices, $E$ a set of edges connecting some vertex pairs in $V$, $f$ a mapping $f : E \rightarrow V \times V$. The most generic class of the substructure of $G$ is a “general subgraph” where $V_s \subset V$, $E_s \subset E$ and $v_i; v_j \in V_s$ for all edges $f(e) = (v_i; v_j) \in E_s$. Another important and generic class of the substructure is an “induced subgraph” where $V_s \subset V$, $E_s \subset E$ and $\forall v; v_j \in V_s$, $e_h = (v_i; v_j) \in E_s \Leftrightarrow f(e_h) = (v_i; v_j) \in E$. An induced subgraph $G_s'$ of a graph $G$ has a subset of the vertices of $G$ and the same edges between pairs of vertices as in $G$. The third important and generic class of the
substructure is a “connected subgraph” where $V_s \subset V, E_s \subset E$ and all vertices in $V_s$ are mutually reachable through some edges in $E_s$. An acyclic subgraph is called a “tree”. Though the labels of vertices and edges are not considered in the aforementioned graph formulation, if we introduce the labels of edges in the tree, and if they are ordered in a way that the label of an edge is always younger than the labels of its lower (upper) and right (left) edges, the tree is defined as an “ordered tree”. If the edge is not ordered or does not have labels, the tree is called an “unordered tree”. If the substructure does not include any branches, it is called a “path” of the original $G$.

2.5.2 Problem Statement

Graphs become increasingly important in modeling complicated structures, such as circuits, images, chemical compounds, protein structures, biological networks, the Web, work flows, and XML documents. There are three key issues regarding the problem definitions. Those issues are

- type of graphs that these algorithms operate on
- type of subgraphs that would be find and
- Whether or not want to find the complete set of patterns.

Depending on the combinations of the issues, different problems may exist. There are two distinct forms of the input to a frequent subgraph discovery algorithm, which are referred to as

- Graph-transaction setting and
- The single-graph setting.

In the graph-transaction setting, the input to a pattern mining algorithm is a set of relatively small graphs (called transactions), whereas in the single-graph setting, the input data is a single large graph. The difference affects the way the frequency of the various patterns is determined. For the graph-transaction setting, the frequency of a pattern is determined by the number of graph transactions that the pattern occurs in,
irrespective of how many times a pattern occurs in a particular transaction, whereas in
the single-graph setting the frequency of a pattern is based on the number of its
occurrences (i.e., embeddings) in the single graph. Due to the inherent differences of
the characteristics of the underlying dataset and the problem formulation, algorithms
developed for the graph-transaction setting cannot be used to solve the single-graph
setting, whereas the latter algorithms can be easily adapted to solve the former
problem.

The earliest studies to find subgraph patterns characterized by some measures
from massive graph data were conducted by Cook and Holder (SUBDUE) [19] and
Yoshida and Motoda (GBI) [113] in the middle of the 1990's. Their approaches used
greedy search to avoid high complexity of the graph isomorphism problem, which
resulted in an incomplete set of characteristic subgraphs.

In SUBDUE[19], the authors propose an adaptive path index for XML
data(APEX). APEX does not keep all paths starting from the root and utilizes
frequently used paths to improve the query performance. APEX also has a nice
property that it can be updated incrementally according to the changes of query
workload. The authors also conclude that the experimental results of their algorithm
with synthetic and real-life data sets clearly confirm that APEX improves query
processing cost typically 2 to 54 times better than the traditional indexes, with the
performance gap increasing with irregularity of XML data. APEX can be
incrementally updated in order to minimize the overhead of construction whenever
the query workload changes. To support efficient query processing, the authors
introduce two structures: the graph structure \textit{GAPEX} and the hash tree \textit{HAPEX}.
\textit{GAPEX} represents the structural summary of XML data with extents. \textit{HAPEX} keeps
the information for frequently used paths and their corresponding nodes in \textit{GAPEX}.
Given a query, then the authors use \textit{HAPEX} to locate the nodes of \textit{GAPEX} that have
extents required to evaluate the query.

