Chapter - I

Introduction
CHAPTER 1
INTRODUCTION

In this chapter the fundamental concepts related to graph mining such as techniques, types, propagations, kernel structures, matching approaches are described.

1.1 Graph Fundamentals

As a general data structure, graphs have become increasingly important in modeling sophisticated structures and their interactions, with broad applications including chemical informatics, bio informatics, computer vision, video indexing, text retrieval and Web analysis etc., Mining frequent sub graph patterns for further characterization, discrimination, classification, and cluster analysis becomes an important task. Moreover, graphs that link many nodes together may form different kinds of networks, such as telecommunication networks, computer networks, biological networks, Web and social community networks. Since such networks have been studied extensively in the context of social networks, their analysis has often been referred to as social network analysis. Furthermore, in a relational database, objects are semantically linked across multiple relations. Mining in a relational database often requires mining across multiple interconnected relations, which is similar to mining in connected graphs or networks. Such kind of mining across data relations is considered multi relational data mining. In this chapter, study of knowledge discovery in such interconnected and complex structured data is done. The core of the problem is mining frequent sub graph patterns over a collection of graphs, which includes concepts and methods for social network analysis that examines methods for multi relational data mining, including both cross-relational classification and user-guided multi relational cluster analysis[21].
**Graph Definition:** A graph $G$ is defined as follows,

\[ G = (V, E) \]

$V$ : finite set of nodes.

$E$ : denotes a set of edges.

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$

Graph $G_1$ is a subgraph of $G_2$,

If $V_1$ is a subset of $V_2$ and $E_1$ is a subset of $E_2$

**Figure 1.1** Graph structure

Graphs become increasingly important in modeling complicated structures, such as circuits, images, chemical compounds, protein structures, biological networks, social networks, the Web, workflows, and XML documents. Many graph search algorithms have been developed in chemical informatics, computer vision, video indexing, and text retrieval. With the increasing demand on the analysis of large amounts of structured data, graph mining has become an active and important theme in data mining. Among the various kinds of graph patterns, frequent substructures [22] are the very basic patterns that can be discovered in a collection of graphs. They are useful for characterizing graph sets, discriminating different groups of graphs, classifying and clustering graphs, building graph indices, and facilitating similarity search in graph databases. Recent studies have developed several graph mining methods and applied
them to the discovery of interesting patterns in various applications. For example, there have been reports on the discovery of active chemical structures in cancer-screening datasets by contrasting the support of frequent graphs between different classes. There have been studies on the use of frequent structures as features to classify chemical compounds, on the frequent graph mining technique to study protein structural families, on the detection of considerably large frequent sub pathways in metabolic networks, and on the use of frequent graph patterns for graph indexing and similarity search in graph databases. Although graph mining may include mining frequent sub graph patterns, graph classification, clustering, and other analysis tasks, this research focuses on mining frequent sub graphs that look at various methods, their extensions, and applications.

1.2 Methods for Mining Graphs

Before presenting graph mining methods, it is necessary to first introduce some preliminary concepts relating to frequent graph mining.

The vertex set of a graph $g$ is represented by $V(g)$ and the edge set is represented by $E(g)$. A label function, maps a vertex or an edge to a label. A graph $g$ is a subgraph of another graph $g_0$, if there is a subgraph isomorphism from $g$ to $g_0$. Given a labeled graph data set, $D = \{G_1; G_2; G_n\}$, this research define support ($g$) (or frequency $f(g)$) as the percentage (or number) of graphs in $D$ where $g$ is a subgraph. A frequent graph is a graph whose support is not less than a minimum support threshold, $\text{min sup}$.
1.2.1 Apriori – based Approach

Apriori-based frequent substructure mining algorithms share similar characteristics with Apriori-based frequent item set mining algorithms. The search for frequent graphs starts with graphs of small “size,” and proceeds in a bottom-up manner by generating candidates having an extra vertex, edge, or path. The definition of graph size depends on the algorithm used. The general framework of Apriori-based methods for frequent substructure mining is related to this algorithm as Apriori Graph.

Apriori Graph adopts a level-wise mining methodology. At each iteration, the size of newly discovered frequent substructures is increased by one. These new substructures are first generated by joining two similar but slightly different frequent sub graphs that were discovered in the previous call to Apriori Graph. This candidate generation procedure is outlined on line 4. The frequency of the newly formed graphs is then checked. Those found to be frequent are used to generate larger candidates in the next round.

The main design complexity of Apriori-based substructure mining algorithms is the candidate generation step [49]. The candidate generation in frequent item set mining is straightforward. For example, suppose we have two frequent item sets of size-3: (abc) and (bcd). The frequent item set candidate of size-4 generated from them is simply (abcd), derived from a join. However, the candidate generation problem in frequent substructure mining is harder than that in frequent item set mining, because there are many ways to join two substructures.
Algorithm : Apriori-based frequent substructure mining.

Input : D, a graph data set; min sup, the minimum support threshold.

Output : $S_k$, the frequent substructure set.

Method : $S1$ frequent single-elements in the data set;

Call Apriori Graph (D, min sup, S1);

1: $S_{k+1} \leftarrow \emptyset$;
2: for each frequent $g_i \in S_k$ do
3:   for each frequent $g_j \in S_k$ do
4:     for each size $(k+1)$ graph $g$ formed by the merge of $g_i$ and $g_j$ do
5:       if $g$ is frequent in $D$ and $g \notin S_{k+1}$ then
6:         insert $g$ to $S_{k+1}$;
7:       if $S_{k+1} \neq \emptyset$ then
8:         call Apriori($D$, min_support, $S_{k+1}$);
9: return;

Figure 1.2 Graph substructure mining

1.2.2 Pattern Growth Approach

A graph G is extended by adding new edge $e$.

Edge $e$ may or may not introduce a new node to G.
1.2.3 Graph Based Decision Tree

A Decision Tree represents the following

- Each branch corresponds to attribute value
- Each leaf node assigns a classification

Figure 1.3 Pattern growth approach

Figure 1.4 Graph Based Decision Tree
The conversion of Data Mining to Graph Mining includes the following,

a. Data Mining  
b. Classification  
c. Clustering  
d. Association rule learning  
e. Graph Mining

The powerful way of representing data and the output is to express it as graphs.

### 1.2.4 Graph Mining Domains

**a. Internet Movie Database** contains the following components, Movie Recommendation, Community Detection, Prediction in opening weekend etc.

![Figure 1.5 Movie Data Mining](image-url)
b. **Web Data Mining** contains the following components Web Content, Topic Prediction, Web Structure Mining, Community Mining, and Web Usage Mining, Website Roadmap etc.

![Web Data Mining Diagram](image)

**Figure 1.6** Web Data Mining

c. **Social Networks Analysis** represents the Relationships and flows between people and computer technologies such as Email, Blogs, Social Networking Software like Orkut, Face Book etc.

![Social Network Diagram](image)

**Figure 1.7** Social Network
d. Bio-Informatics includes the following components,

Protein-Protein Interaction (PPI) Network

**Nodes:** Proteins

**Edges:** Interaction among the Proteins, Open Problems, Function Prediction, Drug Discovery etc.

![Protein-Protein Interaction Network](image)

**Figure 1.8** Protein data mining

### 1.3 Graph Classification Techniques

Graphs are powerful data structures for organizing vast quantities of data. Mining for graph patterns has steadily grown as a topic of interest and has found applications in a wide range of fields, including bioinformatics and chemo informatics [9, 15, and 1], database indexing [8], and web information management.

For example, proteins, whose structures can be represented by graphs, can be classified into two classes: those which perform a certain function and those which do not. Similarly, chemical compounds can be classified into two classes: those which are active and those which are not. Note that in these applications, the positive and negative classes may not necessarily contain comparable number of graphs. It is also possible
that graphs in the negative class may be much more diverse than graphs in the positive class.

