



2  
2007



This is to certify that the  
dissertation entitled

GRAPH BASED METHODS FOR PATTERN MINING

presented by

H. D. K. Moonesinghe

has been accepted towards fulfillment  
of the requirements for the

Ph.D. degree in Computer Science

*Jan Paynter*

Major Professor's Signature

Oct 18, 2007

Date

**PLACE IN RETURN BOX** to remove this checkout from your record.  
**TO AVOID FINES** return on or before date due.  
**MAY BE RECALLED** with earlier due date if requested.

DATE DUE	DATE DUE	DATE DUE

**GRAPH BASED METHODS FOR PATTERN MINING**

By

H. D. K. Moonesinghe

A DISSERTATION

Submitted to  
Michigan State University  
in partial fulfillment of the requirements  
for the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science

2007



## ABSTRACT

### GRAPH BASED METHODS FOR PATTERN MINING

By

H. D. K. Moonesinghe

Pattern mining is the automatic extraction of novel and potentially useful knowledge from large databases. It plays a major role in many applications, such as intrusion detection, business intelligence, and bio-medical applications. This thesis explores two important pattern mining tasks, namely, frequent pattern mining and anomalous pattern mining. We illustrate the limitations of existing pattern mining algorithms and develop a class of algorithms inspired by graph theoretic principles to overcome these limitations.

First, we demonstrate characteristics of existing frequent pattern mining algorithms that prohibit them from scaling-up to very large databases. We introduce a novel data structure known as a *PrefixGraph* to compress crucial information about the database in order to facilitate fast enumeration of frequent patterns. An efficient pattern search and pruning algorithm called *PGMiner* was also developed using principles derived from network flow analysis.

Second, we investigate a class of anomalies that are difficult to distinguish from normal observations, resulting in high false alarm rates in many anomaly detection algorithms. To address this problem, we introduce *OutRank*, a stochastic graph based algorithm for unsupervised anomaly detection. In this method, a graph is constructed using two approaches: one based on the similarity between objects and the other using shared neighbors of the object. The heart of this approach uses a *Markov chain random walk* model to determine the anomaly score of an object.

Recent years have also witnessed the proliferation of graph-based data, spurred by advances in application areas such as bioinformatics, chem-informatics, Web 2.0, and

sensor networks. In this thesis, we systematically develop an anomaly detection framework to detect anomalies in graph-based data. First, frequent subgraph patterns are extracted from the data. We investigate two methods for utilizing the frequent subgraph patterns in graph anomaly detection. The first approach, *gRank*, is an extension of *OutRank* to graph-based data. It uses a substructure-based similarity measure to create a stochastic graph and then performs random walk to determine the anomaly score of a graph. The second approach, called *gEntropy*, creates features based on the frequent subgraph patterns and builds a probability model for the graphs using the *maximum entropy* principle. Anomalous graphs are detected based on the probability that such a graph is generated by the model. Finally, we extend the maximum entropy approach to supervised anomaly detection and show that it produces significant improvements over the unsupervised case.

Graph based methods have been applied in several areas such as *machine learning* and *networking*. In this thesis, we show how graph based methods can be applied successfully in pattern mining area. More specifically, we show how graph theoretic approaches can be utilized to improve the efficiency and the effectiveness of frequent item-set mining and anomaly detection. With the advances in technology, a huge amount of data is accumulated everyday and database sizes have grown to *Terabyte* and *Petabyte* scale. We believe the algorithms developed in this thesis are a step forward towards making pattern mining more effective for such large and complex data sets.

To  
*Parents*  
For their love and affection

## ACKNOWLEDGEMENTS

Many people have contributed to this work directly or indirectly. It is an honor to have come across so many wonderful people.

I express deep gratitude to my advisor Dr. Pang-Ning Tan, for allowing me to pursue this research and helping me along every step of the way. This thesis would not have been in its present form without his generous support, motivation, and guidance. I highly appreciate his periodic feedback and inspirational communication that kept me pointed in the right direction to achieve this goal.

I am grateful to the late Dr. Moon-Jung Chung for being my academic advisor in the early days of my degree. He was a very caring, understanding, great teacher who exposed me to the exciting field of data mining.

I am thankful to the members of my thesis committee— Dr. Anthony S. Wojcik, Dr. Abdol-Hossein Esfahanian, and Dr. Shinhan Shiu. Communication with them has always brought me cheerful spirits and inspiration.

I would like to thank the faculty of the Department of Computer Science and Engineering at Michigan State University (MSU) for providing an excellent learning, research, and teaching experience. In particular, I would like to thank Dr. George Stockman, Dr. Bill Punch, Dr. John Weng, and former faculty member Dr. Jaejin Lee for their support and encouragement.

I am highly indebted to the Department of Computer Science and Engineering at MSU for providing me with an opportunity to pursue graduate studies at this university. Also, I am grateful to both of the past graduate directors— Dr. Wojcik and Dr. Esfahanian, and present graduate director— Dr. Eric Torng for approving continued financial support for my entire Ph.D. education.

I would like to thank all the staff members of the Department of Computer Science and Engineering for their assistance in administrative tasks. I want to thank Ms. Linda Moore for her patience, care, and advice during this period. Many thanks to Ms. Kim Thompson for proof-reading this entire document in such short notice.

I would also like to thank Mr. Mark McCullen for providing valuable teaching tips and resources that helped me to carryout my teaching duties efficiently, and allowing me to spend more time on my research. Thanks also go to my students from the Computer Organization and Architecture class for a good time.

Many thanks to members of the Data Mining research group— Jerry Scripps, Samah Fodeh, Hamed Valizadegan, and Haibin Cheng, and the members of my previous research group – Yuan Zhang, and Ming Wu for their help and a good time.

I am grateful to all of my past teachers at the University of Colombo for providing me with a sound foundation on all aspects of computer science, which helped me immensely in achieving my career goals.

I would like to thank all of my friends both in the U.S. and Sri Lanka. Although I cannot name them all here, I appreciate the support they provided and the wonderful time we had during this period. A special word of thanks goes to my friend Varuni for her support, patience, and understanding in this period.

These acknowledgements would not be complete without an expression of gratitude to my parents for their continuous support and encouragement. Their love accompanies me wherever I go.

## TABLE OF CONTENTS

<b>LIST OF TABLES</b>	<b>ix</b>
<b>LIST OF FIGURES</b>	<b>x</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Frequent Pattern Mining and its Challenges.....	2
1.2 Anomalous Pattern Mining and its Challenges .....	5
1.3 Graph-based Approaches for Pattern Mining.....	7
1.4 Contributions .....	9
1.5 Organization of the Thesis.....	10
<b>2 Background</b>	<b>11</b>
2.1 Graphs: Basic Concepts.....	11
2.2 Frequent Pattern Mining.....	13
2.2.1 Frequent Itemset Mining .....	13
2.2.2 Frequent Subgraph Mining.....	15
2.2.3 Related Work.....	17
2.3 Mining Anomalous Patterns .....	21
2.3.1 Basic Concepts in Anomaly Detection.....	21
2.3.2 Related Work.....	22
<b>3 <i>PGMiner</i>: Mining Frequent Closed Itemsets</b>	<b>25</b>
3.1 Issues in Frequent Closed Itemset Mining .....	25
3.2 <i>PrefixGraph</i> Representation .....	29
3.2.1 Preliminaries.....	29
3.2.2 <i>PrefixGraph</i> Construction .....	30
3.2.3 Analysis of <i>PrefixGraph</i> Structure .....	33
3.3 Frequent Closed Itemset Mining .....	35
3.3.1 Intra-Node Closed Itemset Mining.....	36
3.3.2 Inter-Node Pruning.....	39
3.4 Mining Algorithm.....	46
3.4.1 Implementation Techniques .....	47
3.4.2 Memory Management.....	48
3.5 Experimental Evaluation .....	48
3.5.1 Evaluating Environment.....	48
3.5.2 Performance Comparisons.....	49
3.5.3 Memory Usage .....	50
3.5.4 Scalability .....	57
3.5.5 Effectiveness of the Flow Based Pruning.....	58
3.6 Summary.....	59

<b>4</b>	<b><i>OutRank</i>: Mining Anomalous Data</b>	<b>60</b>
4.1	Anomaly Detection and its Issues .....	60
4.2	Modeling Anomalies Using a Graph .....	63
4.2.1	Graph Representation .....	63
4.2.2	Markov Chain Model.....	64
4.3	Anomaly Detection Algorithms.....	67
4.3.1	<i>OutRank-a</i> : Using Object Similarity .....	67
4.3.2	<i>OutRank-b</i> : Using Shared Neighbors .....	69
4.4	Experimental Evaluation .....	71
4.4.1	Comparison with Other Approaches .....	72
4.4.2	Effect of the Percentage of Outliers .....	75
4.4.3	Effect of the Shared Neighbor Approach .....	79
4.4.4	Choice of Similarity Measures .....	81
4.4.5	Effect of Threshold on the Quality of Solution .....	82
4.5	Discussion.....	82
<b>5</b>	<b>Mining Anomalous Graphs</b>	<b>85</b>
5.1	Mining Graph Based Data .....	86
5.2	Anomaly Detection in Graphs and its Issues.....	87
5.3	Extending <i>OutRank</i> for Graphs .....	89
5.3.1	<i>gRank</i> : Graph Ranking .....	89
5.3.2	Similarity Measures for Graphs.....	91
5.4	<i>gEntropy</i> : Maximum Entropy Based Unsupervised Anomaly Detection .....	97
5.4.1	Maximum Entropy Model .....	97
5.4.2	Parameter Estimation.....	101
5.4.3	<i>Metropolis</i> Sampling Approach.....	101
5.4.4	Feature Vector Generation.....	104
5.5	Maximum Entropy Based Supervised Anomaly Detection.....	107
5.5.1	Maximum Entropy Model in Supervised Setting .....	109
5.5.2	Parameter Estimation.....	110
5.5.3	<i>gEntropySuper</i> : Classifying Anomalous Graphs.....	111
5.6	Experimental Evaluation .....	113
5.6.1	Datasets.....	113
5.6.2	Evaluating Environment .....	114
5.6.3	Evaluation of the Unsupervised Frameworks.....	114
5.6.4	Evaluation of the Supervised Frameworks .....	116
5.7	Discussion.....	119
<b>6</b>	<b>Conclusions</b>	<b>122</b>
6.1	Summary of the Thesis .....	122
6.1.1	Mining Frequent Itemsets .....	122
6.1.2	Mining Anomalous Patterns .....	124
6.2	Future Research .....	127

<b>BIBLIOGRAPHY</b>	<b>130</b>
---------------------	------------

## LIST OF TABLES

Table 2.1 Sample database .....	14
Table 3.1. Characteristics of various condensed representations .....	27
Table 3.2. Sample database .....	30
Table 3.3. Characteristics of the databases .....	49
Table 3.4. Evaluation of the global closedness techniques .....	59
Table 4.1. Outlier rank for sample 2-D dataset .....	67
Table 4.2. Characteristics of the datasets .....	72
Table 4.3. Experimental results .....	73
Table 4.4. Performance comparison with different similarity measures .....	82
Table 5.1. Characteristics of the datasets .....	114
Table 5.2. Experimental results .....	115
Table 5.3. Characteristics of the datasets .....	116
Table 5.4. Experimental results (Accuracy and F-Measure) .....	118
Table 5.5. Experimental results with maximal subgraphs .....	119



## LIST OF FIGURES

Figure 1.1. Execution time and memory usage for large databases ( $K=1000$ ) .....	4
Figure 1.2. Results of applying LOF anomaly detection algorithm on 2D-Data .....	6
Figure 2.1 Sample graph database and frequent graphs .....	16
Figure 3.1. Different compressed data representations .....	27
Figure 3.2. <i>PrefixGraph</i> construction- a running example.....	31
Figure 3.3. <i>PrefixGraph</i> representation of the sample database.....	32
Figure 3.4. Size of bit vector databases at each node .....	36
Figure 3.5. Itemset enumeration tree (search space) of a node .....	38
Figure 3.6. Transaction flow network for the sample database.....	41
Figure 3.7. Execution time (in seconds) for Medical .....	51
Figure 3.8. Execution time (in seconds) for WebView2 .....	51
Figure 3.9. Execution time (in seconds) for Kosarak .....	52
Figure 3.10. Execution time (in seconds) for Chess.....	52
Figure 3.11. Execution time (in seconds) for WebDocs.....	53
Figure 3.12. Execution time (in seconds) for T20I8D500K.....	53
Figure 3.13. Execution time (in seconds) for Pumsb .....	54
Figure 3.14. Execution time (in seconds) for T40I10D100K.....	54
Figure 3.15. Execution time (in seconds) for T100I20D100K.....	55
Figure 3.16. Amount of memory (in MB) required for T40I10D100K .....	55
Figure 3.17. Amount of memory (in MB) required for Pumsb .....	56
Figure 3.18. Amount of memory (in MB) required for Kosarak.....	56
Figure 3.19. Execution time versus number of transactions ( $K=1000$ ).....	57
Figure 3.20. Memory usage of algorithms for large databases ( $K=1000$ ) .....	58
Figure 4.1. Outlier detection with random walk.....	62

Figure 4.2. Sample 2-D data set .....	66
Figure 4.3. Similarity based on the number of shared neighbors .....	69
Figure 4.4. Precision while varying the % of outliers in Led7 .....	76
Figure 4.5. False alarm rate while varying the % of outliers in Led7 .....	76
Figure 4.6. Precision while varying the % of outliers in KDD .....	77
Figure 4.7. False alarm rate while varying the % of outliers in KDD .....	77
Figure 4.8. Precision while varying the % of outliers in Diabetic.....	78
Figure 4.9. False alarm rate while varying the % of outliers in Diabetic .....	78
Figure 4.10. Precision of algorithms on optical dataset .....	79
Figure 4.11. Connectivity of objects in Austra dataset.....	80
Figure 4.12. Connectivity of objects in Led7 dataset .....	81
Figure 4.13. Precision for different threshold values in Led7 dataset.....	83
Figure 4.14. Precision for different threshold values in KDD dataset .....	83
Figure 5.1. AIDS antiviral screening data and its interesting fragments.....	86
Figure 5.2. Compression based graph similarity .....	89
Figure 5.3. Maximal common frequent subgraph of the two graphs .....	92
Figure 5.4. Frequent subgraph lattice .....	94

# 1 Introduction

WITH the growing scale and complexity of data generated in a variety of scientific, engineering, and commercial applications, a key challenge is to automatically process and analyze data to discover potentially useful information. Traditional data analysis tools are insufficient because they are mostly designed for small-scale problems. This has led to considerable interest in developing data mining technology for extracting knowledge buried in the wealth of data. Data mining, as defined by *Fayyad et al.* [FPS96], is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in large databases. It blends traditional data analysis tools with sophisticated algorithms for handling a massive amount of data and analyzing non-traditional complex data types such as sequences and graphs [TSK06].

There are two general classes of techniques in data mining—one aimed at *model building* and the other aimed at *pattern mining*. In model building, the objective is to construct a global representation that summarizes the data or describes the underlying process in which the data is generated. Examples of models include decision trees, support vector machines, ARIMA time series models, and Gaussian mixture models. Model building, especially for classification and regression tasks, has been predominantly driven by advances in areas such as statistical learning, machine learning, and pattern recognition.

In contrast, *patterns* are local structures hidden in the data involving only a subset of the objects or attributes [H98]. There are two notable types of patterns—*frequent* patterns

and *anomalous* patterns. A frequent pattern is a subset of attributes or a substructure of complex objects that occur frequently in the data. Frequent patterns have been used to identify products that are frequently sold together for up-sell and cross-sell promotions [HCH+98], to find motifs in biological sequences [MS99], to discover chemical compound substructures that can be used in drug design [DKN+05], and to detect bugs in software programs [LYY+05]. Meanwhile, an anomalous pattern is an object whose characteristics are considerably different than the rest of the data. Anomalous patterns are useful for a variety of applications such as intrusion detection, fraud detection, and failure detection in complex systems. Other types of data mining patterns include trends, change points, subgroups, emerging patterns, etc.