Dehaspe and Toivonen proposed an ILP-based algorithm, WARMR, enabling
a complete search for frequent subgraphs from graph data in 1998. [30]. WARMR
extends APRIORI [1] to discover frequent queries in data. WARMR looks at a level
of the query lattice at a time, starting from the most general patterns, and iterates
between candidate generation and candidate evaluation phases. In candidate generation, the lattice structure is used for pruning nonfrequent patterns from the next level; in the candidate evaluation phase, frequencies of candidates are computed with respect to the database. Pruning is based on monotonicity of generality with respect to frequency: if a pattern is not frequent then none of its specializations is frequent. So while generating candidates for the next level, all the patterns that are specializations of infrequent patterns can be pruned.

Subsequent work done by Nijssen and Kok proposed a faster algorithm, FARMER [75]. In the above work, the authors present some new ideas to turn one important intermediate step in the process of discovering frequent item sets in a more efficient way. Using an implementation that the authors coined FARMER, they show that indeed a speed-up is obtained and that, using these ideas, the performance is much more comparable to original association rule algorithms. They propose to obtain a gain in efficiency by tackling two properties of the WARMR algorithm: i) while still using the first order logic notation, the authors remove the need for PROLOG and ii) by using a more sophisticated data structure borrowed from an implementation of APRIORI, this algorithm does not depend on a time consuming test for equivalence. This algorithm has some resemblance with the algorithm that was developed in [Blockeel et al., 2000]. That algorithm however did not tackle one of the most time consuming steps of WARMR i.e., a test for equivalence under $\theta$-subsumption. FARMER algorithm pays special attention to this step and offers an alternative solution. Under some restrictions the authors shown that the algorithm is equivalent to WARMR. The authors also conclude that their algorithm has considerable speed-up compared to WARMR.

Inokuchi et al. proposed an approach called AGM to combine Apriori algorithm and mathematical graph theory in 2000 [52]. The authors propose a novel approach efficiently mine the association rules among the frequently appearing substructures in a given graph data set. In their work, a graph transaction is represented by an adjacency matrix, and the frequent patterns appearing in the matrices are mined through the extended algorithm of the basket analysis. They use the artificial simulation data and the carcinogenesis data of Oxford University and
NTP for evaluating the performance of the algorithm. In their process, they stated that they have obtained the largest graphs of the chemical compound discovered with the size of 13 atoms where as the approach of ILP in conjunction with a level wise search proposed by Dehaspe et al. could mine the substructure consisting of 6 predicates at maximum equivalent to the size of a molecule consisting of only 3 atoms or so [29]. This fact shows the practical efficiency of AGM for real world problems.

De Raedt and Kramer proposed the version space based approach called MolFea in 2001 to find characteristic paths from the graph data [31]. The algorithm is of a tight integration of Mitchell’s version space algorithm with Agrawal et al.’s Apriori algorithm. The algorithm can be used to generate patterns that satisfy a variety of constraints on data. Constraints that can be imposed on patterns include the generality relation among patterns and imposing a minimum or a maximum frequency on data sets of interest. The theoretical framework of this algorithm is applied to an important application in chemo-informatics i.e. that of finding fragments of interest within a given set of compounds. Fragments are linearly connected substructures of compounds. Implementations as well as preliminary experiments within the application are presented in this algorithm.

There are many graph-related algorithms developed in the fields of chemical informatics [8] [9] [100] [103], computer vision [109], video indexing [91], and text retrieval [64]. Frequent substructures are very basic one among the various graph patterns that can be discovered in a set of graphs.

2.5.3 Types of Subgraph Mining

There are different ways of defining the types of subgraphs that would be discovered. These definitions are based on three aspects.

The first gives the topology of the subgraph itself and deals with patterns of specified topology. Examples include arbitrary subgraphs, induced subgraphs, trees, and paths.
The second dimension shows their relation within the context of the frequent pattern lattice and contains subgraphs that are frequent, maximal, or closed.

Finally, the third dimension points whether there is a relation between the various embeddings of the discovered patterns and contain subgraphs which have arbitrary overlaps, partial overlaps, and being vertex- and/or edge-disjoint.