Given a training set that contains both positive graphs and negative graphs, the objective of graph classification is to build a prediction model that separates these two classes. Early work [10, 7, and 2] in graph classification took a straight forward two-step approach, which first generates a set of sub graph patterns and then employs a generic classification model in the feature space constructed by mapping the occurrence of a graph pattern to a feature. A major shortcoming of this approach is the decoupling of the sub graph pattern mining and classifier construction. The number of sub graph patterns generated in the first step is usually very large and includes many patterns which may not correspond to features of high classification power. This often leads to prolonged running time and poor classification accuracy.

To overcome this drawback, recent approaches in graph classification integrate sub graph pattern mining and classifier construction. Several boosting algorithms have been proposed which look for discriminative sub graph patterns without examining all possible sub graphs [14, 16, and 18]. These algorithms mine patterns repeatedly in multiple iterations. During each subsequent iteration, misclassified graphs are given higher weights.

However, this approach may take much iteration to reach high classification accuracy, resulting in long execution time. The LEAP algorithm [22] takes a novel divergence from this standard and introduces two concepts: (i) structural leap search and (ii) frequency-descending mining. This method is faster than previous methods because it is able to quickly locate patterns that individually have high discrimination power, without exploring the whole pattern space. Furthermore, it gives a much smaller
pattern set than traditional graph mining algorithms, which facilitates classification model training.

However, this method focuses on the discriminative power of individual patterns and hence does not work well in two scenarios. (1) When no individual pattern has high discrimination power, a group of patterns may jointly have higher discrimination power (see example in Section 3.2). LEAP is not designed for evaluating joint discrimination power of multiple patterns that have low individual discrimination powers. LEAP is therefore apt to fail in identifying these patterns.

(2) Furthermore, the top-k patterns found by LEAP may not necessarily compose the best classifier, especially when these k patterns share most of their supporting graphs. Therefore, LEAP is not a suitable stand-alone graph classification algorithm. To construct a good classification model, we need to invoke LEAP multiple times. We adjust the weight of each graph after each invocation of LEAP so that the next invocation will identify discriminative patterns that are complementary to the ones returned by earlier invocations.

The union of these patterns can then be used as features to train a classifier. Another algorithm gPLS [19] adapts the powerful mathematical tool of PLS (Partial Least Squares) regression to graph mining to collect informative subgraph patterns and build a classifier directly with fewer iterations than typical boosting methods. It creates latent variables involving response variables, thus leading to better predictions. However, these latent variables have the known disadvantage of poor interpretability. CORK [20] is a subgraph-based algorithm for binary graph classification which attempts to discover frequent subgraphs that remove correspondence between graphs in the positive and negative classes.
Given a set of sub graphs, the number of correspondences is the total number of pairs of graphs that cannot be discriminated by these sub graphs. The number of correspondences is sub-modular and can usually achieve good results. However, it is not perfect since sub graphs of vastly different discrimination power may have the same number of correspondences (see example in Section 2).

Therefore, we propose to investigate the discrimination power of co-occurrence of sub graph patterns and design a method to mine co-occurrence rules that can be readily used to classify graphs into positive and negative classes. These co-occurrence rules are able to capture complex graph features that offer high discrimination power. We propose an algorithm, COM (Co-Occurrence rule Miner), which employs an efficient pattern exploration order to locate sub graph patterns whose co-occurrence is indicative of graph classification. These co-occurrence rules offer higher classification accuracy as well as better interpretability than previous approaches.

“How can we mine social networks?” [50] Traditional methods of machine learning and data mining, taking, as input, a random sample of homogenous objects from a single relation, may not be appropriate here[51]. The data comprising social networks tend to be heterogeneous, multi relational, and semi-structured. As a result, a new field of research has emerged called link mining. Link mining is a confluence of research in social networks, link analysis, hypertext and Web mining, graph mining, relational learning, and inductive logic programming. It embodies descriptive and predictive modeling. By considering links (the relationships between objects), more information is made available to the mining process. This brings about several new tasks with examples from various domains as follows:
(i). **Link-based object classification:** In traditional classification methods [23], objects are classified based on the attributes that describe them. Link-based classification predicts the category of an object based not only on its attributes, but also on its links, and on the attributes of linked objects. Web page classification [53] is a well-recognized example of link-based classification. It predicts the category of a Web page based on word occurrence (words that occur on the page) and anchor text (the hyperlink words, that is, the words you click on when you click on a link), both of which serve as attributes. In addition, classification is based on links between pages and other attributes of the pages and links. In the bibliography domain, objects include papers, authors, institutions, journals, and conferences [56].

A classification task is to predict the topic of a paper based on word occurrence, citations (other papers that cite the paper), and co citations (other papers that are cited within the paper), where the citations act as links. An example from epidemiology is the task of predicting the disease type of a patient based on characteristics (e.g., symptoms) of the patient, and on characteristics of other people with whom the patient has been in contact. (These other people are referred to as the patients’ contacts.)

(ii). **Object Type Prediction:** This predicts the type of an object, based on its attributes and its links, and on the attributes of objects linked to it. In the bibliographic domain, which may want to predict the venue type of a publication as conference, journal, or workshop? In the communication domain, a similar task is to predict whether a communication contact is by e-mail, phone call, or mail.

(iii). **Link type prediction:** This predicts the type or purpose of a link, based on properties of the objects involved. Given epidemiological data, for instance, which may try to predict whether two people who know each other are family members,
coworkers, or acquaintances. In another example, which may want to predict whether there is an advisor-advisee relationship between two coauthors? Given Web page data, which can try to predict whether a link on a page is an advertising link or a navigational link?

(iv). **Predicting link existence:** Unlike link type prediction, to know a connection exists between two objects and to predict its type, instead of predicting whether a link exists between two objects. Examples include predicting whether there will be a link between two Web pages, and whether a paper will cite another paper. In epidemiology, one can try to predict with whom a patient came in contact.

(v). **Link cardinality estimation:** There are two forms of link cardinality estimation. First, one may predict the number of links to an object. This is useful, for instance, in predicting the authoritativeness of a Web page based on the number of links to it (in-links). Similarly, the number of out-links can be used to identify Web pages that act as hubs, where a hub is one or a set of Web pages that point to many authoritative pages of the same topic.

In the bibliographic domain, the number of citations in a paper may indicate the impact of the paper—the more citations the paper has, the more influential it is likely to be. In epidemiology, predicting the number of links between a patient and his or her contacts is an indication of the potential for disease transmission. A more difficult form of link cardinality estimation predicts the number of objects reached along a path from an object. This is important in estimating the number of objects that will be returned by a query. In the Web page domain, one may predict the number of pages that would be retrieved by crawling a site (where crawling refers to a methodological, automated search through the Web, mainly to create a copy of all of the visited pages for later
processing by a search engine). Regarding citations, one can also use link cardinality estimation to predict the number of citations of a specific author in a given journal.

(vi). **Object reconciliation:** In object reconciliation, the task is to predict whether two objects are, in fact, the same, based on their attributes and links. This task is common in information extraction, duplication elimination, object consolidation, and citation matching, and is also known as record linkage or identity uncertainty. Examples include predicting whether two websites are mirrors of each other, whether two citations actually refer to the same paper, and whether two apparent disease strains are really the same.

(vii). **Group detection:** Group detection is a clustering task. It predicts when a set of objects belong to the same group or cluster, based on their attributes as well as their link structure. An area of application is the identification of Web communities, where a Web community is a collection of Web pages that focus on a particular theme or topic. A similar example in the bibliographic domain is the identification of research communities.

(viii). **Subgraph detection:** Subgraph identification finds characteristic subgraphs within networks. This is a form of a graph search. An example from biology is the discovery of subgraphs corresponding to protein structures. In chemistry, one can search for subgraphs representing chemical substructures.