The scope of this thesis focuses on the mining of frequent patterns and anomalous patterns. The limitations of current approaches will be illustrated, which motivate the need to develop new ways for mining such patterns. Furthermore, the theoretical underpinning of the methods developed in this thesis is based on concepts from graph theory.

## 1.1 Frequent Pattern Mining and its Challenges

Frequent pattern mining was originally developed for market basket analysis to discover items (products) that customers frequently buy together. These patterns are also known as *frequent itemsets* in the data mining literature. Each itemset is associated with a quantitative measure called *support*, which is defined as the fraction of total transactions that contain the given itemset. Support is a useful measure because it can be used to eliminate spurious patterns that occur simply by chance. Furthermore, it has certain desirable properties that can be exploited to improve the efficiency of the frequent pattern discovery process.

We can formally define the frequent pattern mining problem as follows:

**DEFINITION 1.1 (Frequent Pattern Mining)** Frequent pattern mining is the task of finding all patterns in the database that satisfy a given minimum support threshold.

In addition to the computational challenges of exploring a massive database to find interesting patterns, the large number of redundant or correlated patterns that can be generated is also a concern. Redundant patterns exist because many of the frequent itemsets generated from transaction data have the same support as their supersets. Discarding such redundant patterns not only helps to reduce the number of generated patterns, it also improves the efficiency of frequent pattern mining algorithms. In the context of binary transaction data, the non-redundant patterns are known as frequent *closed itemsets* [PBT+99a]. Frequent closed itemsets provide a compact yet lossless representation of the frequent itemsets; i.e., it is possible to derive the complete set of frequent itemsets along with their support based on the frequent closed itemsets alone. In particular, the number of frequent closed itemsets can be orders of magnitude fewer than the number of frequent itemsets. The difference is even more pronounced for dense databases, where the frequent itemsets tend to be long and enumerating all of them is simply infeasible due to their exponential number.

Despite these advantages, generating frequent closed itemsets from large transaction databases is computationally expensive. For example, Figure 1.1 shows the relative performance of two state-of-the-art frequent closed itemset mining algorithms<sup>1</sup> when applied to a database, whose size is increased from 100,000 transactions to 5 million transactions. This experiment was conducted on a 2.8 GHz Pentium 4 machine with 4 GB memory. Since both of these algorithms store the entire database in memory using a condensed data representation, they broke down when the database size exceeded a million transactions. A more detailed analysis of the scalability issue is presented in Chapter 3.

---

<sup>1</sup> *Algorithm FPClose has shown to be the fastest algorithm based on the FIMI data mining competition.*

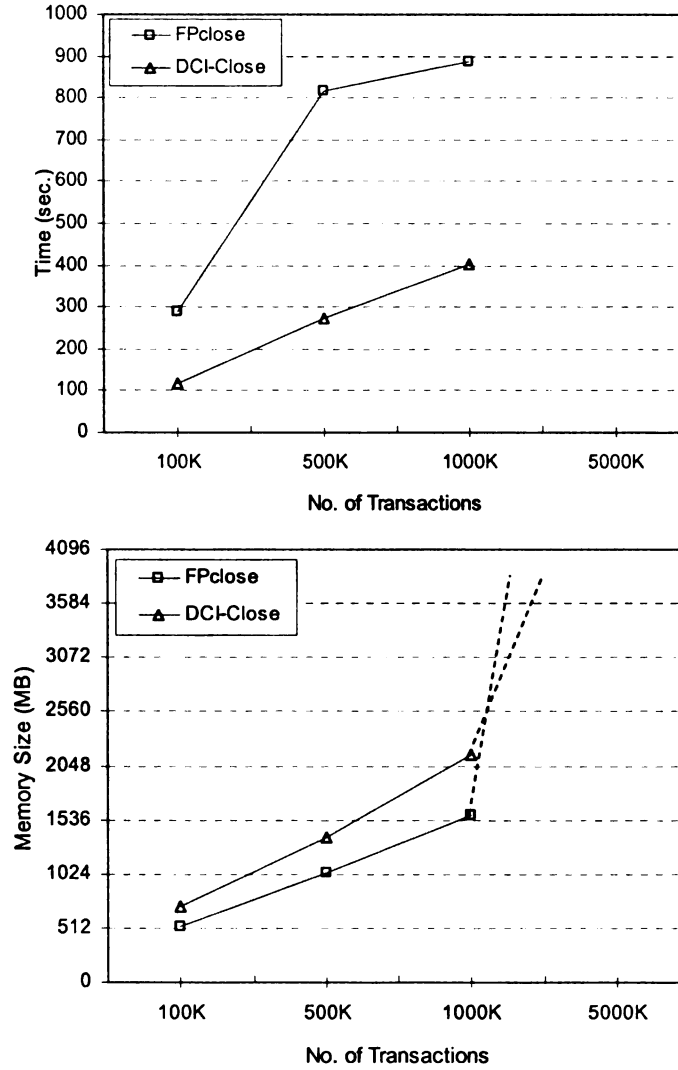


Figure 1.1. Execution time and memory usage for large databases ( $K=1000$ )

In summary, although there have been numerous algorithms developed for frequent closed itemset mining, these algorithms have characteristics that prohibit them from scaling-up to very large databases. Developing efficient and scalable frequent closed itemset mining algorithms is therefore an important research problem that needs to be addressed. Furthermore, any significant improvement to frequent closed itemset mining algorithms will have an impact on other frequent pattern mining problems, such as constraint based mining [HLN99], maximal pattern mining [GZ03], and top-K pattern mining [HWL+02], to name a few. In addition to frequent itemsets in transaction databases, frequent pattern

mining has also been adapted to sequences and graphs. As part of this thesis, we will also illustrate the application of frequent pattern mining to graph anomaly detection.

## 1.2 Anomalous Pattern Mining and its Challenges

Anomalous patterns or outliers are observations that deviate significantly from the majority of the observations in a dataset. Over the years, many outlier detection algorithms have been developed, including statistical-based [Esk00][Lew94], depth-based [JKN98][PS88], distance-based [BS03][JTH01][KNT00][RRS00], and density-based [BKN+00]. While these approaches have proven to be quite effective in some domains, they have several fundamental limitations.

First, existing algorithms typically assume that the anomalous patterns are randomly distributed in the feature space. These algorithms are therefore ineffective when applied to data sets containing small clusters of anomalies, which are often found in many practical applications. For example, attacks in intrusion detection are not isolated random events; they are clustered if they belong to similar classes of intrusions (e.g., variants of a worm). Figure 1.2 shows a synthetic two-dimensional data set, which consists of two clusters of normal observations (labeled as  $C_1$  and  $C_2$ ) and five clusters of anomalies (labeled as  $O_1$  through  $O_5$ ). A density-based anomaly detection algorithm called LOF [BKN+00] was applied to the data set. The data points identified as anomalies by the algorithm are labeled as “+”. Although LOF shows a much higher detection rate compared to traditional distance-based anomaly detection algorithms, Figure 1.2 clearly demonstrates the limitations of the algorithm when applied to data sets containing clusters of anomalies. This is because LOF defines an anomalous pattern based on the density of its predefined neighborhood. If the neighborhood contains fewer points than a minimum threshold, then the observation is declared an anomaly. For data sets with small clusters of anomalies, defining the appropriate neighborhood size and minimum number of points

in a neighborhood is a non-trivial task, especially when the density of the outlying cluster is similar to the density of the normal cluster. Therefore, new data mining algorithms are needed to handle such types of anomalies.

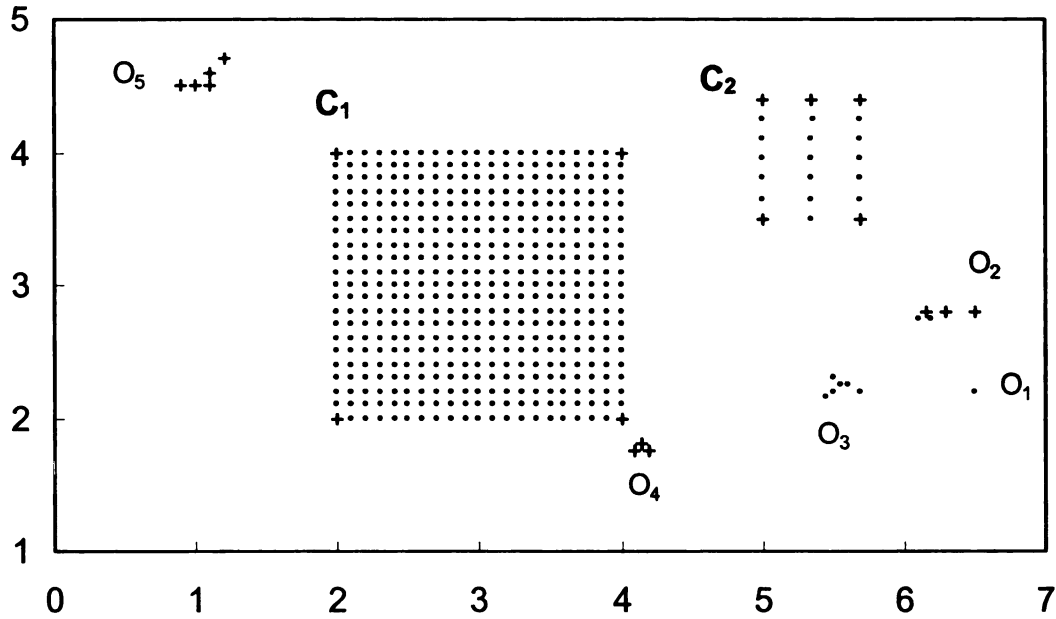


Figure 1.2. Results of applying LOF anomaly detection algorithm on 2D-Data

Second, anomalous pattern detection in complex data is another situation where many existing algorithms fail. For instance, in real-world graph-based datasets such as AIDS antiviral screening data, users are interested in anomalous substructures that correspond to HIV inhibitors, as they help to discover new drugs. Although some of the existing approaches [DKW+05] attempted to address this type of data, their detection rate is still very low even in supervised mode.

As anomaly detection is an essential data mining task, developing efficient algorithms that can achieve significant improvement in detection rate and a lower false alarm rate across these domains is an important research direction and can help numerous applications including intrusion detection, credit card fraud detection, and pharmaceutical research.



## 1.3 Graph-based Approaches for Pattern Mining

As discussed in the preceding sections, there are still many problems with current pattern mining algorithms. In order to provide efficient and robust pattern mining technology to potential users, it is necessary to make significant advances in these pattern mining methods. In this work, our objective is to use a graph based approach for mining such patterns by addressing these existing challenges.

Graph theory, which is a relatively mature field compared to data mining, began in the 18th century with the invention of *Euler's* solution for the *Königsberg Bridge Problem*. Graphs provide a flexible way for modeling complex relationships in data and have been successfully applied in various fields such as networking [ZCM97], image processing [HIK06], pattern recognition [PB06], and machine learning [SWH+05][SHR06][HCL07]. For example, probabilistic graphical models such as Bayesian networks, Hidden Markov Models, and Markov random fields, are widely used model building approaches in machine learning and pattern recognition. These models are developed as a confluence of ideas from graph theory and probability theory.

The attractive feature of graph-based methods is that, once the problem at hand has been abstracted to a relational structure, techniques from graph theory, statistics, and probability theory can be used for purposes of analysis. Graph based methods such as *flow theory* and *random walk* on graphs are important techniques that are already used in data mining applications. For instance, the *PageRank* algorithm used by the *Google* search engine employs the random walk model to calculate the authoritativeness of web pages. Surprisingly, even though graph-based methods have been widely used for model building, there has been very few works on applying these ideas to pattern mining.

This thesis attempts to address the following question: “*Can we improve the effectiveness of pattern mining tasks by incorporating graph theoretic concepts into the min-*

ing process?” Specifically, we attempt to answer this question by examining the applicability of the following graph based concepts:

- *Flow network*: In graph theory, a *flow network* is a directed graph with each edge receiving a flow, where the flow must satisfy a set of constraints such as the capacity of the edges, skew symmetry, and flow conservation. A flow network can be used to model many real-world systems in which a medium propagates through a network of nodes. By transforming a real-world problem into a flow network, it allows us to view the problem from a different viewpoint and enables new theories satisfying the constraints to be developed. This thesis utilizes ideas from flow network theory to develop effective strategies for pruning the search space of the closed itemset mining task, thereby improving both the efficiency and scalability of the mining algorithm.
- *Random walk on a graph*: Random walk is a special case of a *Markov* chain process that enjoys a property called *time-symmetry* or *reversibility*. The basic properties of a random walk are determined by the transition probability matrix associated with the graph that provides the probability of moving from one node to its neighbor. This thesis investigates the effectiveness of using the random walk approach to detect anomalous patterns, particularly for data sets containing small clusters of anomalies. The random walk approach can also be extended to finding anomalies in graph-based data.
- *Markov Random Fields*: Markov Random Field (MRF) is a generalization of *Markov* chains to an undirected graphical model, where the vertices correspond to random variables, and edges represent dependencies between them. Random fields are very attractive from a theoretical standpoint because they allow us to define a probability model based on the dependencies of the graph. A probability distribution over the fields, which corresponds to the *Gibbs* distribution, is also consistent with the *Maximum Entropy* framework. A desirable property of this

framework is that it allows us to build flexible probability models based on features derived from the data.

## 1.4 Contributions

In this work, we develop a class of novel graph based pattern mining techniques. The main contributions of this work are:

1. We develop a graph-based approach for mining frequent closed itemsets. In particular, we introduce a novel data representation, called *PrefixGraph*, containing variable length bit vectors to compress the database. Also, using efficient pruning strategies derived from network flow analysis, we develop a novel algorithm called *PGMiner* to discover the frequent closed itemsets. Our frequent closed itemset mining algorithm is highly scalable and space preserving for very large databases.
2. In order to discover anomalous patterns, we present a *random walk* based algorithm called *OutRank*. Our analysis shows that the technique outperforms traditional density-based and distance-based approaches in terms of their higher detection rates and lower false alarm rates.
3. We systematically develop a graph based framework for mining anomalous patterns in graph datasets. We investigate two different unsupervised approaches. The first approach, called *gRank*, is an extension of the previous *OutRank* algorithm for graph datasets. This method, which is based on substructure similarity, uses the random-walk model for anomaly detection. The second approach, called *gEntropy*, is a probabilistic anomaly detection approach based on the *maximum entropy* principle. Each graph is assigned a probability value representing the likelihood it is generated from a probability model induced from the graph data. The

smaller the probability, the more likely the graph is an anomaly. Finally, we apply the maximum entropy approach to the supervised anomaly detection task. The proposed algorithm, called *gEntropySuper*, uses a frequent subgraph mining algorithm to extract substructure based descriptors and then apply the maximum entropy principle to build a classification model from the frequent subgraphs.

## 1.5 Organization of the Thesis

The rest of the thesis is organized as follows: Chapter 2 illustrates basic concepts, problem definitions, and related work in frequent itemset mining and anomalous pattern mining. Chapter 3 introduces a graph based approach for frequent closed itemset mining. Chapter 4 presents our random walk based anomalous pattern mining algorithm. Chapter 5 discusses our proposed frameworks for mining anomalous patterns in graph datasets. Finally, Chapter 6 concludes with a summary of the thesis contributions and suggestions for future research.

## 2 Background

In this chapter, we illustrate the concepts and terminology used throughout this thesis. More specifically, we discuss preliminaries and definitions of graphs, frequent itemset mining, frequent subgraph mining, and anomalous pattern mining. Further, we discuss related research in both frequent pattern mining and anomalous pattern mining.

### 2.1 Graphs: Basic Concepts

A graph is a mathematical abstraction useful in solving many kinds of problems across several domains. More formally, a *graph*  $G$  is a pair  $(V, E)$ , where  $V$  is a finite set of objects called vertices and  $E$  is a set of 2-element subsets of  $V$  called edges. Also,  $V$  is called a *vertex set* whose elements are called as *vertices or nodes* and  $E$  is a collection of edges, where an *edge* is a pair  $(u, v)$  with  $u, v$  in  $V$ .