Frequent pattern discovery algorithms can be classified into two categories depending on their completeness. Complete algorithms find all frequent patterns that satisfy a given specification (typically the minimum support threshold). On the other hand, heuristic algorithms return only a subset of all frequent patterns. In heuristic algorithms, the interesting point is that depending on the problem formulation, complete algorithms may simply become unfeasible. For those problems, heuristic algorithms can be useful practical solutions.

The frequent subgraph mining usually consists of two steps. In the first step, it generates frequent substructure candidates while the frequency of each candidate is checked in the second step. The second step involves subgraph isomorphism that is NP-complete.

In the following sections, two major types of subgraph mining are discussed.

2.5.3.1 Apriori Algorithm

The first frequent itemset mining algorithm was denoted as AIS [1]. In later, the algorithm was improved and called Apriori. The main improvement was developed the monotonicity property of the support of sets [5] [97]. After the improvement, the monotonicity further got better shape by Mannila et al [68] and Agarwal et al [2]. The Apriori algorithm is based on candidate generation approach. The Apriori algorithm is given in figure 2.2 [87].
Input: Datasets $D$, MinSupp $\alpha$

Output: $F(D, \alpha)$

Step 1: $C_1 := \{ \{i\} \mid i \in I\}$

Step 2: $k := 1$

Step 3: While $C_k \neq \{\}$ do

Step 4: For all transactions $(TID, I) \in D$ do

Step 5: For all candidate sets $X \in C_k$ do

Step 6: If $X \subseteq I$ then

Step 7: Increment $X.support$ by 1

Step 8: End if

Step 9: End for

Step 10: End for

Step 11: $F_k := \{ X \in C_k \mid X.support \geq \alpha \}$

Step 12: $C_{k+1} := \{\}$

Step 13: For all $X, Y \in F_k$ such that $X[i] = Y[i]$

Step 14: For $1 \leq i \leq k-1$, and $X[k] < Y[k]$ do

Step 15: $I := X \cup \{Y[k]\}$

Step 16: If $\forall J \subset I; |J| = k : J \in F_k$ then

Step 17: Add $I$ to $C_{k+1}$

Step 18: End if

Step 19: End for

Step 20: Increment $k$ by 1

Step 21: End while

---

**Figure 2.2** Apriori Algorithm

The number of inputs for the Apriori algorithm is two and they are: set of transactions ($D$) and minimum support ($\alpha$). In each transaction, the items are sorted in their lexicographic order. It was denoted as $X[i]$ that means, the $i$th item in $X$. $F_k$ denoted the frequent $k$-sets. ie. a set $X$ is the $k$-set $\{X[1] \ldots \ldots X[k]\}$. The Apriori used
a breadth-first search technique to search by repeatedly generating and counting candidate sets. Basically, the Apriori algorithm is using monotonicity property. That is a set, is candidate if all of its subsets are counted and frequent. In each iteration, the collection $C_{k+1}$ of candidate sets of size $k+1$ is generated, starting with $k = 0$.

The Apriori algorithm is done in two steps as follows: First one is join step. This step is used to generate the candidate set and second one is the prune step. This step is mainly for finding the frequent set from candidate set. Initially, the set $C_1$ consists of all items in $I$ (Step 1). At level $k$, all candidate sets of size $k+1$ are generated.

First, in the join step, the union $X \cup Y$ of sets $X,Y \in F_k$ is generated if they have the same $k-1$-prefix (Step 10–11). In the prune step, $X \cup Y$ is inserted into $C_{k+1}$ only if all of its $k$-subsets are frequent. It must occur in $F_k$ (Step 12–13). To count the supports of all candidate $k$-sets, the database, which remains on secondary storage in the horizontal layout, is scanned one transaction at a time, and the supports of all candidate sets that are included in that transaction are incremented (Step 4–7). All sets that turn out to be frequent are inserted into $F_k$ (Step 8). If the number of candidate sets is too large to remain into main memory, the algorithm can be easily modified as follows.

The candidate generation procedure stops and the supports of all generated candidates are counted. In the next iteration, instead of generating candidate sets of size $k+2$, the remaining candidate $k+1$-sets are generated and counted repeatedly until all frequent sets of size $k+1$ are generated and counted.