(ix). **Metadata Mining:** Metadata are data about data. Metadata provide semi-structured data about unstructured data, ranging from text and Web data to multimedia databases. It is useful for data integration tasks in many domains. Metadata mining can be used for schema mapping (where, say, the attribute customer id from one database is mapped to customer number from another database because they both refer to the same
entity), schema discovery, which generates schema from semi-structured data and schema reformulation, which refines the schema based on the mined metadata. Examples include matching two bibliographic sources, discovering schema from unstructured or semi-structured data on the Web, and mapping between two medical ontologies.

The exploitation of link information between objects brings on additional tasks for link mining in comparison with traditional mining approaches. The implementation of these tasks, however, invokes many challenges. These challenges are:

(i). Logical versus statistical dependencies: Two types of dependencies reside in the graph-link structures (representing the logical relationship between objects) and probabilistic dependencies (representing statistical relationships, such as correlation between attributes of objects where, typically, such objects are logically related). The coherent handling of these dependencies is also a challenge for multirelational data mining, where the data to be mined exist in multiple tables. One must search over the different possible logical relationships between objects, in addition to the standard search over probabilistic dependencies between attributes. This takes a huge search space, which further complicates finding a plausible mathematical model. Methods developed in inductive logic programming may be applied here, which focus on search over logical relationships.

(ii). Feature construction: In link-based classification, consider the attributes of an object as well as the attributes of objects linked to it. In addition, the links may also have attributes. The goal of feature construction is to construct a single feature representing these attributes. This can involve feature selection and feature aggregation. In feature selection, only the most discriminating features are included. Feature
aggregation takes a multi set of values over the set of related objects and returns a summary of it. This summary may be, for instance, the mode (most frequently occurring value); the mean value of the set (if the values are numerical); or the median or “middle” value (if the values are ordered). However, in practice, this method is not always appropriate.

(iii). **Instances versus classes**: This alludes to whether the model refers explicitly to individuals or to classes (generic categories) of individuals. An advantage of the former model is that it may be used to connect particular individuals with high probability. An advantage of the latter model is that it may be used to generalize to new situations, with different individuals.

(iv). **Collective classification and collective consolidation**: Consider training a model for classification, based on a set of class-labeled objects. Traditional classification methods consider only the attributes of the objects. After training, suppose for a given new set of unlabeled objects. Use of the model to infer the class labels for the new objects is complicated due to possible correlations between objects—the labels of linked objects may be correlated. Classification should therefore involve an additional iterative step that updates (or consolidates) the class label of each object based on the labels of objects linked to it. In this sense, classification is done collectively rather than independently.

(v). **Effective use of labeled and unlabeled data**: A recent strategy in learning is to incorporate a mix of both labeled and unlabeled data. Unlabeled data can help infer the object attribute distribution. Links between unlabeled (test) data allow us to use attributes of linked objects. Links between labeled (training) data and unlabeled (test) data induce dependencies that can help make more accurate inferences.
(vi). **Link prediction:** A challenge in link prediction is that the prior probability of a particular link between objects is typically extremely low. Approaches to link prediction have been proposed based on a number of measures for analyzing the proximity of nodes in a network. Probabilistic models have been proposed as well [44]. For large data sets, it may be more effective to model links at a higher level.

(vii). **Closed versus open world assumption:** Most traditional approaches assume that to know all the potential entities in the domain. This “closed world” assumption is unrealistic in real-world applications. Work in this area includes the introduction of a language for specifying probability distributions over relational structures that involve a varying set of objects.

(viii). **Community mining from multirelational networks:** Typical work on social network analysis includes the discovery of groups of objects that share similar properties. This is known as community mining. Web page linkage is an example, where a discovered community may be a set of Web pages on a particular topic. Most algorithms for community mining assume that there is only one social network [47], representing a relatively homogenous relationship. In reality, there exist multiple, heterogeneous social networks, representing various relationships. A new challenge is the mining of hidden communities on such heterogeneous social networks, which is also known as community mining on multirelational social networks. These challenges will continue to stimulate much research in link mining [48].

1.4 **Label Propagation Techniques**

Label propagation aims to use a graph $G = (V, E, W)$ to propagate topic labels from labeled vertices to un-labeled vertices. Each vertex $v_i$ can have multiple topic labels, i.e. a document can belong to more than one category, and each label is
considered in-dependently of the other labels assigned to a vertex. The labels assigned to the set of labeled vertices \( Y_l = \{y_1, y_l\} \) are used to estimate the labels \( Y_u = \{y_{l+1}, y_{l+u}\} \) on the unlabelled set.

Carreira-Perpinan et al. (Carreira-Perpinan and Zemel, 2004) suggest constructing graphs from ensembles of minimum spanning trees (MST) as part of their label propagation algorithm, with their two methods Perturbed MSTs (PMSTs) and Disjoint MSTs (DMSTs), having a complexity of approximately \( O(T N^2 \log N) \) and \( O(N^2(\log N + t)) \) respectively, where \( N \) is the number of vertices, \( T \) is the number of MSTs ensemble in PMSTs, and \( t \) is the number of MSTs used in DMSTs, typically \( t \ll N^2 \). However, to the best of the authors' knowledge, no studies have performed experiments on constructed graphs with more than several thousand vertices, with the exception of Herbster et al. (Herbster et al., 2009) who build a shortest path tree (SPT) and MST for a graph with 400,000 vertices from Web pages. Herbster et al. (Herbster et al., 2009) also noted that constructing their MST and SPT trees using Prim and Dijkstra algorithms (Cormen et al., 1990) respectively takes \( O(N \log N + |E|) \) time, with the general case of a non-sparse graph having a time complexity of \( O(N^2) \).

A natural adaptation of the Label Propagation (LP) algorithm (Zhu et al., 2003), as our algorithm for the label propagation step due to its efficiency and simplicity. OMV is based closely upon the locality assumption, that vertices that are close to one another, with respect to a distance or measure and should have similar labels. Each vertex is sequentially labeled as the unweighted majority vote on all labels from the neighboring vertices. The time complexity for OMV is \( O(|E|) \), a no-table reduction from the \( O(kN^2) \) required for LP algorithm, where \( k \) is the neighbors per vertex. The complexity being dependent on the number of edges in the graph further benefits from
the apriori limit we impose upon the maximum edges per vertex, ensuring that $|E| = bN$ for some maximum edge limit $b$.

**Technique:**

The basic idea is that we start with a group of users that we have some information about the categories they are interested in. Following the weights in the social network, we propagate the label probabilities from the user seed node (the ones we have label information about) into the general social network population. After several iterations, the algorithm converges and the output is labeled for the unknown nodes.

Here is a pseudo code for the algorithm:

The label propagation algorithm is as follows:

1. Propagate $Y \leftarrow TY$
2. Row-normalize $Y$.
3. Clamp the labeled data. Repeat from step 1 unit $Y$ converges.

Where $Y$ is the matrix with probabilities for each label, $T$ is the original adjacency graph (where weights are normalized first to one). Clamping means that for the labeled data the weights are fixed to be the input probabilities.

Label propagation was first proposed as an efficient method for learning missing labels for graph data in a semi-supervised setting [20]. In such a setting, unlabeled nodes iteratively adopt the label of the plurality of their neighbors until convergence, making it possible to infer a broad range of traits that are fundamentally assertive along the edges of a graph.
In a context very similar to graph partitioning, label propagation has been found to be a very efficient technique for network community detection [16], the challenge of finding naturally dense network clusters in an unlabeled graph [7]. In this context, each node of a graph is initialized with an individual label, and label propagation is iterated where nodes again update their labels to the plurality of their neighbor's labels.

Network community detection and graph partitioning are very similar challenges, with two key differences. First, community detection algorithms need not and should not require apriori specification of the number or size of graph communities. Second, community detection algorithms ought to support overlapping communities, while graph partitions seek explicitly disjoint structure. While some partitioning applications may be from assigning nodes to multiple partitions, the Facebook social graph we aim to partition has a modest maximum degree of 5,000, and so we restrict our investigation of creating true partitions.