If  $(u, v)$  is an edge of a graph  $G$ , then we say that  $u$  and  $v$  are *adjacent* in  $G$ . The edge  $(x, x)$  is called a *self-loop*. A *walk* in a graph is a sequence of vertices and edges such that the vertices and edges adjacent in the sequence are incident. A *path* is a walk in which no vertex is repeated. We denote  $path(u, v)$  as all the edges composed by the walk from vertex  $u$  to  $v$  or vice versa, where  $u, v \in V$ . A *cycle* is a path that begins and ends on the same vertex. A graph with no cycles is called an *acyclic graph*. Furthermore, a graph is *connected* if there is a path between every pair of vertices in the graph.

The *size* of a graph is defined as the number of edges that the graph contains.

In a *directed graph*, edges are ordered pairs, connecting a *source* vertex to a *target* vertex. Moreover, edge  $(u, v)$  is an *out-edge* of vertex  $u$  and an *in-edge* of vertex  $v$  in such a graph. The number of out-edges of a vertex is its *out-degree* and the number of in-edges is its *in-degree*.

In an *undirected graph*, edges are unordered pairs and connect the two vertices in both directions. The number of edges incident to a vertex is called its *degree*.

A *tree* is a special type of a connected graph that has no loops. Also, a *complete* graph is a graph where every pair of distinct vertices is adjacent.

A graph is called a *weighted* graph if each edge has a weight assigned to it.

Let  $G_1$  and  $G_2$  be two graphs and let  $f$  be a function from the vertex set of  $G_1$  to the vertex set of  $G_2$ . Suppose that the following conditions hold:  $f$  is one-to-one and onto, and  $f(v)$  is adjacent to  $f(w)$  in  $G_2$  if and only if  $v$  is adjacent to  $w$  in  $G_1$ . Then we say that the function  $f$  is an isomorphism and that the two graphs  $G_1$  and  $G_2$  are *isomorphic*.

We say that a graph  $G_1$  is a *subgraph* of a graph  $G_2$  if  $G_1$  is isomorphic to a graph, all of whose vertices and edges are in  $G_2$ . Note that by this definition a graph is always a subgraph of itself. Two subgraphs  $G_1$ , and  $G_2$  are said to be *edge disjoint* if they do not have any edge in common.

A *flow network* is a directed graph  $G=(V,E)$  with a *source* vertex  $s$  and a *sink* vertex  $t$ . Each edge has a *capacity* function  $c$ . Also, there is a *flow* function  $f$  defined over every vertex pair, which satisfies the following constraints:

1. Capacity constraint:  $f(u, v) \leq c(u, v) \quad \forall (u, v) \text{ in } V \times V$
2. Skew symmetry:  $f(u, v) = -f(v, u) \quad \forall (u, v) \text{ in } V \times V$
3. Flow conservation:  $\sum_{v \text{ in } V} f(u, v) = 0 \quad \forall u \text{ in } V - \{s, t\}$

The *flow* of the network is the net flow entering the sink vertex  $t$  (which is equal to the net flow leaving the source vertex  $s$ ):  $|f| = \sum_{u \text{ in } V} f(u, t) = \sum_{v \text{ in } V} f(s, v)$ .

## 2.2 Frequent Pattern Mining

In this section, we describe the concept of frequent itemset mining, frequent subgraph mining, and related research.

### 2.2.1 Frequent Itemset Mining

In this section, we present the basic terminology used in the frequent itemset mining.

Let  $I = \{i_1, i_2, \dots, i_m\}$  be a set of  $m$  distinct *items*. An *itemset*  $X$  is considered as a non-empty subset of items; i.e.  $X \subseteq I$ . Moreover, an itemset with  $k$  items is called a *k-itemset*.

A transaction  $t = \{tid, X\}$  is a tuple, where  $tid$  is the transaction identifier and  $X \subseteq I$  is the itemset. A transaction  $t = \{tid, X\}$  is said to contain an itemset  $Y$  if  $Y \subseteq X$ . A transaction database  $D = \{t_1, t_2, t_3, \dots, t_N\}$  is the set of all transactions.

The *support* of an itemset  $X$ , denoted as  $\sigma(X)$ , is defined as the fraction of total transactions that contain  $X$ . Mathematically,  $\sigma(X)$  can be stated as follows:

$$\sigma(X) = \left| \{t_i \mid X \subseteq t_i \wedge t_i \in D\} \right|$$

**DEFINITION 2.1 (Frequent Itemset)** An itemset  $X$  is called *frequent* if  $\sigma(X) \geq \xi$ , where  $\xi$  is a user specified minimum support threshold.

Given a database  $D$  and a minimum support threshold  $\xi$ , the problem of mining for frequent itemsets is to find all itemsets that pass the support threshold.

In particular, the number of frequent itemsets that can be generated from a given database can possibly be very large. In order to address this more compact representation for frequent itemsets called *closed* itemsets have been proposed.

**DEFINITION 2.2 (Frequent Closed Itemset)** An itemset  $X$  is called a *frequent closed itemset* if  $\sigma(X) \geq \xi$ , where  $\xi$  is a user specified minimum support threshold and if there exists no proper superset  $Y \supset X$  with  $\sigma(X) = \sigma(Y)$ .

Similar to the *problem* of frequent itemset mining, the *problem* of mining for frequent closed itemsets is to find all closed itemsets that passes the support threshold.

Table 2.1 Sample database

Transaction ID	Items
1	$a, b, c, d,$
2	$b, d, a, e, f, g$
3	$d, a, b, c, e$
4	$a, c, b, d$
5	$b, c, e$
6	$b, d, e, h$

**EXAMPLE 2.1:** Consider the sample database  $D$  shown in Table 2.1. Here  $\{a, b, c, d, e, f, g, h\}$  is the set of items.  $\{ab\}$ ,  $\{abc\}$ , and  $\{bd\}$  are some of the itemsets. An itemset  $\{abc\}$  is called a 3-itemset. The support of itemsets  $\{bd\}$ ,  $\{abc\}$ , and  $\{abcd\}$  is 5, 3, and 3 respectively. With minimum support threshold  $\xi = 3$ , itemsets such as  $\{abc\}$ ,  $\{bd\}$ ,  $\{abcd\}$  become frequent. Note that itemset  $\{abce\}$  is not frequent since its support count  $(2) < \xi$ . Itemset  $\{abd\}$  with support 4 is closed since none of the supersets of  $\{abd\}$  have support count 4. Note that itemset  $\{abc\}$  is not closed since superset itemset  $\{abcd\}$  has the same support count (3) as  $\{abc\}$ .

During the frequent itemset generation, *anti-monotone* property of frequent itemsets called the *Apriori Principle* can be used to reduce the itemset enumeration search space.

**THEOREM 2.1 (Apriori Principle)** If an itemset  $X$  is infrequent, then all of the itemsets  $Y \supset X$  cannot be frequent.

**PROOF.** Let itemset  $X$  be an infrequent itemset. i.e. it does not support the minimum support threshold  $\xi$ . Now suppose an item  $i$  is added to itemset  $X$  to form itemset  $Y$ . Then the resultant itemset (i.e.  $\{i\} \cup X \supset X$ ) cannot occur more frequently than  $X$ . Therefore, itemset  $Y$  cannot be frequent. ■



In other words, if an itemset is frequent, then all of its subsets become frequent. Using frequent itemsets (or frequent closed itemsets), association rules can be derived. An association rule can be formally defined as follows.

**DEFINITION 2.3 (Association Rule)** An association rule is an implication of the form  $X \Rightarrow Y$ , where  $X$  and  $Y$  are itemsets and  $X \cap Y = \emptyset$ .

The support of a rule denoted as  $s(X \Rightarrow Y)$  is defined as  $\sigma(X \cup Y)/N$ , where  $N$  is the total number of transactions. The confidence of the rule denoted as  $c(X \Rightarrow Y)$  is defined as  $\sigma(X \cup Y)/\sigma(X)$ .

Given a transaction database  $D$ , a minimum support threshold  $\xi_s$ , and minimum confidence threshold  $\xi_c$ , the problem of association rule mining is to find all the association rules that pass both support thresholds.

The complete set of association rules can be mined in two steps: (1) mining the set of frequent itemsets using  $\xi_s$ , (2) derive the association rules on these itemsets using  $\xi_c$ . Here we illustrate this with an example.

**EXAMPLE 2.2:** Consider the sample database  $D$  shown in Table 2.1. For frequent itemset  $\{abcd\}$ ,  $bd$  is a subset with support 5 and the confidence of rule R1:  $bd \Rightarrow ac$  is  $\sigma(abcd)/\sigma(X) = 3/5 = 60\%$ . Notice that association rules R1:  $ac \Rightarrow bd$  and R2:  $abc \Rightarrow d$ , generated from  $\{abcd\}$  have confidence 100%.

## 2.2.2 Frequent Subgraph Mining

In this section, we present the basic terminology used in frequent subgraph mining.

Let  $g = (V, E)$  be a graph, where  $V$  is a finite set of objects called vertices (or nodes) and  $E$  is a set of 2-element subsets of  $V$  called edges. A *labeled graph* is represented by a 4-tuple  $g = (V, E, L, l)$ , where  $L$  is a set of labels, and a label function  $l: V \cup E \rightarrow L$  maps a vertex or an edge to a label.

**DEFINITION 2.4 (Frequent subgraph)** a subgraph  $g$  is said to be *frequent* if the occurrence of  $g$  (i.e.  $\sigma(g)$ ) in a given graph database  $G = \{g_i \mid i=0..n\}$  is greater than or equal to a user specified minimum support threshold  $\xi$ .

Given a database  $G$  and a minimum support threshold  $\xi$ , the problem of *frequent subgraph mining* is to find all frequent subgraphs in the graph database  $G$ .

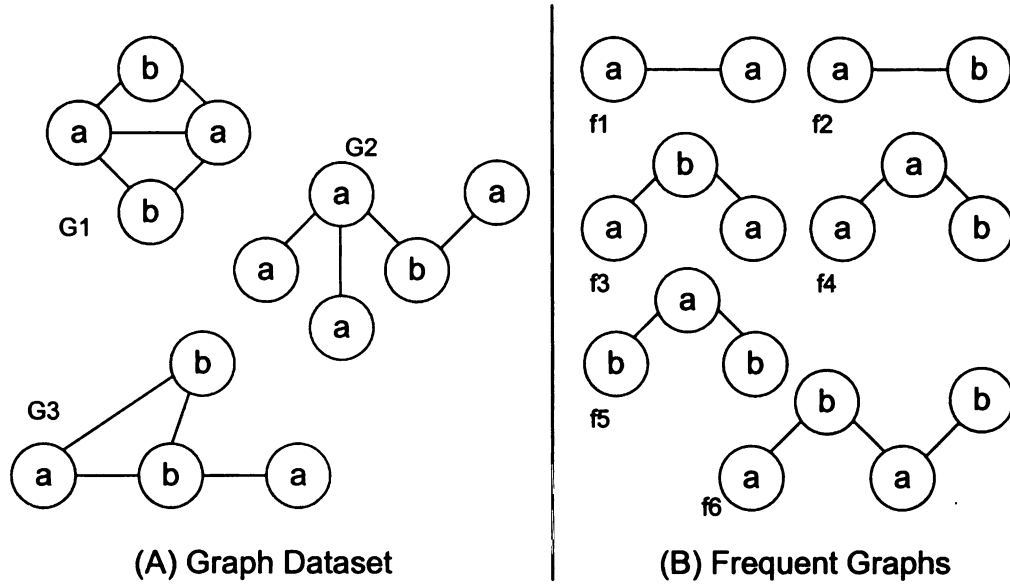


Figure 2.1 Sample graph database and frequent graphs

**EXAMPLE 2.2:** Consider the sample database  $\{G1, G2, G3\}$ , shown in Figure 2.1 (A). With minimum support threshold  $\xi = 2$ , frequent subgraphs  $f1:2$ ,  $f2:3$ ,  $f3:3$ ,  $f4:2$ ,  $f5:2$ , and  $f6:2$ , where  $m:n$  denotes subgraph  $m$  with support count  $n$ , become frequent as shown by Figure 2.1 (B). Here we have six frequent subgraphs of size 1 to 3.

During the frequent subgraph generation, *anti-monotone* property of frequent patterns called the *Apriori Principle* can be used to reduce the subgraph enumeration search space.

### 2.2.3 Related Work

In this section, we first describe the related research in frequent itemset mining, and then describe research in frequent subgraph mining.

The frequent itemset mining problem was first introduced by Agrawal et al. in [AS94b]. Since then, a number of efficient algorithms for mining frequent itemsets have been proposed [HPY00, AIS93, OLP+03]. A popular algorithm is the *Apriori* algorithm [AS94b]. However, one of the major drawbacks of this algorithm is that it requires making several passes over the database. As an alternative, tree projection algorithms [HPY00, AAP+98], where transactions in the database are projected to a lexicographic tree, were proposed. *FP-Tree* [HPY00] is one such algorithm, which creates a compact tree structure and applies partitioned-based, divide and conquer method for mining.

Several concepts have been proposed in the past to address the issue of redundancy in frequent itemsets. The concept of frequent closed itemsets was introduced by Pasquier et al. [PBT+99a]. A level wise algorithm called *A-Close* [PBT+99b] was developed by Pasquier et al., and it employs a breadth first strategy for mining frequent closed itemsets. Since *A-Close* needs to scan the database several times, its performance is quite poor compared to other algorithms on large databases. Algorithms such as *CLOSET* [PHM00] and *CLOSET+* [WHP03] are based on the FP-Tree structure. These algorithms use a mining technique based on recursive conditional projection of the FP-Trees. Non-closed itemsets are pruned using subset checking with the previously mined result set. However, this requires all the closed itemsets previously discovered to be present in memory. Hence the memory usage of these algorithms depends on the number of frequent closed itemsets. *CLOSET+* introduces a new duplicate itemset detection technique—upward checking that is suitable for handling sparse data sets more efficiently. *FPclose* [GZ03] is another algorithm that is based on the FP-Tree structure and uses arrays to efficiently traverse the FP-Tree structure, thereby gaining significant time savings compared to

*CLOSET* and *CLOSET+*. *FPclose* also uses multiple conditional FP-trees for checking the closure of itemsets and achieves better speedup compared to *CLOSET+*, which uses a global tree structure for this task.

The *CHARM* algorithm, which was proposed by Zaki in [ZH02], uses a vertical TID representation. For dense databases, it uses a dual itemset-tidlist search tree and adopts the *Diffset* technique [ZG03] to store the itemset-tidlist at each node of the tree. Similar to *CHARM*, *CloseMiner* [SSM05] also uses frequent closed TID sets. Several alternative algorithms employ the vertical bit vector representation to store the database in a condensed manner. *MAFIA* [BCG01], and *DCI-Close* [LOP06a] are key algorithms in this category. Although they show good performance when mining smaller databases, the length of the bit vector is quite high for large databases. Also, for sparse databases the bit vectors contain mostly zeros and these algorithms spend a lot of time intersecting these regions. To reduce these costs, *MAFIA* uses compressed vertical bit map structures. Also, *DCI-Close* adopts many optimization techniques to reduce the number of bit wise intersections. *DCI-Close* uses closure operation for generating closed itemsets. This technique completely enumerates closed itemsets without duplication and needs no previously mined closed itemsets.