### 2.5.3.2 FP-Growth Pattern Mining Algorithm

Han et al [43] devised an FP-growth method that mines the complete set of frequent itemsets without candidate generation. It is employed in a divide-and-conquer manner. In first scan, the database derives a list of frequent items in which items are ordered by frequency descending order. The database is compressed into a frequent pattern tree (FP-tree) using frequency descending order list. The FP-tree is mined by starting from each frequent length-1 pattern, constructing its conditional
pattern base, then constructing its conditional FP-tree, and performing mining recursively on such a tree. The pattern growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree. The FP-growth algorithm transformed the problem of finding long frequent patterns to searching for shorter ones recursively and then concatenating the suffix. It uses the least frequent items as a suffix, offering good selectivity. The performance studies of FP-Growth exhibit that the method significantly reduces search time.

### 2.5.3.3 Other Related Issues

In the context of frequent subgraph mining, Apriori-based algorithms have two kinds of considerable overheads:

(1) joining two size-$k$ frequent graphs (or other structures like paths in [106]) to generate size-$(k + 1)$ graph candidates, and

(2) checking the frequency of these candidates separately.

These overheads constitute the performance bottleneck of Apriori-based algorithms. To avoid the overheads incurred in Apriori-based algorithms, non-Apriori based algorithms such as gSpan [110], MoFa [11], FFSM [46], SPIN [85], and Gaston [77] have been developed recently.

These algorithms are inspired by PrefixSpan [80], TreeMinerV [118], and FREQT [7] at mining sequences and trees, respectively.

All of these algorithms adopt the pattern growth methodology [39], which intends to extend patterns from a single pattern directly.

Among these, Molecular Fragment Miner (MoFa) [11] is extensively used in bioinformatics.
Graph–Based Substructure Pattern Mining (gSpan) [110] uses canonical labeling and the right-most extension technique.

GrAph/Sequence/Tree extraction (Gaston) [77] begins by searching for frequent-path, then frequent free trees and finally cyclic graphs.

Yan et al. presented another new method: Closed graph pattern mining (CloseGraph) [111]. The success of this method lies in the development of the novel concepts of equivalent occurrence and early termination, which help CloseGraph prune the search space substantially with small additional cost.

The Apriori-based approach uses breadth-first search (BFS) strategy because of its level-wise candidate generation. To determine whether a size-\((k + 1)\) graph is frequent; it has to check all of its corresponding size-\(k\) subgraphs to obtain an upper bound of its frequency. Thus, before mining any size-\((k + 1)\) subgraph, the Apriori-based approach usually has to complete the mining of size-\(k\) subgraphs. Therefore, BFS is necessary in the Apriori-like approach. In contrast, the pattern growth approach is more flexible on the search method. Both breadth-first search and depth-first search (DFS) can work.

All the existing subgraph mining algorithms were packed into some system of approaches.

2.6 Applications

The applications of frequent subgraphs are characterizing graph sets, discriminating different groups of graphs, classifying and clustering graphs, and building graph indices. Apart from these applications, subgraph mining technique can also be applied to bioinformatics, cheminformatics, network topology, web search engines, and other searching techniques, etc., some examples are, Borgelt and Berthold [11] illustrated the discovery of active chemical structures in a dataset that screened for the human immunodeficiency virus (HIV) by contrasting the support of frequent graphs between different classes. Deshpande et al. [30] used frequent
structures as features in the classification of chemical compounds. Huan et al. [47] successfully applied the frequent graph mining technique to study protein structural families. Frequent graph patterns were also used as indexing features by Yan et al. [111] to perform fast graph search. Their method outperforms the traditional path-based indexing approach significantly. Koyuturk et al. [59] proposed a method to detect frequent subgraphs in biological networks.

Moreover, as subgraph mining techniques have been increasingly applied to nontraditional domains, there is a need to develop efficient and general-purpose frequent pattern discovery algorithms that are capable of capturing the spatial, topological, geometric, and/or relational nature of the datasets that characterize these domains. In many application domains, there exist datasets that possess inherently structural or relational characteristics, which are suitable for graph-based representations, and can greatly benefit from GDM algorithms [e.g., network topology, very large scale integration (VLSI) circuit design, protein–protein interactions, biological pathways, Web graph, etc.