Label propagation fails to detect overlapping communities [7], though an adaptation does exist [9]. But this failure of the ordinary label propagation algorithm in fact makes it a strong candidate for graph partitioning. The remaining difficulty is therefore that label propagation provides no way of constraining the sizes of any of the resulting community partitions. Our contribution addresses precisely this difficulty.

A previous attempt to “constraint” label propagation utilizes an optimization framework with a cost penalty to encourage balanced partitions [3], but this approach does not cover constraints in a formal sense. The constraint-based algorithm we introduce in this work over the possibility of precisely constraining the size of all the resulting shards. It is in fact not limited to constraints that produce balanced partitions. It is also worth noting that the previous cost penalty approach lacks the computational
efficiency of label propagation, and the largest graph that framework was originally tested on contained just 120,000 edges.

The frequent subgraph mining discussed the graphs with labeled, undirected, connected simple graphs without any specific constraints. That is, the database to be mined contains a set of graphs, each consisting of a set of labeled vertices and labeled but undirected edges, with no other constraints. However, many applications or users may need to enforce various kinds of constraints on the patterns to be mined or seek variant substructure patterns. For example, to mine patterns, each of which contains certain specific vertices/edges, or where the total number of vertices/edges is within a specified range patterns where the average density of the graph patterns is above a threshold. Although it is possible to develop customized algorithms for each such case, there are too many variant cases to consider. Instead, a general framework is needed—one that can classify constraints on the graph patterns. Efficient constraint-based methods can then be developed for mining substructure patterns and their variants.

1.4.1 Mining Closed Frequent Substructures

The first important variation of a frequent substructure is the closed frequent substructure. Take mining frequent subgraphs as an example. As with frequent item set mining and sequential pattern mining, mining graph patterns may generate an explosive number of patterns. This is particularly true for dense data sets, because all of the subgraphs of a frequent graph are also frequent. This is an inherent problem, because according to the Apriori property, all the subgraphs of a frequent substructure must be frequent. A large graph pattern may generate an exponential number of frequent subgraphs. For example, among 423 confirmed active chemical compounds in an AIDS
antiviral screen data set, there are nearly 1 million frequent graph patterns whose support is at least 5%.

This renders the further analysis on frequent graphs nearly impossible [43]. One way to alleviate this problem is to mine only frequent closed graphs, where a frequent graph $G$ is closed if and only if there is no proper super graph $G_0$ that has the same support as $G$. Alternatively, one can mine maximal subgraph patterns where a frequent pattern $G$ is maximal if and only if there is no frequent super-pattern of $G$. A set of closed subgraph patterns has the same expressive power as the full set of subgraph patterns under the same minimum support threshold, because the latter can be generated by the derived set of closed graph patterns.

On the other hand, the maximal pattern set is a subset of the closed pattern set. It is usually more compact than the closed pattern set. However, one cannot use it to reconstruct the entire set of frequent patterns—the support information of a pattern is lost if it is a proper sub pattern of a maximal pattern, yet carries a different support [29].

### 1.4.2 Pattern-Growth Approach Mining

A typical pattern-growth graph mining algorithm, such as span or Close Graph mines labeled, connected, undirected frequent or closed subgraph patterns. Such a graph mining framework can easily be extended for mining alternative substructure patterns. Here this research discusses a few such alternatives.

First, the method can be extended for mining unlabeled or partially labeled graphs. Each vertex and each edge in the previously discussed graphs contain labels. Alternatively, if none of the vertices and edges in a graph is labeled, the graph is
unlabeled. A graph is partially labeled if only some of the edges and/or vertices are labeled. To handle such cases, build a label set that contains the original label set and a new empty label. Label is assigned to vertices and edges that do not have labels. Notice that label may match with any label or with it only, depending on the application semantics. With this transformation, span (and Close Graph) can directly mine unlabeled or partially labeled graphs [31].

Second, examine whether span can be extended for mining non simple graphs. A non simple graph may have a self-loop (i.e., an edge joins a vertex to itself) and multiple edges (i.e., several edges connecting two of the same vertices). In span, this work always first grows backward edges and then forward edges. In order to accommodate self-loops, the growing order should be changed to backward edges, self-loops, and forward edges. If this work allows sharing of the same vertices in two neighboring edges in a DFS code [34], the definition of DFS lexicographic order can handle multiple edges smoothly. Thus span can mine non simple graphs efficiently, too.

Third, this work can see how span can be extended to handle the mining of directed graphs. In a directed graph, each edge of the graph has a defined direction. If this work use a 5-tuple, (i; j; li; l(i; j); l j), to represent an undirected edge. Then for directed edges, a new state is introduced to form a 6-tuple, (i; j;d; li; l(i; j); l j), where d represents the direction of an edge. Let d = +1 be the direction from i (vi) to j (vj), whereas d = -1 is that from j (vj) to i (vi). Notice that the sign of d is not related to the forwardness or backwardness of an edge. When extending a graph with one more edge, this edge may have two choices of d, which only introduces a new state in the growing procedure and need not change the framework of span.
Fourth, the method can also be extended for mining disconnected graphs. There are two cases to be considered: (1) the graphs in the data set may be disconnected, and (2) the graph patterns may be disconnected. For the first case, this work can transform the original data set by adding a virtual vertex to connect the disconnected graphs in each graph. This work then apply span on the new graph data set. For the second case, this research redefines the DFS code. A disconnected graph pattern can be viewed as a set of connected graphs, \( r = g_0; g_1; \ldots; g_m \), where \( g_i \) is a connected graph, \( 0 \leq i \leq m \). Because each graph can be mapped to a minimum DFS code, a disconnected graph \( r \) can be translated into a code, \( t = (s_0; s_1; \ldots; s_m) \), where \( s_i \) is the minimum DFS code of \( g_i \). The order of \( g_i \) in \( r \) is irrelevant. Thus, this research enforce an order in \( fsig \) such that \( s_0 \leq s_1 \leq \ldots \leq s_m \). can be extended by either adding one-edge \( s_{m+1} \) \( (s_m \leq s_{m+1}) \) or by extending \( s_m, \ldots, \) and \( s_0 \). When checking the frequency of node in the graph data set, make sure that \( g_0; g_1; \ldots; g_m \) is disconnected with each other.

Finally, if this research views a tree as a degenerated graph, it is straightforward to extend the method for mining frequent sub trees. In comparison with a general graph, a tree can be considered as a degenerated direct graph that does not contain any edges that can go back to its parent or ancestor nodes. Thus if this research considers that traversal always starts at the root (because the tree does not contain any backward edges), then the span is ready to mine tree structures. Based on the mining efficiency of the pattern growth–based approach, it is expected that span can achieve good performance in tree-structure mining.

### 1.4.3 Constraint-Based Mining of Substructure Patterns

Various kinds of constraints can be associated with a user’s mining request. Rather than developing many case-specific substructure mining algorithms, it is more
appropriate to set up a general framework of constraint-based substructure mining so
that systematic strategies can be developed to push constraints deep into the mining
process. Constraint-based mining of frequent substructures can be developed
systematically, similar to the constraint-based mining of frequent patterns and
sequential patterns. Take graph mining as an example. As with the constraint based
frequent pattern mining framework, graph constraints can be classified into a few
categories, including ant monotonic, monotonic, and succinct. Efficient constraint-
based mining methods can be developed in a similar way by extending efficient graph-
pattern mining algorithms, such as span and Close Graph[36].

Constraint-based substructure mining represents the commonly encountered
classes of constraints to see how the constraint-pushing technique can be integrated into
the pattern-growth mining framework.

(i). **Element, set, or subgraph containment constraint.** Suppose a user requires that
the mined patterns contain a particular set of subgraphs. This is a succinct constraint,
which can be pushed deep into the beginning of the mining process. That is, this
research can take the given set of subgraphs as a query, perform selection first using the
constraint, and then mine on the selected data set by growing (i.e., extending) the
patterns from the given set of subgraphs. A similar strategy can be developed if this
research requires that the mined graph pattern must contain a particular set of edges or
vertices.