Frequent itemset mining algorithms such as *Apriori*, *FPgrowth*, *DCI\_Close*, *CHARM*, and *FPclose* work well when the main memory of the computer can hold the entire database. When the support threshold is low or the database is very large, main memory may not be able to hold the entire representation of the database. A few studies have addressed this problem and proposed several approaches for mining frequent itemsets out of core, using the secondary memory. One such early approach is the partitioning [SON95], where the database is partitioned to many small databases and frequent itemsets are mined from each such small database. The main problem of this approach is that when the data structure of the candidate itemsets is larger, disk resident data structures are required and therefore significant disk I/O is required to access them. To address this issue,

Grahne et al [GZ04] proposed a recursive projection based partitioning for *FPgrowth* algorithm. In this approach, they recursively project the database and create FP-trees until it fits in main memory. Then, these memory resident FP-trees can be mined using an in-core algorithm. This approach suffers from two major problems: first, they blindly project the database to a FP-tree and if it does not fit in memory, the entire FP-tree constructed so far needs to be deleted and new set of FP-Trees needs to be built starting from the next level itemsets; second, when the *FP-tree* for level-1 itemset does not fit in memory, all combinations of frequent-2 itemsets (level 2) are needed to be projected and this is very costly. Recently, fast disk based frequent itemset mining algorithm was proposed by Buehrer et al [BPG06] for *FPgrowth* approach using the 64 bit computing capabilities available.

The problem of mining frequent closed itemsets out of core is even more challenging, since the closedness of an itemset cannot be decided on the basis of knowledge available in a single data partition. Lucchess et al. [LOP06b] addressed this issue by proposing the first out of core closed itemset mining algorithm. In that they proposed a merging strategy to derive the global closed itemsets from the local closed itemsets mined in each partition. However, their partitioning method of bit vectors is not scalable and incurred many I/O cost for larger databases.

There have been several parallel frequent itemset mining algorithms in the literature [ZEL01], [SK98a], [AS96], [PCY95], [CHN+96], [HKK00], [CHX98], [ZPO+98], [CX98], [SK96]. Most such parallel algorithms are *Apriori* based [AS94b]. Agrawal et al. [AS96] presents three different parallel versions of the *Apriori* algorithm on distributed memory environment. The count distribution algorithm replicates the generation of the candidate set and is a straightforward parallelization of *Apriori*. The other two algorithms (data distribution and hybrid) partition the candidate set among processors. Park et al. [PCY95] uses a similar approach by replicating the candidate set on all processors. Several parallel mining algorithms were presented in [SK98a] for generalized association

rules, and they addressed the problem of skewed transactional data. Cheung et al. [CX98] also addressed the effect of data skewness in their *FPM* (Fast Parallel Mining) algorithm, which is based on the count distribution approach. A parallel tree projection based algorithm, called *MLFPT*, based on *FP-Tree* algorithm is presented in [ZEL01] for a shared memory environment.

In frequent subgraph mining, the goal is to develop algorithms to discover frequently occurring subgraphs in the graph database. Although there are many efficient and scalable frequent pattern mining algorithms exist for itemset mining and sequence mining, developing efficient and scalable algorithms for subgraph mining is particularly challenging because subgraph isomorphism, which is a computationally expensive operation, plays a key role throughout the mining process. Despite that, several efficient subgraph mining algorithms such as *FSG* [KK01], *gSpan* [YH02], *FFSM* [HWP03], *Gaston* [NK04], *SPIN* [HWP04], and *CloseGraph* [YH03] are available and can be used in many practical situations.

In this thesis, we use *FSG* algorithm [KK01] to generate frequent subgraph patterns, which are subsequently used to build anomaly detection model. *FSG* takes a graph database and a minimum support  $\xi$  and generates all connected subgraphs that occur in at least  $\xi\%$  of the graphs. *FSG* follows an *Apriori* style level-by-level approach (breadth first search) to generate subgraph patterns. It starts by enumerating frequent subgraphs consisting of one edge and proceeds to generate larger subgraphs by joining them. At each level, sub graphs are grown by adding one edge at a time. Once a subgraph is generated, its occurrence in the graph database is computed to determine whether it is frequent. *FSG* uses a number of optimization techniques to join subgraphs efficiently, and to compute the frequency of the subgraphs. For more information, readers should refer to [KK01].

## 2.3 Mining Anomalous Patterns

In this section, we present the basic concepts used in mining anomalous patterns. We discuss anomaly detection in two types of input data: data that is described by feature vectors and data that has graph based representation. We also discuss the related research.

### 2.3.1 Basic Concepts in Anomaly Detection

In anomalous pattern mining, the goal is to find objects that have *surprising* or *unusual* occurrence from the majority of objects. Such objects are referred to as anomalies or outliers. Although it remains difficult to give a clear definition of what an outlier is, an often quoted definition of an outlier by *Hawkins* [Haw80] can be stated as follows.

**DEFINITION 2.5 (Outlier)** An outlier is an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism.

Discovering outlier objects can be performed in a *supervised* setting or an *unsupervised* setting. Supervised outlier detection schemes require a training set containing both anomalies and normal objects and use a training mechanism to learn the discriminating features of the database. In situations where training data is not available or rare, outlying objects must be detected by examining the entire dataset. Each object is then considered as an outlier or normal object based on the degree to which that particular object is anomalous. This type of scenario is known as unsupervised learning. Unsupervised learning has both advantages and disadvantages over supervised learning. The major disadvantage is the lack of our known domain knowledge about the problem for the learning algorithm. Nevertheless, this lack of known knowledge can turn out to be an advantage, as it lets the learning algorithm look for patterns that have not previously been considered. In outlier detection this helps to detect unknown anomalies, which is a key challenge for any outlier detection scheme, regardless of the learning method.

To measure the quality of the outlier detection scheme, several metrics can be used. *Precision*, *recall* and *false alarm* rate are the widely used metrics.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$False\ Alarm = \frac{FP}{FP + TN}$$

where  $TP$  ( $TN$ ) is the number of true positives (negatives) and  $FP$  ( $FN$ ) is the number of false positives (negatives).

In the context of outlier detection, *precision* or the detection rate is the number of objects detected as outliers from all the objects available. *Recall* measures the fraction of outlier objects detected. *False alarm* rate is the fraction of normal objects predicted as outlier objects. Obviously, a good outlier detection scheme must have higher precision/recall and lower false alarm rate.

Precision and recall can be combined into one metric called *F-measure*, which can be defined as:

$$F\text{-measure} = \frac{2TP}{2TP + FP + FN}$$

In fact, *F-measure* is the *harmonic mean* of precision and recall.

### 2.3.2 Related Work

In this section, we discuss related research in anomaly detection. First, we consider data objects described by feature vectors; i.e. each instance of data is represented by a list of features or a record (tabular data). Outlier detection for such data has been extensively studied in the past. These existing approaches can be classified into several categories, statistical based, depth based, clustering based, distance based, and density based.



In the statistical-based approach, they assume a standard distribution model (e.g. normal) and flag as outliers those objects that deviate from the model [Esk00][Lew94]. One problem with this approach is that most distribution models apply directly to the feature space and are univariate, which makes this approach unsuitable even for moderate high-dimensional data. Also, for any arbitrary dataset identifying the model that fits well with it can be difficult and expensive.

Depth-based approach is another category based on computational geometry [PS88][JKN98]. Objects are organized in a *convex hull* so that the outliers are more likely to occur in the outer region [PS88]. However, these approaches suffer from the curse of dimensionality.

In the clustering approach, outliers are detected using the clustering algorithms. But these algorithms are not mainly designed for outlier detection and therefore show low detection rate.

The distance-based approach was originally proposed by Knorr et al. [KNT00]. In this approach, an object is considered as an outlier if at least a fraction  $r$  of the objects are located farther than  $q$  of the objects. This notion is extended in [JTH01], by considering outliers as the top- $n$  data points whose distances to their  $k$ -th nearest neighbor are among the highest. This approach is not suitable when the dataset has both dense and sparse regions and it is known as the local density problem.

Density-based approach was proposed by Breuning et al. [BKN+00]. It considers the local density of the object's neighborhood and computes *Local Outlier Factor* (LOF) for each object. The neighborhood is defined by the distance to the  $k$ -th nearest neighbor, and objects with high LOF value are considered as outliers. Although, density based approach does not suffer from local density problem, selecting the right threshold- $k$  is non-trivial, and also this approach is very sensitive to the choice of  $k$ .

In the above approaches, outlier detection has been done for general data, where each object is represented as a set of features. Anomaly detection has been extended for ob-

jects that are represented by graphs. Very few unsupervised anomaly detection algorithms have been developed for graph based data. One such state of the art approach is the compression based anomaly detection scheme [NC03]. Its general idea is as follows: *Subdue*, which is an algorithm to discover repetitive graph patterns, is run in multiple iterations on the graph dataset and after each run, the graph dataset is compressed using the best substructure discovered. Best substructure is determined using the *Minimum Descriptor Length* principle (MDL) [CH00]. Here, every instance of the substructure is replaced by a single new vertex representing it. Then an anomaly score is computed for each graph in the dataset based on the percentage of subgraph that is compressed away. A major advantage of this method is because of the compression of the graphs, subsequent mining becomes much faster and graphs are easy to manipulate. Also, outlier detection has been extended to time sequence of graphs in [IK04]. Here the edge weights vary over time and this forms a time dependent adjacency matrix.

## 3 *PGMiner*: Mining Frequent Closed Itemsets

In this chapter, we introduce a graph based approach for closed itemset mining. The main aspects of this work are summarized below:

- We present a novel *PrefixGraph* representation of a transaction database.
- We develop an algorithm called *PGMiner* (***PrefixGraph Miner***) that uses pruning strategies developed from network flow analysis.
- We perform extensive experiments to compare the performance of *PGMiner* against other existing algorithms on a variety of real and synthetic data sets.

The rest of the chapter is organized as follows: Section 3.1 discusses issues in the existing state-of-the-art frequent closed itemset mining algorithms. Section 3.2 introduces the *PrefixGraph* representation. Closed itemset enumeration approach is discussed in Section 3.3. In Section 3.4, we present our *PGMiner* algorithm. Experimental results are reported in Section 3.5.

### 3.1 Issues in Frequent Closed Itemset Mining

Numerous algorithms have been developed to improve the efficiency of the closed itemset mining task [PBT+99b][PHM00][ZH02][WHP03][GZ03][SSM05][LOP06a]. There are two typical strategies adopted by these algorithms: (1) an effective pruning strategy to

reduce the combinatorial search space of candidate itemsets and (2) a compressed data representation to facilitate in-core processing of the itemsets. *Item merging* and *sub-itemset pruning* are two of the most commonly used strategies employed by current algorithms [WHP03]. These strategies ensure that many of the non-closed itemsets will not be examined when searching for frequent closed itemsets, thereby reducing the runtime of the algorithms.

Apart from the pruning strategies used, having a condensed representation of the database is also vital to achieve good efficiency. Existing algorithms such as *FPclose* [GZ03] and *CLOSET+* [WHP03] construct a frequent pattern tree (FP-tree) structure [HPY00] to encode the relevant itemsets and frequency information. In this representation, each transaction in the database is represented as a path from the root of the FP-tree. Compression is achieved by merging prefix paths of transactions that share the same items. A vertical database representation is another popular strategy, in which each item is associated with a column of values indicating the transactions that contain the item. For example, algorithms such as *CHARM* [ZH02] use vertical tid-lists as their data representation while *MAFIA* [BCG01] and *DCI-Close* [LOP06a] use vertical bit vectors. Figure 3.1 shows the difference between the data compression techniques used to represent the database given in Table 3.2.

Although an FP-tree often provides a compact representation of the data, our analysis shows that there are situations where the storage requirements of the tree may exceed even the database size, especially when the support threshold is low or when the database is very sparse. This is because each tree node encodes not only the item label, but also the support count and pointers to the parent, sibling, and child nodes. As shown in Table 3.1, the size of the initial FP-tree exceeds the database size for databases such as *Chess* and *Kosarak*. Algorithms that use the FP-tree structure must also recursively build smaller subtrees and traverse multiple branches of the trees to collect frequency information dur-

ing the mining process. The overhead of reconstructing and traversing the FP-trees may degrade the overall performance of the algorithm [GZ03].

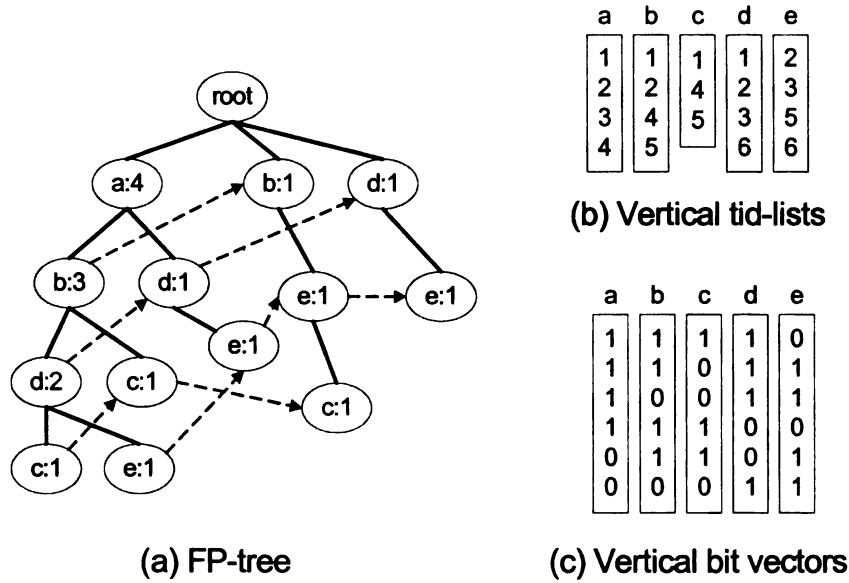


Figure 3.1. Different compressed data representations

Table 3.1 shows that the memory requirements for storing vertical bit vectors are generally less than that for FP-tree and vertical *tid-lists*, with the exception of the *Kosarak* data set. Vertical bit vectors also allow for fast support counting using simple bitwise AND operations. However, when the database is large and sparse, the handling of long bit-vectors is quite inefficient since there are lots of zeros in the vectors.

Table 3.1. Characteristics of various condensed representations

Database Name	Database Size	Min Support	FP-Tree		Vertical Bit Vector Size	Vertical TID-list Size	Size of <i>PrefixGraph</i>
			Num Nodes	Size			
Chess	474.4 Kb	25%	31,812	621 Kb	19.9 Kb	435.3 Kb	239.6 Kb
Pumsb	14.0 MB	45%	183,349	3.5 MB	377.2 Kb	8.58 MB	4.28 MB
WebDocs	1150.4 MB	10%	50,313,644	959.6 MB	52.8 MB	296.5 MB	111.6 MB
Kosarak	36.8 MB	0.08%	3,425,391	65.3 MB	189.3 MB	26.1 MB	9.2 MB

Regardless of the representation, current closed itemset mining algorithms must compare the support of a candidate itemset against the support of its supersets. To perform this task more efficiently, many algorithms such as *CLOSET+* [WHP03], *FPclose* [GZ03], and *CHARM* [ZH02] store their intermediate result set, which contains all the closed itemsets that have been mined so far, in another data structure (FP-Tree, hash table etc.). Such a storage method is feasible as long as the number of closed itemsets is small. When the number of closed itemsets is large, it will consume considerable memory to store and time to search the itemsets. In fact, our analysis shows that in some cases the amount of memory occupied by the result set is several orders of magnitude larger than the size of initial FP-tree or vertical database representation. For example, in the *Chess* database with 25% support threshold, storing the result set takes up to 102MB, even though the size of the initial FP-Tree is only 621Kb! The cost for searching the result set (to determine whether a candidate itemset is closed) can also be very expensive. Our analysis on the *Chess* database shows that *FPclose* spends about 70% of its overall computation time searching the result-set.

In summary, both FP-Tree and vertical representations have their own strengths and limitations. A major limitation of the existing approaches is their poor scalability to large databases. In order to address these limitations, we introduce a novel representation called *PrefixGraph*, which leverages some of the positive aspects of existing representations. From FP-tree, it borrows the idea of projecting a database onto different nodes of a graph—but without the extra cost of traversing multiple branches of the tree to collect frequency information. A *PrefixGraph* also uses bit vectors to encode the database projection at each node. However, the length of its bit vector is considerably shorter than that used by existing vertical bit vector representations. We will discuss in more details how the graph is constructed in the next section. From Table 3.1, note that the size of the *PrefixGraph* structure is moderate compared to other representations. For the *Kosarak* data set, it yields the most compact representation.