2.7 Ranking in Graph

Ranking is a crucial part of information retrieval. It is able to compute sorted score when given documents as graphs. The score may represent (degrees of relevance, preference and importance etc.). The ranking measure evaluates feature selection, analyzing the multiples of relevance. The ranking is also used to compute the importance of scores of features.

In ranking, order of relationships among objects can be easily explained and the goal is to learn from these examples a real-valued ranking function that induces a ranking or ordering over the object space. When the data is represented as a graph then the problem of ranking function is considered as vertices corresponds to objects and edges gives similarities between objects.

Frequent and periodic pattern mining methods can generate a very large number of small patterns at all but the highest minimum support values. On the other hand, potentially significant patterns could occur at lower supports and be lost in the
sheer number of such patterns. Thus, the extracted combinatorial periodic patterns, statistical or heuristic ranking schemes for mined patterns are useful for practical applications [13, 46, 112]. Ranking methods of subgraphs depends on the graph data set. Or in other words the ranking methods are inter-related with the application of subgraphs over the entire set of graph.

2.8 Related Works

The problem of graph query processing has been addressed in various fields since it is a critical problem for many applications. In content-based image retrieval, Petrakis and Faloutsos [81] represent each graph as a vector of features and index graphs in high dimensional space using R-trees. In [93], Shokoufandeh et al. represent and index graphs by a signature computed from the eigenvalues of adjacency matrix. Instead of casting a graph to a vector form, Berretti et al. [10] proposes a metric indexing scheme which organizes graphs hierarchically according to their mutual distances. In 3D protein structure search, algorithms using hierarchical alignments on secondary structure elements, e.g., Madej et al. [66], or geometric hashing [108], have already been developed for a decade. There are other literatures related to graph retrieval in these fields, which cannot be exhausted. In short, these systems are designed for other graph retrieval tasks, such as exact or similar whole graph retrieval [81] [93] [10] and 3D geometric graph retrieval [66] [108]. They are either inapplicable or inefficient to the problem discussed in this paper.

In semistructured/XML databases, query languages built on path expressions become popular. Efficient index tech- niques for path expression are initially shown in DataGuide [31] and 1-index [72]. A(k)-index [58] further proposes k-bisimilarity to exploit local similarity existing in semistructured databases. APEX [18] and D(k)-index [15] consider the adaptivity of index structure to fit the query load. Index Fabric [23] represents every path in a tree as a string and stores it in a Patricia trie. For more complicated graph queries, Shasha et al. [90] extends the path-based technique to do full scale graph retrieval, which is also used in Daylight system [54]. Srinivasa et al. [99] builds the index based on multiple vector spaces with different abstract levels of graphs. However, no algorithm is considered to index graphs using frequent
structures, which is the emphasis of this study. Washio and Motoda [107] has a
general introduction on the recent progress of GDM. In frequent graph mining,
Inokuchi et al. [52], Karamochi and Karypis [62], and Vanetik et al. [106] propose
Apriori-based algorithms to discover frequent subgraphs. Yan and Han [111] and
Borgelt and Berthold [11] apply the pattern-growth approach to directly generate
frequent subgraphs.
2.9 Summary

Initially this chapter started the discussion with association rule mining and its sub problems. The discussion concludes that the problem of frequent itemset construction is the major and cost effective task of association analysis. This problem was deeply discussed with their advantages and disadvantages. The Apriori algorithm and the FP-Growth algorithm are shown in detail in this chapter. The rule construction procedure is argued in effective manner. Measures are important role in data mining and association analysis. This chapter also discusses the issues in graph mining and the conversion of graph database. This chapter also clarifies the usage of measures, types and some survey regarding measures in association analysis. The association analysis is applied to various domains including graph also. The subgraph mining with candidate generation and without candidate generation approach is also discussed with some existing works. A deep literature review is also done about subgraph mining and ranking.