(ii). **Geometric constraint.** A geometric constraint can be that the angle between each
pair of connected edges must be within a range, written as “CG = min angle _ angle
(e1;e2;v;v1;v2) _ max angle,” where two edges e1 and e2 are connected at vertex v
with the two vertices at the other ends as v1 and v2, respectively. CG is an ant
monotonic constraint because if one angle in a graph formed by two edges does not satisfy CG, further growth on the graph will never satisfy CG. Thus CG can be pushed deep into the edge growth process and reject any growth that does not satisfy CG [37].

(iii). Value-sum constraint. For example, such a constraint can be that the sum of (positive) weights on the edges, some, be within a range low and high. This constraint can be split into two constraints, some_low and some_high. The former is a monotonic constraint, because once it is satisfied, further “growth” on the graph by adding more edges will always satisfy the constraint. The latter is an ant monotonic constraint, because once the condition is not satisfied, further growth of some will never satisfy it. The constraint pushing strategy can then be easily worked out. Notice that a graph-mining query may contain multiple constraints.

For example, this research may want to mine graph patterns that satisfy constraints on both the geometric and minimal sum of edge weights. In such cases, this research should try to push multiple constraints simultaneously, exploring a method similar to that developed for frequent item set mining. For the multiple constraints that are difficult to push in simultaneously, customized constraint-based mining algorithms should be developed accordingly.

1.4.4 Mining Approximate Frequent Substructures

An alternative way to reduce the number of patterns to be generated is to mine approximate frequent substructures, which allow slight structural variations. With this technique, this research can represent several slightly different frequent substructures using one approximate substructure.
The principle of minimum description length is adopted in a substructure discovery system called SUBDUE, which mines approximate frequent substructures. It looks for a substructure pattern that can best compress a graph set based on the Minimum Description Length (MDL) principle, which essentially states that the simplest representation is preferred. SUBDUE adopts a constrained beam search method. It grows a single vertex incrementally by expanding a node in it. At each expansion, it searches for the best total description length; the description length of the pattern and the description length of the graph set with all the instances of the pattern condensed into single nodes [39]. SUBDUE perform approximate matching to allow slight variations of substructures, thus supporting the discovery of approximate substructures.

There should be many different ways to mine approximate substructure patterns. Some may lead to a better representation of the entire set of substructure patterns, whereas others may lead to more efficient mining techniques. More research is needed in this direction.

1.4.5 Mining Coherent Substructures

A frequent substructure G is a coherent subgraph if the mutual information between G and each of its own subgraphs is above some threshold. The number of coherent substructures is significantly smaller than that of frequent substructures. Thus, mining coherent substructures can efficiently prune redundant patterns (i.e., patterns that are similar to each other and have similar support).

A promising method was developed for mining such substructures. Its experiments demonstrate that in mining spatial motifs from protein structure graphs, the discovered coherent substructures are usually statistically significant. This indicates
that coherent substructure mining selects a small subset of features that have high
distinguishing power between protein classes.

1.4.6 Mining Dense Substructures

In the analysis of graph pattern mining, researchers have found that there exists
a specific kind of graph structure, called a relational graph, where each node label is
used only once per graph. The relational graph is widely used in modeling and
analyzing massive networks (e.g., biological networks, social networks, transportation

In biological networks, nodes represent objects like genes, proteins, and
enzymes, whereas edges encode the relationships, such as control, reaction, and
correlation, between these objects. In social networks, each node represents a unique
entity, and an edge describes a kind of relationship between entities. One particular
interesting pattern is the frequent highly connected or dense subgraph in large relational
graphs. In social networks, this kind of pattern can help identify groups where people
are strongly associated. In computational biology, a highly connected subgraph could
represent a set of genes within the same functional module (i.e., a set of genes
participating in the same biological pathways). This may seem like a simple constraint-
pushing problem using the minimal or average degree of a vertex, where the degree of a
vertex \( v \) is the number of edges that connect \( v \).

Unfortunately, things are not so simple. Although average degree and minimum
degree display some level of connectivity in a graph, they cannot guarantee that the
graph is connected in a balanced way. Some part of a graph may be loosely connected
even if its average degree and minimum degree are both high [40].
The removal of edge e1 would make the whole graph fall apart. This research may enforce the following downward closure constraint; a graph is highly connected if and only if each of its connected subgraphs is highly connected. However, some global tightly connected graphs may not be locally well connected. It is too strict to have this downward closure constraint.

Thus, this research adopt the concept of edge connectivity, as follows: Given a graph G, an edge cut is a set of edges Ec such that E(G) \(\subseteq\) Ec is disconnected. A minimum cut is the smallest set in all edge cuts. The edge connectivity of G is the size of a minimum cut. A graph is dense if its edge connectivity is no less than a specified minimum cut threshold.

Now the problem becomes how to mine closed frequent dense relational graphs that satisfy a user-specified connectivity constraint. There are two approaches for mining such closed dense graphs efficiently: a pattern-growth approach called Close-Cut and a pattern-reduction approach called Splat.

This research briefly outlines their ideas as follows. Similar to pattern-growth frequent item set mining, Close Cut first starts with a small frequent candidate graph and extends it as much as possible by adding new edges until it finds the largest super graph with the same support (i.e., its closed super graph). The discovered graph is decomposed to extract subgraphs satisfying the connectivity constraint. It then extends the candidate graph by adding new edges, and repeats the above operations until no candidate graph is frequent. Instead of enumerating graphs from small ones to large ones, Splat directly intersects relational graphs to obtain highly connected graphs.
Let pattern $g$ be a highly connected graph in relational graphs $G_{i1}; G_{i2}; \ldots$, and $G_{iL}$ ($i1 < i2 < \ldots < iL$). In order to mine patterns in a larger set $G_{i1}; G_{i2}; \ldots; G_{iL}, G_{iL+1}$, Splat intersects $g$ with graph $G_{iL+1}$. Let $g_0 = g \setminus G_{iL+1}$. Some edges in $g$ may be removed because they do not exist in graph $G_{iL+1}$. Thus, the connectivity of the new graph $g_0$ may no longer satisfy the constraint. If so, $g_0$ is decomposed into smaller highly connected subgraphs.

This research progressively reduces the size of candidate graphs by intersection and decomposition operations. This research calls this approach a pattern-reduction approach. Both methods have shown good scalability in large graph data sets. Close Cut has better performance on patterns with high support and low connectivity. On the contrary, Splat can filter frequent graphs with low connectivity in the early stage of mining, thus achieving better performance for the high-connectivity constraints. Both methods are successfully used to extract interesting patterns from multiple biological networks.

1.5 Graph Kernel Matching

Graphs are well-studied versatile representations of structured data and have become ubiquitous in many application domains like chemical informatics and bioinformatics. Comparing graphs is a fundamental problem and computing meaningful similarity measures is a prerequisite to apply a variety of machine learning algorithms to the domain of graphs. Consequently related problems have been extensively studied involving essential graph theoretical questions, which are typically NP-hard, like, e.g., the maximum common subgraph problem. However, graph similarity can be needed in various ways and its computation not necessarily requires solving these problems exactly to yield a similarity measure appropriate for a wide range of applications.
To become applicable to the wealth of so-called kernel methods, including Support Vector Machines as the most prominent example, similarity measures must satisfy the additional constraints to be symmetric and positive (p.s.d.).

While recent development of graph kernels primarily focuses on large datasets of graphs with simple labels (cf. Table 1), it has been observed on several occasions that the prediction accuracy can be increased by annotating vertices or edges with additional attributes (see, e.g., Borgwardt et al., 2005; Frohlich et al., 2005; Harchaoui & Bach, 2007). Since attributes in many cases include continuous values, a meaningful similarity measure must tolerate certain divergence. Therefore, kernels designed for graphs with simple labels often are not suitable for attributed graphs.

We propose a new graph kernel which is related to the maximum common subgraph problem. Instead of deriving a similarity measures from a maximum common subgraph our approach counts the number of matching between subgraphs up to size and therefore has polynomial runtime. Attributes of mapped vertices and edges are assessed by an exile scoring scheme and, thus, the approach can be applied to general attributed graphs.