## 3.2 *PrefixGraph* Representation

### 3.2.1 Preliminaries

A *PrefixGraph* consists of a set of nodes and a set of directed edges connecting pairs of nodes. Any item in the database that satisfies the support threshold is represented as a node in the *PrefixGraph*. Each node is also associated with a projected bit vector database (see Figure 3.3 for an overview). Before illustrating the *PrefixGraph* structure further, we give some useful definitions.

Note that here, items in a given itemset are assumed to be sorted according to some total order,  $\prec$ . We use the notation  $x \prec y$  to indicate  $x$  precedes  $y$  according to the total order.

**DEFINITION 3.1 (Prefix 2-Item)** At a node  $k$ , an item  $i$  is called its prefix 2-item if  $i \prec k$  and  $\{i, k\}$  is a frequent 2-itemset.

**DEFINITION 3.2 (Prefix Itemset)** Consider an itemset  $X = \{i_1, i_2, \dots, i_{j-1}, i_j, i_{j+1}, \dots, i_n\}$ . A prefix itemset of  $X$  with respect to node  $i_j$  is defined as all the items  $\{i_1, i_2, \dots, i_{j-1}\}$ .

**DEFINITION 3.3 (Suffix Node)** Let  $S$  be a set of nodes sorted based on frequency descending order. A suffix node with respect to node  $j$  is any node  $k \in S$  such that  $j \prec k$ .

**DEFINITION 3.4 (Suffix Link)** A directed edge between node  $i$  and its suffix node  $k$  is called a suffix link.

**DEFINITION 3.5 (Farthest-Node)** Let  $S$  be a set of nodes sorted based on frequency descending order and  $T(j)$  be a set of suffix nodes for node  $j$ . Let  $W(j) \subseteq T(j)$  be a subset of the suffix nodes such that  $\forall n \in W(j), jn$  is a suffix link in the *PrefixGraph* and  $\{j, n\}$  is a frequent 2-itemset. The Farthest-Node of node  $j$  is defined as the suffix node  $k$ , such that  $\forall n \in W(j), n \prec k$ .

**EXAMPLE 3.1:** Consider the *PrefixGraph* shown in Figure 3.3 for the sample database in Table 3.2. The total order of the nodes is  $a \prec b \prec d \prec e \prec c$ . The prefix 2-item for node  $b$  is  $a$ , while the prefix 2-items for node  $c$  are  $a$  and  $b$ . The nodes  $d$ ,  $e$ , and  $c$  are suffix nodes of  $b$  because  $b$  precedes these nodes. The edges  $bd$ ,  $be$ , and  $bc$  are examples of suffix links associated with node  $b$ . Finally,  $c$  is the Farthest-Node of  $b$ .

### 3.2.2 *PrefixGraph* Construction

We now illustrate the construction of the *PrefixGraph* using the transaction database given in Table 3.2 with support threshold  $\xi = 2$ .

Table 3.2. Sample database

Transaction ID	Items	Frequent Items
1	$a, b, c, d,$	$a, b, d, c$
2	$b, d, a, e, f, g$	$a, b, d, e$
3	$d, a, e$	$a, d, e$
4	$i, a, c, b$	$a, b, c$
5	$b, c, e$	$b, e, c$
6	$d, e, h$	$d, e$

First, we scan the database to identify the set of frequent items and their corresponding support counts. These frequent items form the nodes of the *PrefixGraph*. For the sample database, the list of frequent items are  $\langle (a:4), (b:4), (c:3), (d:4), (e:4) \rangle$ , where  $(m:n)$  denotes an item  $m$  with support count  $n$ .

Once the nodes are identified, we order them based on the descending order of support count as shown in Figure 3.2(a). Next, we scan the database again to identify the prefix 2-items for each node. For example, the frequent 2-itemsets for nodes  $a$ ,  $b$ ,  $d$ ,  $e$ , and  $c$  are  $(ab, ad, ac, ae)$ ,  $(ba, bd, be, bc)$ ,  $(da, db, de)$ ,  $(ea, eb, ed)$ , and  $(ca, cb)$ , respectively. The prefix 2-items and their corresponding support counts for these nodes are  $\{\}$ ,  $\{a:3\}$ ,



$\{a:3, b:2\}$ ,  $\{a:2, b:2, d:3\}$ , and  $\{a:2, b:3\}$  respectively. For each node, we store its set of prefix 2-items in a header table as shown in Figure 3.2(b).

The next stage of the graph construction is to store the transactions as bits in the projected bit vector database of the nodes. We scan the database, and for each transaction, infrequent items are removed and the remaining items are sorted based on the frequency descending order. Let  $T$  be the resulting itemset. Now for each item  $k$  in  $T$ , we select the corresponding node  $k$  and compare its prefix 2-items against the prefix itemset of  $T$ . If there is a match, then these matching items are stored as bits in the projected bit vector database of node  $k$ .

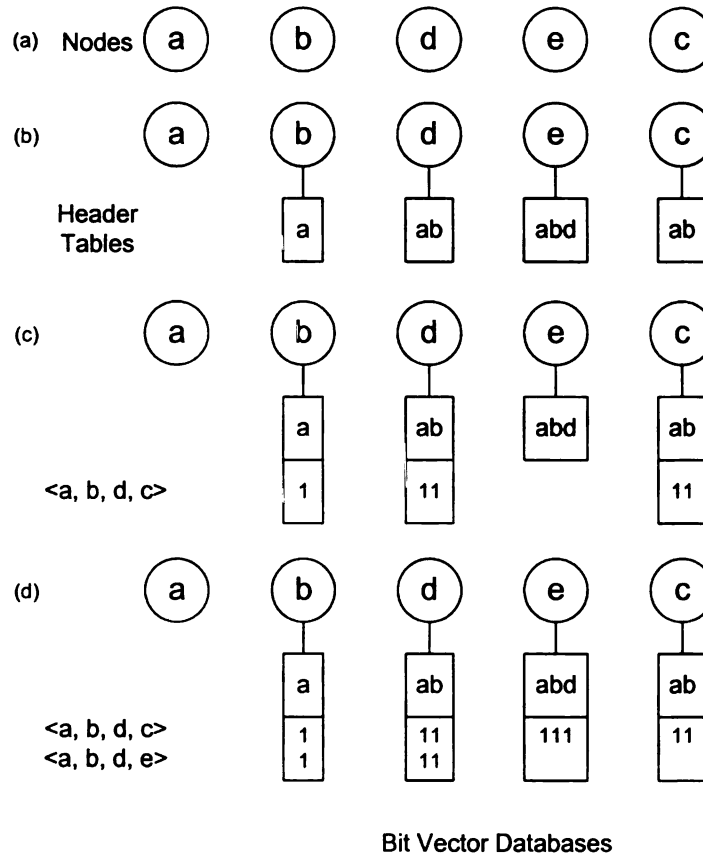


Figure 3.2. *PrefixGraph* construction- a running example

For example, consider the first transaction of the sample database. After removing the infrequent items, the remaining items are:  $T = \{a, b, d, c\}$ . Since the transaction has 4 frequent items we need to consider the nodes  $a$ ,  $b$ ,  $d$ , and  $c$ . Node  $a$  has no prefix 2-items and therefore nothing is stored. For node  $b$ , item  $a$  of the transaction matches with its prefix 2-item (i.e.  $a$ ), and therefore bit  $\langle 1 \rangle$  is stored in its projected bit vector DB. For node  $d$ , items  $\{a, b\}$  of  $T$  match with its prefix 2-items and therefore bits  $\langle 11 \rangle$  are stored in the projected bit vector DB. Similarly for node  $c$ , the bits for items  $\{a, b\}$  of the transaction are stored in the projected bit vector DB. Figures 3.2(c) and 3.2(d) show the *PrefixGraph* after storing the first two transactions. When storing a transaction such as  $\{d, e\}$  at node  $e$ , we need to store bits  $\langle 001 \rangle$  in node  $e$ , since only item  $d$  of the transaction matches with the prefix 2-items of node  $e$ .

In the *PrefixGraph* structure, suffix links are created based on the transactions. For each item  $k$  in the transaction  $T$ , a suffix node  $m$  is selected such that  $m \in T$  and  $\forall n \in$  suffix nodes of  $k$ ,  $m \prec n$ . A suffix link is then created from node  $k$  to node  $m$ . For example, consider the transaction  $\{a, b, d, c\}$ . For node  $b$  we select the suffix node  $d$  (out of  $d$  and  $c$ ) and add a suffix link from  $b$  to  $d$ . Figure 3.3 shows the complete *PrefixGraph* structure after storing all the transactions in the sample database.

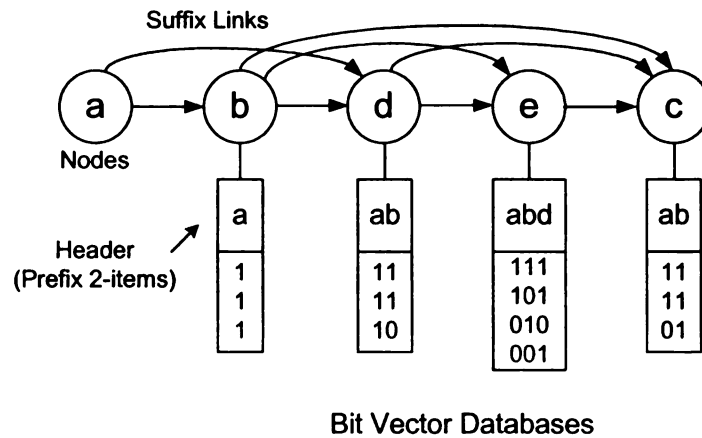


Figure 3.3. *PrefixGraph* representation of the sample database

Instead of explicitly creating links, the links are incorporated directly into the projected bit vector database. More specifically, we can group the bit vectors of the transactions that have the same suffix link together and store them contiguously in the projected bit vector database of the node. For this purpose, the projected bit vector database of each node is partitioned into bins, and the set of bit vectors in each bin corresponds to a suffix link. For example, transactions  $\{a, b, d, e\}$  and  $\{a, d, e\}$  both have the same suffix link  $(de)$  at node  $d$ , and thus, can be stored together in a bin. If a transaction has no suffix link beyond a given node, these transactions are stored in an additional bin called the *terminating bin*. For example, the transaction  $\{a, d, e\}$  is stored in the *terminating bin* of node  $e$ . All bins must be arranged contiguously, so that the intersection of bit vectors (item wise) can be done fast as a one large chunk of words. Also, we need to keep track of the starting location of each bin in order to identify the suffix links.

A summary of the *PrefixGraph* construction procedure is given in Algorithm 1.

---

**Algorithm 1 (*PrefixGraph* Construction)**

---

**Input:** A transaction database  $D$  and support threshold  $\xi$

**Output:** *PrefixGraph* structure

**Method:**

- 1: Scan the database  $D$  and find the frequent 1-itemsets (nodes) and their supports.
  - 2: Sort the nodes in descending order of support.
  - 3: Find the frequent 2-itemsets for each node and create the header tables.
  - 4: For each transaction  $T$ :
    - a. Sort the frequent items in  $T$  in descending order of their support.
    - b. For each item  $k$  in  $T$ , select node  $k$  and match the prefix 2-items of node  $k$  with the items in  $T$  and if there is a match, store the matching items as a bit vector in the bit vector database of node  $k$ .
- 

### 3.2.3 Analysis of *PrefixGraph* Structure

The *PrefixGraph* construction algorithm requires three scans of the database. The first two scans are necessary to find frequent 1-itemset and 2-itemset, while the third scan is needed to construct the projected bit vector databases.

**PROPOSITION 3.1:** *The size of the projected bit vector database of a node is bounded by the support count of the node times the number of prefix 2-items of that node.*

**PROOF.** Let  $m$  be the number of prefix 2-items of a node  $k$ . Since the number of transactions that contain item  $k$  is equal to the support count  $\sigma(k)$ , the size of the projected bit vector database of node  $k$  must be equal to  $\lceil m \times \sigma(k) / 8 \rceil$  bytes. But in a projected bit vector database, projected transactions starting with item  $k$  are not stored as bit vectors at that node. Therefore, the size of the projected bit vector database at node  $k$  is  $\leq \lceil m \times \sigma(k) / 8 \rceil$  bytes. ■

Proposition 3.1 shows an important benefit of the *PrefixGraph* structure as we use bit vector intersections during mining. According to Proposition 3.1, the length of the bit vector is at most equal to the support count of the node. Since the support count of an item is usually much smaller than the database size, we will have shorter bit vectors and accordingly faster bit vector intersections.

The size of the projected bit vector database at each node varies as shown in Figure 3.4 for the *Chess* and *T40I10D100K* databases with minimum support thresholds 30% and 0.10% respectively. Except for the nodes in the middle, all other nodes have small projected bit vector databases, which facilitate faster mining of itemsets.

Now let us theoretically analyze the space complexity of the *PrefixGraph*. Let  $I$  be the number of items in the database and let  $N$  be the total number of transactions. The algorithm for *PrefixGraph* construction has three database scans. First database scan requires  $O(I)$  memory space to store the itemset vector for computing the frequent items (nodes). Suppose  $|F_I|$  is the number of frequent items (size 1) discovered. Then the second database scan requires  $O(|F_I|^2/2)$  to store the upper triangular matrix for frequent 2-itemset computation. Third scan constructs *PrefixGraph* in memory. It has  $|F_I|$  nodes and each node has a header table and a projected database attached to it. Let  $m$  be the maximum number of prefix 2-items of a node and let  $s$  be the maximum support of a node.

Now the header tables require  $m|F_I|$  memory space. Also, all of the  $|F_I|$  projected databases require  $|F_I|(m.s/8)$  memory space (Proposition 3.1). Since we reclaim memory for frequent 2-itemset computation (i.e. matrix) before building the projected databases, only the maximum size of these two steps should be considered as the space requirement. Therefore, the total space complexity of *PrefixGraph* is  $O(I + m|F_I| + \max((|F_I|^2/2), (|F_I|ms/8)))$ .

We also analyze the time complexity of the *PrefixGraph* construction algorithm. Let  $|t|$  be the maximum size of a transaction  $t$  in the database. Now the first scan of the database takes  $O(N|t|)$  to construct the frequent 1-itemsets. The second scan requires us to consider all the combinations of size 2 items in  $t$ ; i.e. it takes  $O(N \binom{|t|}{2})$  time complexity for all  $N$  transactions. The third scan requires each transaction to be sorted and stored in the projected database of the nodes. Using of *quicksort* to sort a transaction takes  $|t|\log |t|$  time. In the worst case a transaction may be stored in all the  $|F_I|$  nodes and the cost is  $|F_I||t|$ . Therefore this step takes  $O(N|t|(\log |t| + |F_I|))$  for all  $N$  transactions. Then, the total time complexity is the sum of the complexities of all these three steps.

### 3.3 Frequent Closed Itemset Mining

In this section, we will study how to efficiently mine frequent closed itemsets from the *PrefixGraph* structure. The algorithm proceeds in two phases: first, we find the frequent closed itemsets for each node (these are known as *local closed* itemsets). We then check whether the local closed itemsets are also *globally closed* using various inter-node pruning techniques. Here we give the formal definitions of local and global closed itemsets.