In recent years various graph kernels have been proposed, see (Vishwanathan et al., 2010) and references therein. Gartner et al. (2003) and Kashima et al. (2003) devised graph kernels based on random walks, which count the number of labeled walks two graphs have in common. Subsequently random walk kernels were extended to avoid repeated consecutive vertices and were combined with vertex label enrichment techniques by Mahe et al. (2004). The runtime was improved particularly for graphs with simple labels (Vishwanathan et al., 2010). Random walk kernels have been
extended to take vertex and edge attributes into account and was thereby successfully applied to protein function.

The proposed techniques can be classified into approaches that use explicit feature mapping and those that directly compute a kernel function. If explicit representations are manageable, these approaches usually outperform other kernels regarding runtime on large datasets, since the number of vector representations scales linear with the dataset size. However, these approaches do not support attributed graphs, cf. Table 1. The computation technique proposed for random walk and tree pattern kernels, in contrast, can be extended to compare vertex and edge attributes by kernels. However, compared to graph let kernels these approaches are based on simple features including repeated vertices.

Naturally simple: However, computing kernels by counting common subgraphs of unbounded size is known to be NP-complete (Gartner et al., 2003). Thus, another direction in the development of graph kernels focuses on small subgraphs of a size k2 f3; 4; 5g, referred to as graph lets, which primarily apply to unlabeled graphs (Shervashidze et al., 2009). Further more tree patters, which are allowed to contain repeated vertices just like random walks, were proposed by Ramon & Gartner (2003) and later re need by Mahe & Vert (2009).

While these approaches are based on all common sub tree patterns of a specified height, others only take the entire neighborhood of each vertex up to given distance into account (Shervashidze et al., 2011), thus reducing the number of features and the required run-time significantly. Menchetti et al. (2005) proposed a weighted decomposition kernel, which determines matching substructures by a restrictive kernel (selectors) and weights each matching by a kernel needed on the context of the
matching. A kernel based on shortest-paths was developed by Borgwardt & Kriegel (2005), which rest computes the length of shortest-paths between all pairs of vertices and then counts pairs with similar labels and distance. Instead of comparing pairs of individual vertices, the kernel proposed by Costa & De Grave (2010) associates a string encoding the neighborhood subgraph with each vertex.

Several graph kernels were tailored especially to chemical compound. For attributed molecular graphs Frohlich et al. (2005) proposed a similarity measure based on an optimal assignment of vertices. However, the proposed function was shown not be p.s.d. (Vishwanathan et al., 2010). Established techniques in chemical informatics are based on features which can be; 1. Directly generated from the molecular graph, e.g. all paths or subgraphs up to a certain size (Wale et al., 2008), similar to graph lets, 2. Taken from a pride need dictionary or 3 generated in a preceding data mining phase, e.g. using frequent subgraph mining.

We propose a technique that uses small subgraphs contained in the two graphs under comparison, similar to graph lets, but simultaneously provide the edibility to compare vertex and edge attributes by means of arbitrary kernel functions.

1.5.1 Kernels on Graph spaces

To apply kernel methods to the classification of graphs, it remains to define positive definite kernel functions on the set of all graphs or on application specific subsets of graphs. A typical application of this kind of kernels is the classification of chemical compounds, given their atom bond structure. The strategy to define such kernel functions on graphs that has been followed mostly in literature is loosely speaking to decompose the graphs into possibly overlapping parts and then to apply one of the above described kernels on sets.
Let us first consider an intersection kernel with set cardinality as the measure and based on a decomposition that maps each graph into the set of all of its subgraphs. Using this kernel function, graphs satisfying certain properties can be identified. In particular, one could decide whether a graph has a Hamiltonian path, i.e., a sequence of adjacent vertices and edges that contains every vertex and edge exactly once. Now this problem is known to be NP-complete; therefore it is strongly believed that such kernels can not be computed in polynomial time. Furthermore, it can be shown that computing any graph kernel based on the intersection of injective decompositions is at least as hard as deciding graph isomorphism. We thus need to consider alternative, less expressive, graph kernels.

In literature deferent approaches have been tried to overcome this problem. [Graepel, 2002] restricted the decomposition to paths up to a given size, and [Deshpande et al., 2002] only consider the set of connected graphs that occur frequently as subgraphs in the graph database. The approach taken there to compute the decomposition of each graph is an iterative one. An alternative approach is based on measuring the number of walks in (directed or undirected) graphs with common label sequence. Although the set of common walks can be infinite, the inner product in this feature space can be computed in polynomial time by first building the product graph and then computing the limit of a matrix power series of the adjacency matrix [Gartner et al., 2003]. An alternative walk based kernel function exploits only the length of all walks between all pairs of vertices with given label.

1.6 Graph Mining Strategies

Many of the traditional mining applications also apply to the case of graphs. As in the case of management applications, the mining applications are far more
challenging to implement because of the additional constraints which arise from the structural nature of the underlying graph. In spite of these challenges, a number of techniques have been developed for traditional mining problems such as frequent pattern mining, clustering, and classification.

1.6.1 Pattern Mining in Graphs

The problem of frequent pattern mining has been widely studied in the context of mining transactional data. Recently, the techniques for frequent pattern mining have also been extended to the case of graph data. The main difference in the case of graphs is that the process of determining support is quite different. The problem can be defined in different ways depending upon the application domain.

In the first case, a group of graphs, to determine all patterns which support a fraction of the corresponding graphs. In the second case, this work have a single large graph, and this work wish to determine all patterns which are supported at least a certain number of times in this large graph. In both cases, this work needs to account for the isomorphism issue in determining whether one graph is supported by another. However, the problem of defining the support is much more challenging, if overlaps are allowed between different embeddings. This is because if this work allows such overlaps, then the anti monotonic property of most frequent pattern mining algorithms is violated.

For the first case, where this work have a data set containing multiple graphs, most of the well known techniques for frequent pattern mining with transactional data can be easily extended. For example, Apriori style algorithms can be extended to the case of graph data, by using a similar level-wise strategy of generating \((k + 1)\)-candidates from \(k\)-patterns. The main difference is that this work needs to define the
join process a little differently. Two graphs of size $k$ can be joined, if they have a structure of size $(k - 1)$ in common. The size of this structure could be defined in terms of either nodes or edges. In the case of the AGM algorithm, this common structure is defined in terms of the number of common vertices. Thus, two graphs with $k$ vertices are joined, only if they have a common subgraph with at least $(k - 1)$ vertices. A second way of performing the mining is to join two graphs which have a subgraph containing at least $(k - 1)$ edges in common.

The FSG algorithm can be used in order to perform edge-based joins. It is also possible to define the joins in terms of arbitrary structures. For example, it is possible to express the graphs in terms of edge-disjoint paths. In such cases, subgraphs with $(k + 1)$ edge disjoint paths can be generated from two graphs which have $k$ edge disjoint paths, of which $(k - 1)$ must be common. Another strategy which is often used is that of pattern growth techniques, in which frequent graph patterns are extended with the use of additional edges.

As in the case of frequent pattern mining problem, this work use lexicographic ordering among edges in order to structure the search process, so that a given pattern is encountered only once. For the second case in which this work has a single large graph, a number of different techniques may be used in order to define the support in presence of the overlaps. A common strategy is to use the size of the maximum independent set of the overlap graph to define the support. This is also referred to as the maximum independent set support.