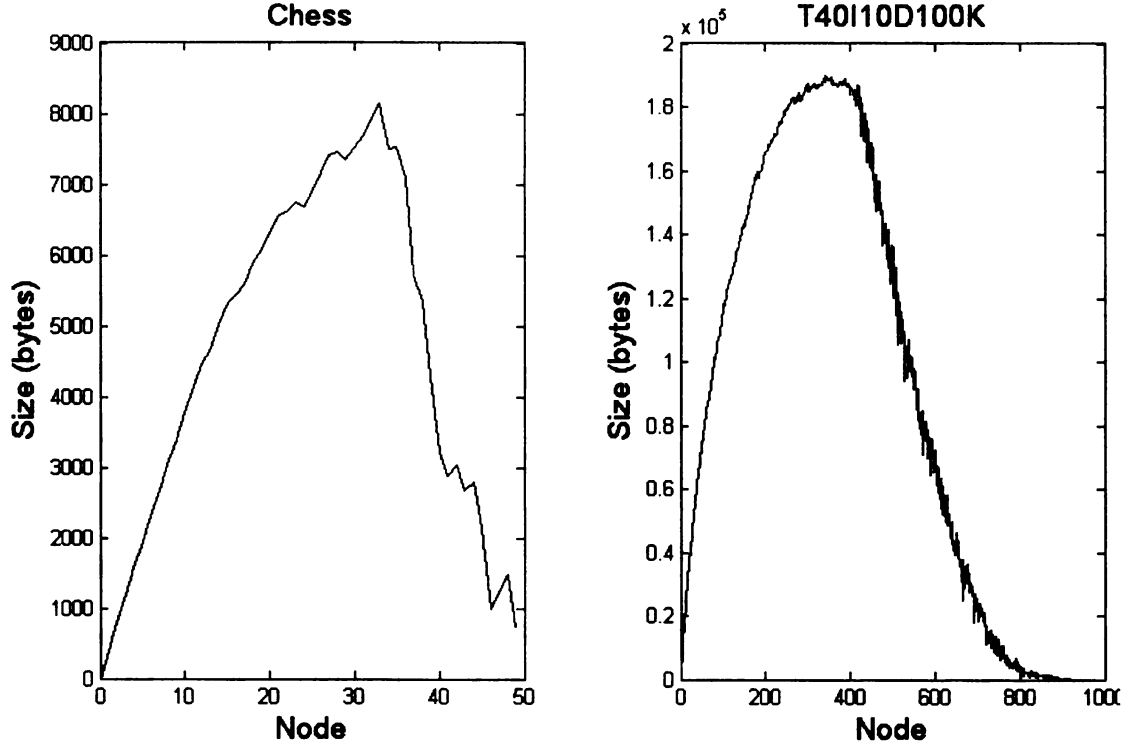


Figure 3.4. Size of bit vector databases at each node

**DEFINITION 3.6 (Local Closed itemset)** An itemset  $X$ , derived under node  $n$  is defined as locally closed, if there is no itemset  $Y (\supset X)$  derived under the same node  $n$  with  $\sigma(Y) = \sigma(X)$ .

**DEFINITION 3.7 (Global Closed itemset)** An itemset  $X$ , derived under node  $n$  is defined as globally closed, if there is no itemset  $Y (\supset X)$  derived under any node  $k$ , ( $k \in \text{set of all nodes}$ ) with  $\sigma(Y) = \sigma(X)$ .

### 3.3.1 Intra-Node Closed Itemset Mining

In intra-node closed itemset mining we mine the locally closed itemsets from the projected bit vector database for each node. As shown in the next proposition, itemsets that are not locally closed are guaranteed to be non-globally closed. Such itemsets can therefore be excluded from further consideration.

**PROPOSITION 3.2:** *For any given itemset  $X$ , derived under node  $n$ , if  $X$  is not locally closed then it is not globally closed.*

**PROOF.** Since  $X$  is not locally closed,  $\exists Y$  derived under node  $n$ , s. t.  $X \subset Y$  and  $\sigma(Y) = \sigma(X)$ . Therefore, by the Definition 3.7,  $X$  cannot be globally closed. ■

In general, to generate frequent itemsets of a node, bit vectors of all distinct pairs of the itemsets are intersected and the cardinality of the resulting bit vector is checked. This is carried out recursively in a depth first manner until all the itemsets are enumerated. For example, in the itemset enumeration tree given in Figure 3.5, for itemset  $\{a\}$ , we generate all its combinations ( $\{ab\}, \{ac\}, \{ad\}$ ). Then, starting from  $\{ab\}$ , ( $\{abc\}, \{abd\}$ ) are generated. If the support of  $\{ab\}$  is identical to the support for one of its immediate supersets, then  $\{ab\}$  will be marked as not closed. Note that any itemset  $\{i_1, i_2, \dots, i_k\}$  generated under a node  $n$  must have its node label appended as  $\{i_1, i_2, \dots, i_k, n\}$ . We have omitted item  $n$  from the set enumeration tree in Figure 3.5 for brevity. Similar to several past algorithms [BCG01, LOP06a, PHM00, ZH02], we also use two additional pruning techniques to rapidly identify the local closedness of the frequent itemset once it is generated.

**PROPOSITION 3.3: (sub-itemset pruning)** *For a frequent itemset  $X$  and an already found closed itemset  $Y$ , if  $X \subset Y$  and  $\sigma(X) = \sigma(Y)$ , then  $X$  and all  $X$ 's descendent itemsets in the set enumeration tree are not closed.*

**PROOF.** Let  $X$  and  $Y$  be frequent itemsets in the set enumeration tree s.t.  $X \subset Y$  and  $\sigma(X) = \sigma(Y)$ . Since  $Y$  is already enumerated according to set enumeration order and found to be closed,  $Y$  can generate itemset  $Y \cup X$ . Since  $\sigma(X) = \sigma(Y)$ ,  $\text{tid-set}(X) = \text{tid-set}(Y)$ , and  $(Y \cup X) \supset X$  itemset  $X$  is not closed. Similarly, any descendent itemset  $X_i$  of  $X$  can be generated by  $Y$  according to set enumeration order and therefore is not closed. ■

**PROPOSITION 3.4: (item merging)** *For a frequent itemset  $X$  and an already found frequent itemset  $Y$ , if the  $\text{tid-set}(X) \subseteq \text{tid-set}(Y)$  and  $Y \not\subset X$ , then  $X$  and all  $X$ 's descendent itemsets in the set enumeration tree are not closed.*

**PROOF.** Let  $Y$  be a frequent itemset already enumerated and let  $X$  be a frequent itemset in the set enumeration tree s.t.  $Y \not\subseteq X$ . According to the itemset enumeration order  $Y$ 's sub tree must contain  $Y \cup X$ , which is already enumerated. So, if  $tid-set(X) \subseteq tid-set(Y)$ , then  $\sigma(Y \cup X) = \sigma(X)$ . Since  $(Y \cup X) \supset X$ ,  $X$  is not closed by definition of the closed itemset. Further, any descendent itemset  $X_i$  of  $X$  can be enumerated by  $Y$  as  $Y \cup X_i$  with the same support count. Thus  $X$ 's descendents itemset are also not closed. ■

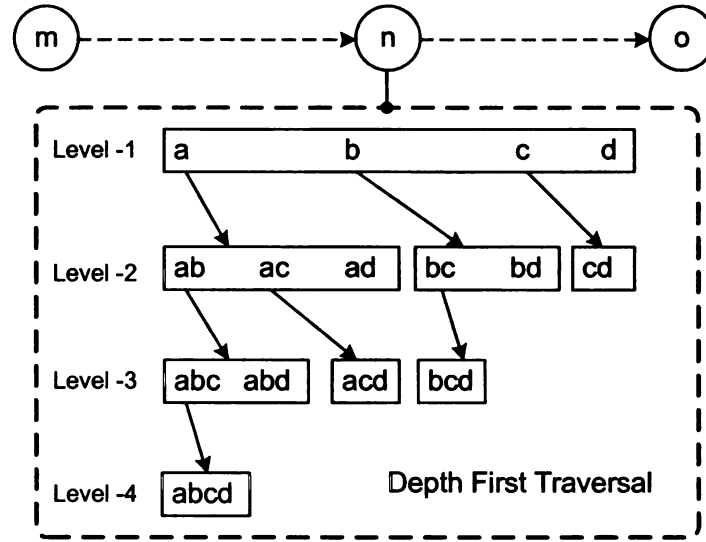


Figure 3.5. Itemset enumeration tree (search space) of a node

A direct implementation of sub-itemset pruning requires storing possibly a large set of closed itemsets and performing subset checking to determine whether an itemset is set-included in a superset. To reduce these overheads, we limit the applicability of this pruning strategy to itemsets between two successive levels of the depth first search space. For example, itemset  $\{ab\}$  at level-2 generates its level-3 itemsets ( $\{abc\}$ ,  $\{abd\}$ ). Based on Proposition 3.3, if  $\{abc\}$  and  $\{ac\}$  have identical support counts, we can prune  $\{ac\}$  and its sub-tree.



The applicability of the item merging proposition to an itemset  $X$  requires that we perform subset checking of  $X$ 's bitmap with the bitmaps of all the processed (i.e. already mined) local itemsets of level-1 down to the parent level of itemset  $X$ . For example, if the itemset is  $\{bcd\}$ , we need to check whether its bitmap is a subset of the bitmap of  $a$ . If it is a subset of  $\text{bitmap}(a)$ , then  $\{bcd\}$  is not closed according to Proposition 3.4. In our vertical bit vector representation, such bitmap subset checking can be performed very fast.

The main advantage of local closed itemset mining is that itemset generation and support counting are very fast, since the projected database contains short bit vectors. Unlike *FPclose* and *CLOSET+*, the information needed for support counting is locally available and there is no need to traverse any other nodes.

Application of both propositions ensures that we generate only the complete set of local closed itemsets for that node. In the next section, we develop an efficient flow based pruning strategy to verify whether the local closed itemsets are also globally closed.

### 3.3.2 Inter-Node Pruning

In order to develop inter-node closed itemset pruning, we consider *PrefixGraph* structure as a network with transactions flowing through the nodes. Therefore, the problem of discovering a global closed itemset can be mapped to a network flow problem.

Let us first analyze the suffix links of the *PrefixGraph* structure. For a node  $n$  in the *PrefixGraph*  $G$ , we define the out-neighborhood and in-neighborhood of  $n$  by  $N^+(n) = \{m \in V(G) \mid (n, m) \in E(G)\}$  and  $N^-(n) = \{m \in V(G) \mid (m, n) \in E(G)\}$ , respectively (here  $V(G)$  and  $E(G)$  are the set of nodes and edges respectively).

For an edge  $(n, m)$  of the *PrefixGraph*  $G$ ,  $f(n, m)$  is the flow along the edge and is considered as the set of transactions that flows through the edge  $(n, m)$ . Furthermore, we have,  $0 \leq |f(n, m)| \leq \sigma(\{nm\})$ , where  $|f(n, m)|$  denotes the number of transactions.

Based on this, for any node  $n$ , its outgoing flow can be defined as  $OutF(n) = \bigcup_{m \in N^+(n)} f(n, m)$  and incoming flow can be defined as  $InF(n) = \bigcup_{m \in N^-(n)} f(m, n)$ . More

specifically, we denote  $f_X(n, m)$  as all the transactions containing itemset  $X$  that flow from node  $n$  to  $m$ . Then, for a given itemset  $X$  derived under node  $n$ , the outgoing flow of  $X$  can be defined as  $OutF_X(n) = \bigcup_{m \in N^+(n)} f_X(n, m)$ . Similarly, the incoming flow of itemset

$X$  derived under node  $n$  can be defined as  $InF_X(n) = \bigcup_{m \in N^-(n)} f_X(m, n)$ . Here we give some

properties of this flow based representation.

**POSTULATE 3.1:** *Given an itemset  $X$  derived under node  $n$ , where  $|X| \geq 2$ , the following properties hold:*

- i.  $\sigma(X) = |InF_X(n)|$
- ii.  $InF_X(n) \supseteq OutF_X(n)$
- iii.  $\forall m: OutF_X(n) \supseteq InF_{Xm}(m)$ , where  $n \prec m$  and  $Xm = X \cup \{m\}$ .

**EXAMPLE 3.2:** Consider the *PrefixGraph* shown in Figure 3.6, for the database given in Table 3.2. The out-neighborhood of node  $b$ ,  $N^+(b) = \{d, e, c\}$  and the in-neighborhood of node  $d$ ,  $N^-(d) = \{b, a\}$ . Transaction flow along edges  $(b, d)$  and  $(d, e)$  are  $f(b, d) = \{t_1, t_2\}$  and  $f(d, e) = \{t_2, t_3\}$  respectively, where  $t_i$  is the transaction ID. The flows can also be defined with respect to a given itemset. For example,  $f_{\{a, b\}}(d, e) = \{t_2\}$ . The incoming flow for itemset  $\{a, d, e\}$  at node  $e$ ,  $InF_{\{a, d, e\}}(e) = \{t_2, t_3\}$  and  $|InF_{\{a, d, e\}}(e)| = 2 = \sigma(ade)$ .

Under the network flow representation, a closed itemset can be defined as follows.

**THEOREM 3.1:** *An itemset  $X$  derived under node  $n$  is globally closed if  $\forall m: InF_X(n) \neq InF_{Xm}(m)$ , where  $n \prec m$  and  $Xm = X \cup \{m\}$ .*

**PROOF.** This theorem is simply a re-statement of the definition of closed itemset that no supersets of  $X$  have the same support as  $X$ . Note that each immediate superset  $Xm$  must be generated at some node  $m$  in the *PrefixGraph* structure and  $\sigma(Xm) = |InF_{Xm}(m)|$ . ■

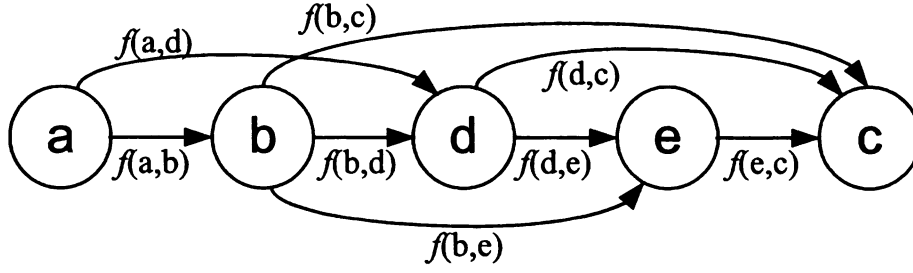


Figure 3.6. Transaction flow network for the sample database

**COROLLARY 3.1:** *The following conditions hold if  $X$  is not globally closed.*

- i.  $\exists m: InF_X(n) = InF_{Xm}(m)$ , where  $n \prec m$ .
- ii.  $\exists m: InF_X(n) \subseteq InF(m)$ , where  $n \prec m$ .

**PROOF.** Condition (i) follows directly from the contra positive of Theorem 3.1. Condition (ii) holds because  $InF_{Xm}(m) \subseteq InF(m)$  (from the definition of incoming flow). ■

Based on this flow based representation, we develop several theorems that will assist us in identifying whether a given local closed itemset is globally closed.

**THEOREM 3.2:** *For any itemset  $X$  derived under node  $n$  if,*

- i.  $X$  is locally closed and
- ii.  $\exists X' \text{ s. t. } X \subset X' \text{ and } X' \text{ is known to be a globally closed itemset under the same node } n \text{ then } X \text{ must also be globally closed.}$

**PROOF.** To construct the proof, by contradiction, assume that  $X$  is not globally closed even though  $X'$  is globally closed. By Corollary 3.1,  $\exists m: InF_X(n) \subseteq InF(m)$ . Since  $X \subset X'$ ,  $InF_X(n) \subseteq InF_{X'}(n)$ . Due to the transitive property of subset relation,  $InF_X(n) \subseteq InF(m)$ , which contradicts the previous statement that  $X'$  is a globally closed itemset. Thus,  $X$  must be globally closed. ■

This theorem states that if we have a globally closed itemset derived under some node, then all locally closed subsets of the itemset are also globally closed (*upward closure* property). For example, in the search space given in Figure 3.5, suppose we found

itemset  $\{acd\}$  is globally closed; then all of the locally closed subsets of  $\{acd\}$  derived under node  $n$  are guaranteed to be globally closed.

Efficient implementation of this theorem requires keeping all of the globally closed itemsets of a particular node in memory for subset checking. To avoid this, we devise two optimization methods: First, when checking the global closedness of local closed itemsets, we start from the maximal closed itemset (leaf) of the enumeration tree. That way, if we determine the leaf itemset as globally closed (using other techniques described later), then all the local closed itemsets in its path (to the root) become globally closed. Second, based on our analysis, we found that there is temporal locality that can be exploited during the search, i.e., most local closed itemsets are subsets of the most recently found globally closed itemset. Therefore, each time we discovered a new globally closed itemset, we keep a copy of this itemset in memory. Then, when a new local closed itemset is found we compare it against this copy.