1.6.2 Clustering Algorithms for Graph Data

This section will discuss a variety of algorithms for clustering graph data. This includes both classical graph clustering algorithms as well as algorithms for clustering
XML data. Clustering algorithms have significant applications in a variety of graph scenarios such as congestion detection, facility location, and XML data integration. Within the context of graph algorithms, the clustering can be of two types:

- **Node Clustering Algorithms:**

  In this case, this work has one large graph, and this work attempt to cluster the underlying nodes with the use of a distance (or similarity) value on the edges, the edges of the graph are labeled with numerical distance values. These numerical distance values are used in order to create clusters of nodes. A particular case is one in which the presence of an edge refers to a similarity value of 1, whereas the absence of an edge refers to a similarity value of 0. This work note that the problem of minimizing the inter-cluster similarity for a fixed number of clusters essentially reduces to the problem of *graph partitioning* or the *minimum multi-way cut problem*. This is also referred to as the problem of mining dense graphs and pseudo-cliques. Recently, the problem has also been studied in the database literature as that of *quasi-clique determination*. In this problem, this work determines groups of nodes which are “almost cliques”. In other words, an edge exists between any pair of nodes in the set with high probability.

- **Graph Clustering Algorithms:**

  Number of graphs which need to be clustered based on their underlying structural behavior. This problem is challenging because of the need to match the structures of the underlying graphs, and use these structures for clustering purposes. Such algorithms are discussed both in the context of classical graph data sets as well as semistructured data. Therefore, this work will discuss both of these variations.
The graph clustering problem is related to the minimum cut and graph partitioning problems. In this case, it is assumed that the underlying graphs have weights on the edges. It is desired to partition the graph in such a way so as to minimize the weights of the edges across the partitions. The simplest case is the 2-way minimum cut problem, in which this work wishes to partition the graph into two clusters, so as to minimize the weight of the edges across the partitions. This version of the problem is efficiently solvable, and can be resolved by repeated applications of the maximum flow problem. This is because the maximum flow between source s and sink t determines the minimum s-t cut. By using different source and sink combinations, it is also possible to find the global minimum cut. A second way of determining a minimum cut is by using a contraction based edge sampling approach. This is a probabilistic technique in which this work successively sample edges in order to collapse nodes into larger sets of nodes. By successively sampling different sequences of edges and picking the optimum value, it is possible to determine a global minimum cut. Both of the above techniques are quite efficient and the time complexity is polynomial in terms of the number of nodes and edges.

1.6.3 Classical Algorithms for Clustering XML and Graph Data.

There are two main techniques used for clustering of XML documents. These techniques are as follows:

(i) Structural Distance-based Approach:

This approach includes the distances between documents and uses them in order to compute clusters of documents. Such distance based approaches are quite general and effective techniques over a wide variety of non numerical domains such as categorical and string data. It is therefore natural to explore this technique in the
context of graph data. One of the earliest works on clustering tree structured data is the XClust algorithm, which was designed to cluster XML schemas for efficient integration of large numbers of Document Type Definitions (DTDs) of XML sources. It adopts the agglomerative hierarchical clustering method which starts with clusters of single DTDs and gradually merges the two most similar clusters into one larger cluster. The similarity between two DTDs is based on their element similarity, which can be computed according to the semantics, structure, and context information of the elements in the corresponding DTDs.

One of the shortcomings of the XClust algorithm is that it does not make full use of the structure information of the DTDs, which is quite important in the context of clustering tree like structures. The method computes similarity measures based on the structural edit distance between documents. This edit distance is used in order to compute the distances between clusters of documents.

Another clustering technique which falls in this general class of methods is the S-GRACE algorithm. The main idea is to use the element sub element relationships in the distance function rather than the simple use of the tree edit distance. S-GRACE is a hierarchical clustering algorithm. An XML document is converted to a structure graph (or s graph), and the distance between two XML documents is defined according to the number of the common element sub element relationships, which can capture better structural similarity relationships than the tree edit distance in some cases.

(ii) Structural Summary Based Approach:

In many cases, it is possible to create summaries from the underlying documents. These summaries are used for creating groups of documents which are similar to these summaries. The XML documents are modeled as rooted ordered
labeled trees. A framework for clustering XML documents by using structural summaries of trees is presented. The aim is to improve algorithmic efficiency without compromising cluster quality. XProj is the solution technique called as partition based algorithm.

1.6.4 Classification Algorithms for Graph Data

Classification is a central task in data mining and machine learning. As graphs are used to represent entities and their relationships in an increasing variety of applications, the topic of graph classification has attracted much attention in both academia and industry. For example, in pharmaceutics and drug design, this work is interested to know the relationship between the activity of a chemical compound and the structure of the compound, which is represented by a graph. In social network analysis, this work study the relationship between the health of a community (e.g., whether it is expanding or shrinking) and its structure, which again is represented by graphs. Graph classification is concerned with two different but related learning tasks.

(i) Label Propagation

A subset of nodes in a graph is labeled. The task is to learn a model from the labeled nodes and use the model to classify the unlabeled nodes.

(ii) Graph classification

A subset of graphs in a graph dataset is labeled. The task is to learn a model from the labeled graphs and use the model to classify the unlabeled graphs. The concept of label or belief is a fundamental technique which is used in order to leverage graph structure in the context of classification in a number of relational domains. The scenario of label propagation occurs in many applications. As an example, social
network analysis is being used as a mean for targeted marketing. Retailers track customers who have received promotions from them. Those customers who respond to the promotion (by making a purchase) are labeled as positive nodes in the graph representing the social network, and those who do not respond are labeled as negative.

The goal of target marketing is to send promotions to customers who are most likely to respond to promotions. It boils down to learning a model from customers who have received promotions and predicting the responses of other potential customers in the social network. Intuitively, this work wants to find out how existing positive and negative labels propagate in the graph to unlabeled nodes. Based on the assumption that “similar” nodes should have similar labels, the core challenge for label propagation lies in devising a distance function that measures the similarity between two nodes in the graph. One common approach of defining the distance between two nodes is to count the average number of steps it takes to reach one node from the other using a random walk.

However, it has a significant drawback: it takes $O(n^3)$ time to derive the distances and $O(n^2)$ space to store the distances between all pairs. However, many graphs in real life applications are sparse, which reduces the complexity of computing the distance. A method whose complexity is nearly linear to the number of non zero entries of the sparse coefficient matrix.

A survey of label propagation methods can be found. Kernel-based Graph Classification Methods. Kernel-based graph classification employs a graph kernel to measure the similarity between two labeled graphs. The method is based on random walks. For each graph, this work enumerates its paths, and this work derives probabilities for such paths. The graph kernel compares the set of paths and their
probabilities between the two graphs. A random path (represented as a sequence of node and edge labels) is generated via a random walk: First, this work randomly selects a node from the graph. During the next and each of the subsequent steps, this work either stops (the path ends) or randomly select an adjacent node to continue the random walk.

The choices this work makes are subject to a given stopping probability and a node transition probability. By repeating the random walks, this work derives a table of paths, each of which is associated with a probability. In order to measure the similarity between two graphs, this work needs to measure the similarity between nodes, edges, and paths.

1.6.5 Similarity between nodes, edges, and path kernels

The similarities between various graph kernel components are as follows:

(i) Node/Edge kernel

An example of a node/edge kernel is the identity kernel. If two nodes/edges have the same label, then the kernel returns 1 otherwise 0. If the node/edge labels take real values, then a Gaussian kernel can be used instead.

(ii) Path kernel

A path is a sequence of node and edge labels. If two paths are of the same length, the path kernel can be constructed as the product of node and edge kernels. If two paths are of different lengths, the path kernel simply returns 0.
(iii) Graph kernel

As each path is associated with a probability, this work can define the graph kernel as the expectation of the path kernel over all possible paths in the two graphs. The above definition of a graph kernel is straightforward. However, it is computationally infeasible to enumerate all the paths. In particular, in cyclic graphs, the length of a path is unbounded, which makes enumeration impossible. Thus, more efficient approaches are needed to compute the kernel. It turns out that the definition of the kernel can be reformulated to show a nested structure.

In the case of directed acyclic graphs the nodes can be topologically ordered such that there is no path from node $j$ to $i$ if $i < j$, the kernel can be redefined as a recursive function, and dynamic programming can handle this problem in $O(||X|| \cdot ||X'||)$, where $X$ and $X'$ are the set of nodes in two graphs. In the case of cyclic graphs, the kernel’s feature space (label sequences) is possibly infinite because of loops. The computation of cyclic graph kernel can still be done with linear system theory and convergence properties of the kernel.