**THEOREM 3.3:** *For any itemset  $X$  derived under node  $n$  if,*

- i.  $X$  is locally closed and*
- ii.  $|InF_X(n)| - |OutF_X(n)| > 0$*

*then  $X$  is a globally closed itemset.*

**PROOF.** To construct the proof, by contradiction, assume that  $X$  is not globally closed but  $|InF_X(n)| > |OutF_X(n)|$ . By Corollary 3.1,  $\exists m: InF_X(n) = InF_{Xm}(m)$ . From the third property of Proposition 3.1,  $|OutF_X(n)| \geq |InF_{Xm}(m)|$ . Putting them together, it follows that  $|InF_X(n)| \leq |OutF_X(n)|$ , which contradicts our initial assumption. Thus,  $X$  must be globally closed. ■

According to this theorem, if the bitmap of a local closed itemset  $X$ , derived under node  $n$ , has at least one transaction that terminates at node  $n$  (i.e. those transactions do not flow to other nodes), then  $X$  is globally closed. In our *PrefixGraph* structure, all we need to do is to examine the bits in the *terminating* bin of the corresponding itemset's bitmap. If at least one bit is '1' in the *terminating* bin of the itemset, then that itemset is globally

closed. This is a very fast operation that requires checking the itemset's own bit vector to determine the global closedness.

**THEOREM 3.4:** *For any itemset  $X$  derived under node  $n$  if,*

- i.  $X$  is locally closed and*
- ii.  $InF_X(n) = OutF_X(n)$  and*
- iii.  $X$  has exactly one suffix link to node  $m$*

*then  $X$  is not a globally closed itemset.*

**PROOF.** Let  $InF_X(n) = OutF_X(n)$ . Since  $X$  has exactly one suffix link to a node  $m$ ,  $OutF_X(n) = InF_{Xm}(m)$ . Putting them together, we obtain  $InF_X(n) = InF_{Xm}(m)$ , which according to Corollary 3.1 means that  $X$  is not globally closed. ■

Theorem 3.4 suggests that if all of the transactions that belong to itemset  $X$  flow to exactly one other node, then  $X$  is not closed. In the *PrefixGraph* representation, once an itemset is generated its links can be analyzed by checking the bins of the bit vector. Based on the number of links, we can decide whether the itemset is not closed.

For the remaining local closed itemsets in which the previous theorems are inapplicable, we need to test whether they are globally closed. In order to determine the global closedness of a local closed itemset, we need to visit every suffix node and compare the support of its corresponding superset, which is a very expensive operation. The following theorem reduces the number of such nodes that need to be visited.

**THEOREM 3.5:** *Let  $X$  be any itemset derived under node  $n$  and let  $t$  be the Farthest-Node of  $n$  w. r. t. itemset  $X$ . Then for any itemset  $X'$  s. t.  $X' \supset X$  derived under node  $m$ ,  $n \prec m \prec t$ ,  $\sigma(X') \neq \sigma(X)$ .*

**PROOF.** Let  $adj_X(n) = (n_1, n_2, \dots, n_m, t)$  be the possible set of nodes that itemset  $X$  can have a outgoing flow, with  $n \prec n_1 \prec n_2, \dots \prec t$ . Let  $InF_X(n)$  be the transaction flow for itemset  $X$ . Let  $X'$  be an itemset derived under any node  $\in adj_X(n) \setminus \{t\}$ , s. t.  $X' \supset X$ . If  $InF_X(n) \subseteq$  incoming flow for any node  $\in adj_X(n) \setminus \{t\}$  then  $\sigma(X') = \sigma(X)$ . Since  $InF_X(n)$  is

divided among the edges of  $nn_1, nn_2, \dots, nn_m$ , so is the incoming flow of any  $X'$  derived under  $adj_X(n) \setminus \{t\}$  i.e.

$$|InF_X(n_1)|, |InF_X(n_2)|, \dots, |InF_X(n_m)| < |InF_X(n)|$$

Therefore, there is no  $X'$  with  $\sigma(X') = \sigma(X)$ , that can be derived under nodes  $adj_X(n) \setminus \{t\}$ . ■

For a given itemset, this theorem identifies the first possible node that can generate a superset itemset with identical support. So all of the nodes between the current node, where the itemset is generated, and the farthest node w.r.t. the itemset (excluding the farthest node itself) can be ignored. Using this theorem, we can identify the set of nodes that can possibly generate a superset itemset with an identical support for a given itemset  $X$ , derived under node  $n$ , as:  $GEN_X(n) = \{m \in \text{set of nodes} \mid \text{Farthest-Node}_X(n) \prec m \text{ and } \forall \text{ items } j \text{ of } X, j \in \text{prefix 2-items}(m)\}$ .

**COROLLARY 3.2:** *Let  $X$  be an itemset,  $t$  be the farthest node of  $X$  and  $X'$  be the itemset derived under node  $t$ , s. t.  $X' \supset X$ . Then the next possible node  $t'$  that can generate a superset itemset with identical support to  $X$  is the node  $k = \text{Farthest-Node}_X(t)$  if  $k \in GEN_X(n)$ . Otherwise,  $t'$  is the immediate node that precedes  $t$  and  $\in GEN_X(n)$ .*

Although Corollary 3.2 can possibly reduce the size of  $GEN_X(n)$ , it requires us to completely generate the bit vector for each local closed itemset under consideration. In practice this is not very profitable. So, here we design an efficient technique that will examine only the nodes in  $GEN_X(n)$  and avoid complete regeneration of bit vectors at each node.

In this method, to identify the global closedness of an itemset  $X$ , generated under node  $n$ , we visit each node in  $GEN_X(n)$  until we determine its global closedness. Once we visit a node  $k \in GEN_X(n)$ , we can generate the itemset  $Xk$  using its bit vector database and compare the support count with  $X$ . Here we have used two observations in designing this technique:

- i. Most of the local closed itemsets are globally closed

- ii. Local closed itemsets that need a global closedness check under this scenario are mostly the leaves in the search space of the node and therefore they have low support count (and longer itemset length)

These observations suggest that in most situations itemset  $X$  cannot have a superset itemset in a node  $k \in GEN_X(n)$  with an identical support. So, when checking the superset of the itemset  $X=\{i_1, i_2, \dots, i_k, n\}$  derived in node  $n$  under node  $k \in GEN_X(n)$ , we first select item  $n$  and  $i_1$ , and generate the bit vector by intersecting the corresponding bit vectors in node  $k$ . If  $\sigma(ni_1) < \sigma(X)$  then node  $k$  cannot generate itemset  $X$  and we stop processing that node immediately. Otherwise, we intersect each item  $\in X$  in that order, until we determine whether itemset  $X$  can be generated by node  $k$  with an identical support count.

There are several optimization strategies that can be employed here. We found a temporal locality property that can be exploited during the subsequent generation of itemsets in a node. In order to facilitate fast subsequent itemset generation, we keep the bit vectors of the most common subsets of the itemset, once they have been generated under a node for reuse. Here we keep in memory a 2-bits wide bit vector for each node that needs to be checked. The first 1-bit wide vector always saves the bit vector of itemset  $\{ni_1\}$ . In the second bit vector, we keep the bit vector of the most frequent items. For example, if the itemsets that come to a node  $k$  are  $\{abcden\}$ ,  $\{abdfn\}$ , and  $\{abcdgn\}$ , we keep  $\{abdn\}$  in the second bit vector. So, if another itemset, say  $\{abdmn\}$ , needs to be checked under node  $k$ , we can directly use the bit vector of  $\{abdn\}$ . We found this technique to be very profitable.

This itemset regeneration based closedness check is efficient because of the following reasons: first our bit vectors are shorter in length, so that intersection is fast. Second, we keep track of the bit vectors of most common itemsets in memory, which avoids complete regeneration. Third and more importantly, after applying Theorems 3.2-3.4, the remaining percentage of itemsets that needs global closedness is much smaller. We have analyzed this in Section 3.4.

## 3.4 Mining Algorithm

Based on the above discussion, we have the following algorithm for frequent closed itemset mining.

---

### Algorithm 2 (*PGMiner*)

---

**Input:** *PrefixGraph* structure  $G$  and support threshold  $\xi$

**Output:** The complete set of frequent closed itemsets

**Method:**

- 1: starting from the node with highest support count call  $\text{MineNode}(n)$  for each node  $n \in V(G)$ .

#### Procedure $\text{MineNode}(n)$

- 1: use the depth first search paradigm to mine the local closed itemsets at node  $n$  in a top down manner by intersecting its bit vectors. If the support of an itemset is identical to its immediate supersets, then the itemset is marked as not closed. To improve efficiency, use Propositions 3.3 and 3.4 to further prune the non-closed local itemsets.
  - 2: once at a leaf itemset  $X$  of the search path, use Theorems 3.2, 3.3 and 3.4 to detect global closedness for the local closed itemset found. If not detected, search the nodes in  $\text{GEN}_X(n)$  (Theorem 3.5) using the regeneration method.
  - 3: if an itemset is globally closed, mark all the local closed itemsets in the search path to the root as globally closed (by Theorem 3.2). Output any global closed itemset found.
  - 4: stop when all prefix 2-items in the node have been processed, and reclaim memory of the bit vector DB of that node.
- 

The following theorem proves the correctness of the *PGMiner* algorithm.

**THEOREM 3.6:** *PGMiner returns only the closed frequent itemsets.*

**PROOF.** The *PGMiner* algorithm consists of two parts—intra-node local closed itemset enumeration and inter-node pruning of non-globally closed itemsets. By Proposition 3.2, we have shown that all globally closed itemsets must be locally closed, which is why it is sufficient to check only the locally closed itemsets during inter-node pruning.

The intra-node closed itemset mining step is correct because any non-locally closed itemset must be pruned in one of the following ways: (1) by checking the support of an itemset to the support of its immediate superset in the itemset enumeration tree, (2) by



by *sub-itemset pruning*, or (3) by *item merging*. To verify that the inter-node pruning step identifies only the globally closed itemsets, we have previously shown that the theorems used for checking the global closedness of an itemset (i.e., Theorems 3.2 – 3.4) are correct. For the remaining local closed itemsets in which the theorems are inapplicable, the itemset regeneration method is used to test whether they are globally closed. Since this method explicitly visits every suffix node to determine whether the support of a locally closed itemset is identical to the support of one of its supersets, the globally closed itemsets found by itemset regeneration method must be correct. ■

The time required by *PGMiner* depends on the number of nodes in the *PrefixGraph* and the cost of mining itemsets at each node ( $T_{node}$ ); i.e.  $|F|/T_{node}$ . Let  $m$  be the maximum number of prefix 2-items. So, the maximum number of bit vector intersections performed at a particular node is equal to the size of its itemset enumeration tree. In the worst case, this requires  $2^m - 1$  intersections but the optimizations and pruning techniques employed by the algorithm reduce this significantly. For space complexity, *PGMiner* needs to store the projected bit vector database at each node. Since each node contains at most  $m$  prefix 2-items, the maximum number of bit vectors to be stored at each node is equal to  $(m + m - 1 + m - 2 + \dots + 1) = m(m + 1)/2$ . Therefore, the maximum size of the projected bit vector database at a node is  $(s/8)m(m + 1)/2$ , where  $s$  is the maximum support of the node. Note that *PGMiner* reclaims all allocated memory of the current node before mining the next node. Therefore the total space requirement of *PGMiner* is equal to that of a node; i.e.  $O(sm^2)$ .

### 3.4.1 Implementation Techniques

The input database format used by our algorithm is in horizontal representation, where the database is arranged as a set of rows with each row representing a transaction in terms of set of items. Internally we convert each transaction to bit vectors and construct the bit vector DB for each node as described earlier.

Since we use vertical bit vector representation, we need a fast method to count the support of an itemset represented by the bit vector (i.e. number of 1's). We use a lookup table to store the number of 1's of each 16-bit value for all  $2^{16}$  possible 16-bit values. For example, the 16 bit value of '5' has 2 ones, '25' has 3 ones and '10,000' has 5 ones. Also, when intersecting bit vectors we intersect word (32 bits in most architectures) by word. Once a word is intersected, its support count is determined from the lookup table.

### 3.4.2 Memory Management

In this algorithm, we always start mining from the node with the highest support count, i.e. node mining order is from left to right of the *PrefixGraph*. Once all the closed itemsets of a node are discovered, we can safely remove the bit vector DB and reclaim memory space. As shown by Figure 3.4, bit vectors of middle nodes are in general wider and therefore take more memory space in subsequent mining. So reclaim of memory from the earlier nodes facilitates efficient memory utilization.

## 3.5 Experimental Evaluation

In this section, we describe the evaluation environment used to execute our algorithms and the results obtained

### 3.5.1 Evaluating Environment

We compared the performance of *PGMiner* with existing state of the art algorithms in each of the 3 data representation categories. In the vertical TID-list category, we chose *CHARM* [ZH02], which uses the *DiffSet* [ZG03] technique to efficiently compress the database. In the vertical bit-vector category, we chose the *DCI-Close* [LOP06a] algorithm. In the FP-Tree category, we chose the *FPclose* [GZ03] algorithm, which is a

faster algorithm compared to *CLOSET* [PHM00] and *CLOSET+* [WHP03] under the same category. *FPclose* and *DCI-Close* implementations were obtained from *FIMI* repository<sup>2</sup>, while *CHARM* implementation with *DiffSets* was obtained from its author.

We experimented with wide variety of real and synthetic databases as shown in Table 3.3. The *Medical* database contains claims and diagnoses for patients and their prescribed medicine. All other real-world databases were obtained from *FIMI* repository. Synthetic databases were generated using the *IBM Quest* synthetic data generator [Alm].

Our machine environment consists of a 2.8 GHz Intel *Pentium* 4 processor with 1 GB of memory running *Linux*. All recorded execution times refer to real time, which includes CPU time and I/O time.

Table 3.3. Characteristics of the databases

Dataset	No. of Transactions	No. of Items
Medical <sup>3</sup>	5,939,734	5,912
WebView2	77,513	3,340
Chess	3,196	75
WebDocs	1,692,082	5,267,656
Pumsb	49,046	2,113
Kosarak	990,002	41,270
T40I10D100K	100,000	1,000
T100I20D100K	100,000	997
T20I8D500K	500,000	8,612
T50I10DxK	25,000-50,000,000	25,000

### 3.5.2 Performance Comparisons

Execution time comparison of *PGMiner* against other algorithms is shown in Figures 3.7 – 3.15. When an algorithm took a considerably longer time compared to the rest, it was eventually terminated. (Also, run time of *CHARM* for *WebDocs* and *kosarak* datasets could not be recorded, as we had some runtime problems with *CHARM*).

<sup>2</sup> <http://fimi.cs.helsinki.fi>

<sup>3</sup> Database contains visits for 427,214 patients during the period of 2001-2005.

Our analysis shows that in seven out of nine databases tested, *PGMiner* shows the best runtime when compared to all other algorithms at low support thresholds. For the remaining two databases (*Chess* and *Pumsb*), although *PGMiner* outperforms both *FPclose* and *CHARM*, *DCI-Close* shows better runtime. This is because their search space enumeration method seems better suited for these smaller databases. We found that in some cases, all other algorithms fail to mine databases at low support thresholds, while *PGMiner* can still run for even smaller levels of support thresholds.

In summary, *PGMiner* shows better run time performance because it has very low overhead due to the effectiveness of its flow based pruning strategies. Unlike other algorithms, *PGMiner* does not need to store the entire result set in memory. The *PrefixGraph* structure also has shorter bit vectors and this significantly reduces the bit vector intersection cost for large databases. Thus, *PGMiner* has better runtime and can scale to very lower levels of support thresholds.