1.6.6 Boosting-based Graph Classification Methods

The kernel based method provides an elegant solution to graph classification, it does not explicitly reveal what graph features (substructures) are relevant for classification. To address this issue, a new approach of graph classification based on pattern mining is introduced. The idea is to perform graph classification based on a graph’s important substructures. This work can create a binary feature vector based on the presence or absence of a certain substructure (subgraph) and apply an off-the-shelf classifier.
Since the entire set of subgraphs is often very large, this work must focus on a small subset of features that are relevant. The most straightforward approach for finding interesting features is through frequent pattern mining. However, frequent patterns are not necessarily relevant patterns. For instance, in chemical graphs, ubiquitous patterns such as C-C or C-C-C are frequent, but have almost no significance in predicting important characteristics of chemical compounds such as activity, toxicity, etc.,

Boosting is used to automatically select a relevant set of subgraphs as features for classification. LPBoost (Linear Program Boost) learns a linear discriminate function for feature selection. To obtain an interpretable rule, this work need to obtain a sparse weight vector, where only a few weights are nonzero. Graph boosting can achieve better accuracy than graph kernels, and it has the advantage of discovering key substructures explicitly at the same time.

The problem of graph classification is closely related to that of XML classification. This is because XML data can be considered an instance of rich graphs, in which nodes and edges have features associated with them. Consequently, many of the methods for XML classification can also be used for structural graph classification. In a rule-based classifier (called XRules) was proposed in which this work associate structural features on the left-hand side with class labels on the right-hand side. The structural features on the left-hand side are determined by computing the structural features in the graph which are both frequent and discriminative for classification purposes. These structural features are used in order to construct a prioritized list of rules which are used for classification purposes. The top-k rules are determined based
on the discriminative behavior and the majority class label on the right hand side of these k rules is reported as the final result.

1.6.7 Other Related Work

The problem of node classification arises in a number of different application contexts such as relational data classification, social network classification, and blog classification, which uses link-based similarity for node-classification in the context of relational data. This approach constructs link features from the underlying structure and uses them in order to create an effective model for classification. Recently, this technique has also been used in the context of link-based classification of blogs. However, all of these techniques use link-based methods only. Since many of these techniques arise in the context of text data, it is natural to examine whether such content can be used in order to improve classification accuracy.

A method to perform collective classification of email speech acts has been shown that the analysis of relational aspects of emails (such as emails in a particular thread) significantly improves the classification accuracy. The use of graph structures during categorization improves the classification accuracy of web pages.

1.6.8 The Dynamics of Time-Evolving Graphs

Many networks in real applications arise in the context of networked entities such as the web, mobile networks, military networks, and social networks. In such cases, it is useful to examine various aspects of the evolution dynamics of typical networks, such as the web or social networks. Thus, this line of research focuses on modeling the general evolution properties of very large graphs which are typically encountered. Considerable study has been devoted to that of examining generic
evolution properties which hold across massive networks such as web networks, citation networks and social networks. Some examples of such properties are as follows:

➢ **Densification**

Most real networks such as the web and social networks continue to become denser over time. This essentially means that these networks continue to add more links over time (than are deleted). This is a natural consequence of the fact that much of the web and social media is a relatively recent phenomenon for which new applications continue to be found over time. In fact most real graphs are known to exhibit a densification power law, which characterizes the variation in densification behavior over time. This law states that the number of nodes in the network increases super linearly with the number of nodes over time, whereas the number of edges increases super-linearly over time. In other words, if \( n(t) \) and \( e(t) \) represent the number of edges and nodes in the network at time \( t \), the value of the exponent lies between 1 and 2.

➢ **Shrinking Diameters**

The small world phenomenon of graphs is well known. For example, this can be considered a verification of the (internet version of the) widely known rule of “six degrees of separation” in (generic) social networks. The diameters of massive networks such as the web continue to shrink over time. This may seem surprising, because one would expect that the diameter of the network should grow as more nodes are added. However, it is important to remember that edges are added more rapidly to the network than nodes.
As more edges are added to the graph it becomes possible to traverse from one node to another with the use of a fewer number of edges. While the above observations provide an understanding of some key aspects of specific aspects of long term evolution of massive graphs, they do not provide an idea of how the evolution in social networks can be modeled in a comprehensive way. It uses the maximum likelihood principle in order to characterize the evolution behavior of massive social networks. This work uses data driven strategies in order to model the online behavior of networks. The work studies the behavior of four different networks, and uses the observations from these networks in order to create a model of the underlying evolution. It also shows that edge locality plays an important role in the evolution of social networks.

A complete model of a node’s behavior during its lifetime in the network is studied in this work. Another possible line of work in this domain is to study methods for characterizing the evolution of specific graphs. For example, in a social network, it may be useful to determine the newly forming or decaying communities in the underlying network. The expanding or contracting communities in a social network may be characterized by examining the relative behavior of edges, as they are received in a dynamic graph stream.

The techniques in this paper characterize the structural behavior of the incremental graph within a given time window, and uses it in order to determine the birth and death of communities in the graph stream. This is the first piece of work which studies the problem of evolution in fast streams of graphs. It is particularly challenging to study the stream case, because of the inherent combinatorial complexity of graph structural analysis, which does not lend itself well to the stream scenario. Statistical analysis and visualization in order to provide a better idea of the changing
community structure in an evolving social network. A method in performs parameter-free mining of large time Evolve g graphs. This technique can determine the evolving communities in the network, as well as the critical change-points in time. A key property of this method is that it is parameter-free, and this increases the usability of the method in many scenarios.

This is achieved with the use of the MDL principle in the mining process. A related technique can also perform parameter free analysis of evolution in massive networks use of the MDL principle. The method can determine which communities have shrunk, split, or emerged over time. The problem of evolution in graphs is usually studied in the context of clustering, because clusters provide a natural summary for understanding both the underlying graph and the changes inherent during the evolution process. The need for such characterization arises in the context of massive networks, such as interaction graphs, community detection in social networks and generic clustering changes in linked information networks.

An event based framework, which provides an understanding of the typical events which occur in real networks, when new communities may form, evolve, or dissolve. Thus, this method can provide an easy way of making a quick determination of whether specific kinds of changes may be occurring in a particular network. A key technique used by many methods is to analyze the communities in the data over specific time slices and then determine the change between the slices to diagnose the nature of the underlying evolution.

The method deviates from this two-step approach and constructs a unified framework for the determination of communities with the use of a best fit to a temporal-smoothness model. It is a spectral method for evolutionary clustering, which
is also based on the temporal-smoothness concept. Also it is the technique for evolutionary characterization of networks in multi-modal graphs. Finally, a heuristic method combines the problem of clustering and evolutionary analysis into one framework, and shows how to determine evolving clusters in a dynamic environment. It uses a density-based characterization in order to construct nanoclusters which are further leveraged for evolution analysis.

A different approach is to use association rule based mining techniques. The algorithm takes a sequence of snapshots of an evolving graph, and then attempts to determine rules which define the changes in the underlying graph. Frequently occurring sequences of changes in the underlying graph are considered important indicators for rule determination. Furthermore, the frequent patterns are decomposed in order to study the confidence that a particular sequence of steps in the past will lead to a particular transition. The probability of such a transition is referred to as confidence. The rules in the underlying graph are then used in order to characterize the overall network evolution. Another form of evolution in the networks is in terms of the underlying flow of communication (or information).

Since the flow of communication and information implicitly defines a graph (stream), the dynamics of this behavior can be very interesting to study for a number of different applications. Such behaviors arise often in a variety of information networks such as social networks, blogs, or author citation graphs.

In many cases, the evolution may take the form of cascading information through the underlying graphs. The idea is that information propagates through the social network through contact between the different entities in the network.