### 3.5.3 Memory Usage

The memory usage for all the algorithms on several databases is shown in Figures 3.16 - 3.18. We found that *PGMiner* mines all of the databases with low memory usage when compared with the other algorithms. In all these cases, *FPclose* shows higher memory consumption because of its storage based pruning strategy and the large FP-Tree structure it has to build for larger databases. However, *DCI-Close* shows better memory usage when compared with *FPclose* and *CHARM*. But, in some cases (e.g. *T40I10D100K*), its memory consumption gets suddenly high when the threshold is gradually lowered. Note that the memory usage of *PGMiner* does not grow quite as rapidly as other algorithms during the mining process.

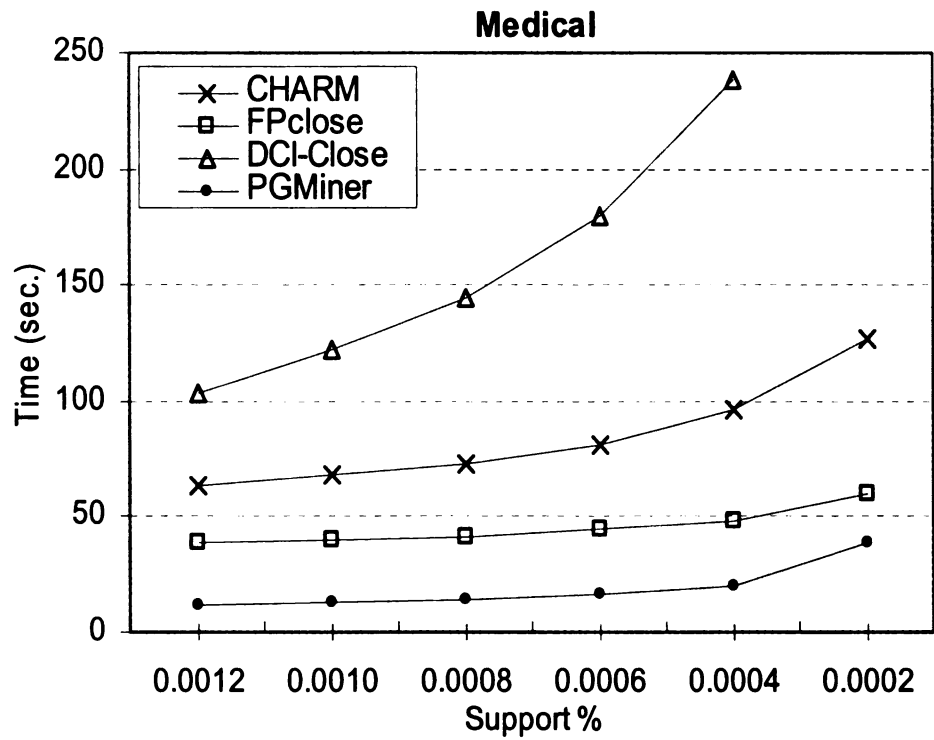


Figure 3.7. Execution time (in seconds) for Medical

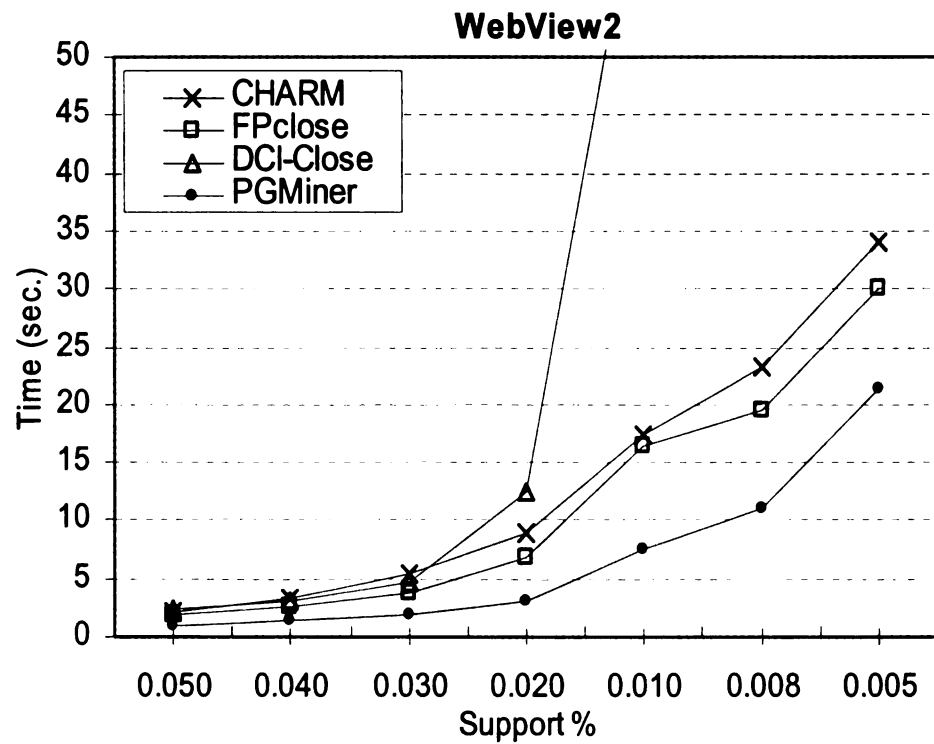


Figure 3.8. Execution time (in seconds) for WebView2

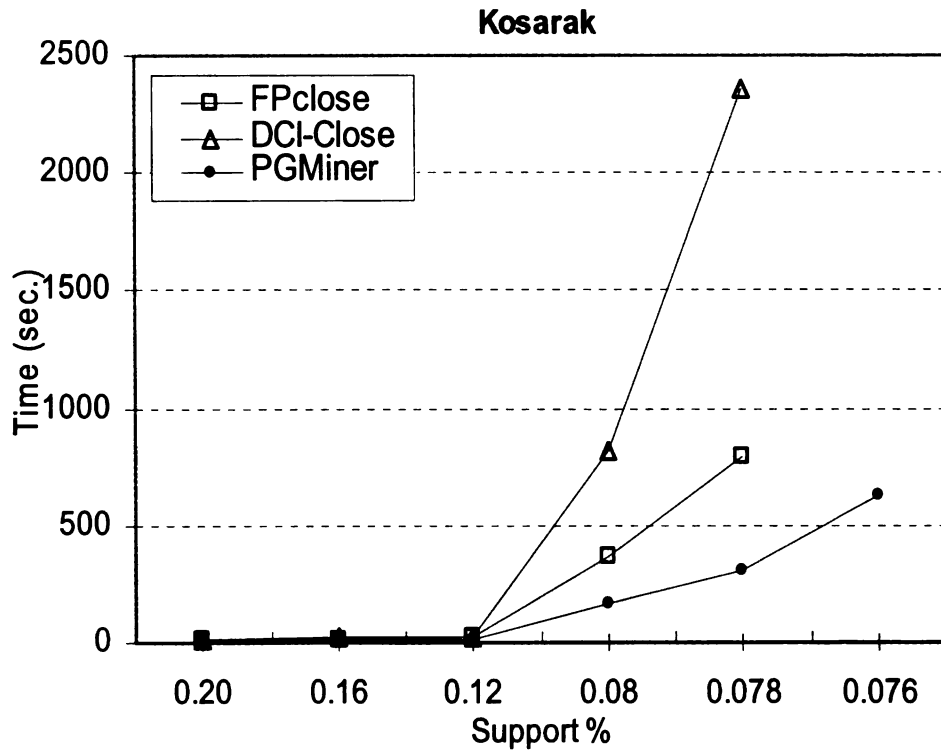


Figure 3.9. Execution time (in seconds) for Kosarak

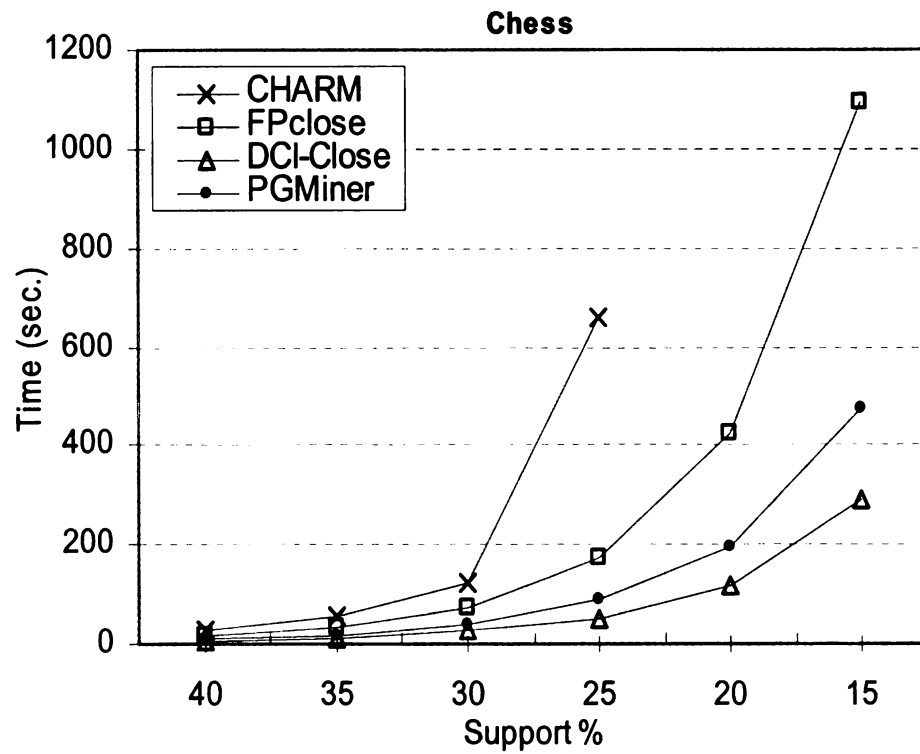


Figure 3.10. Execution time (in seconds) for Chess

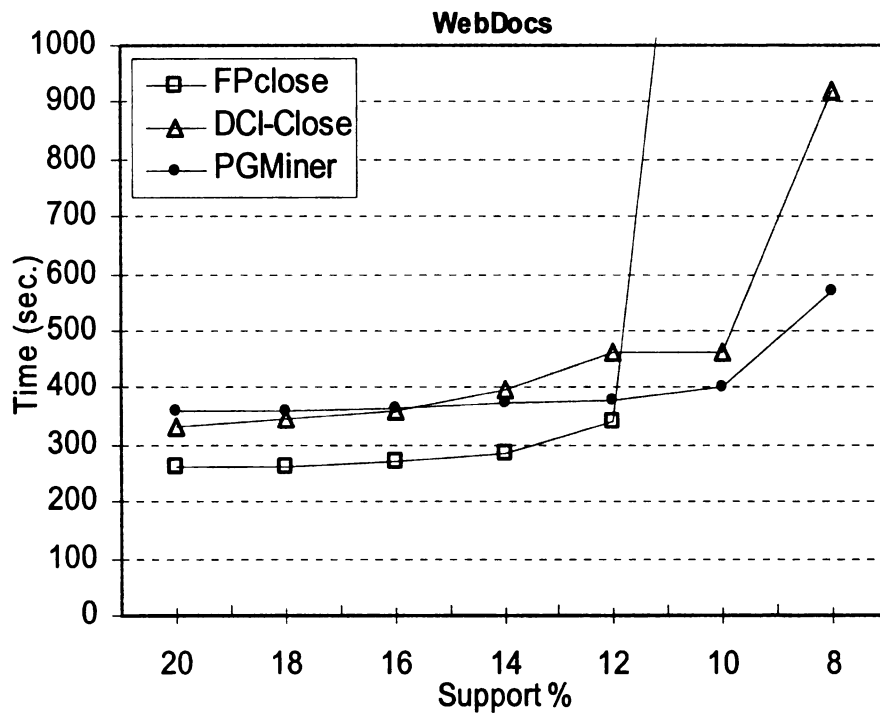


Figure 3.11. Execution time (in seconds) for WebDocs

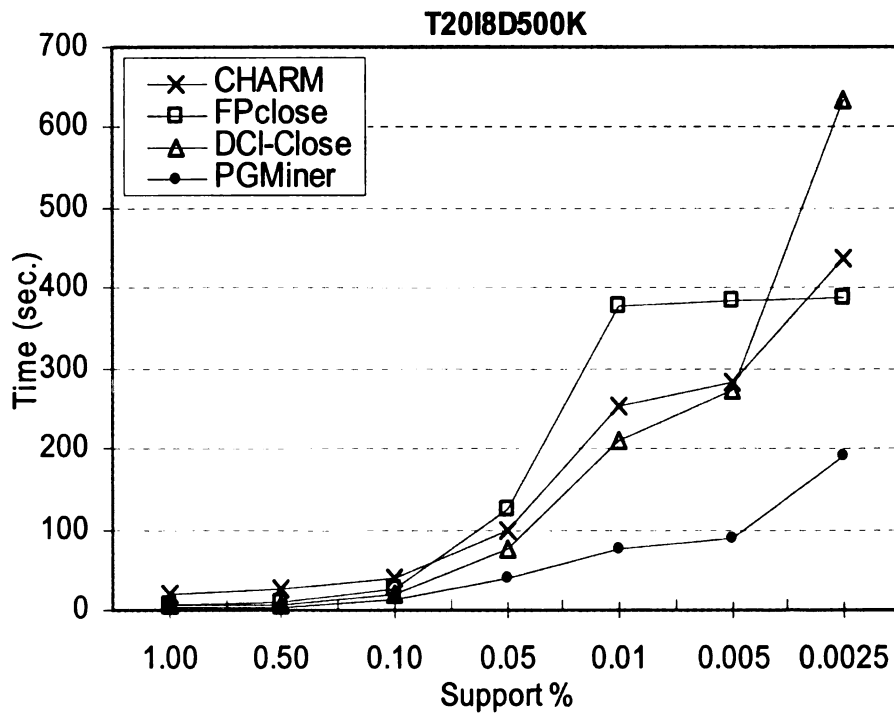


Figure 3.12. Execution time (in seconds) for T20I8D500K

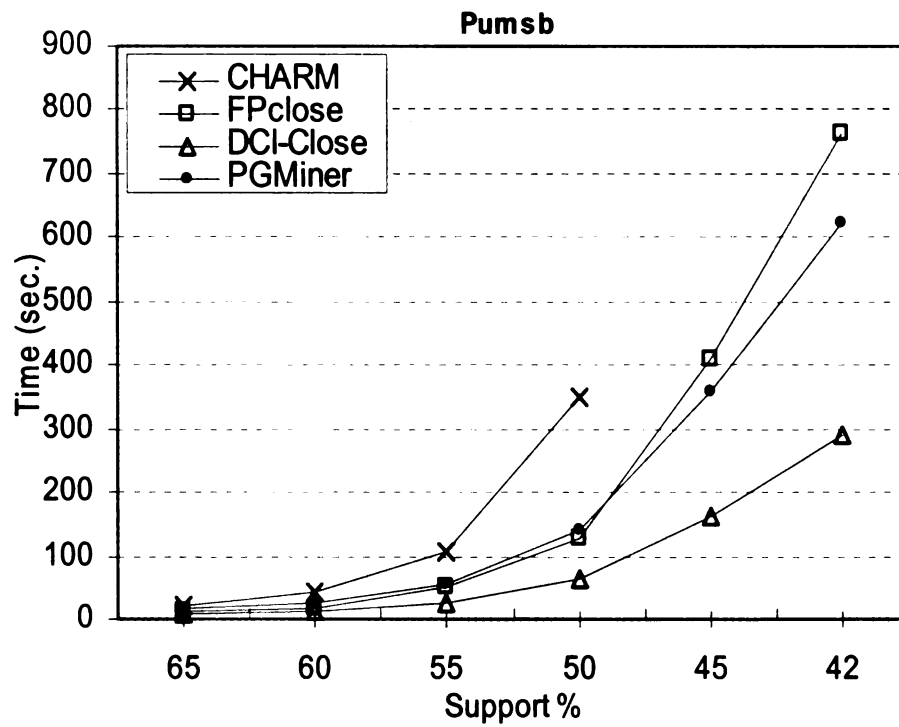


Figure 3.13. Execution time (in seconds) for Pumsb

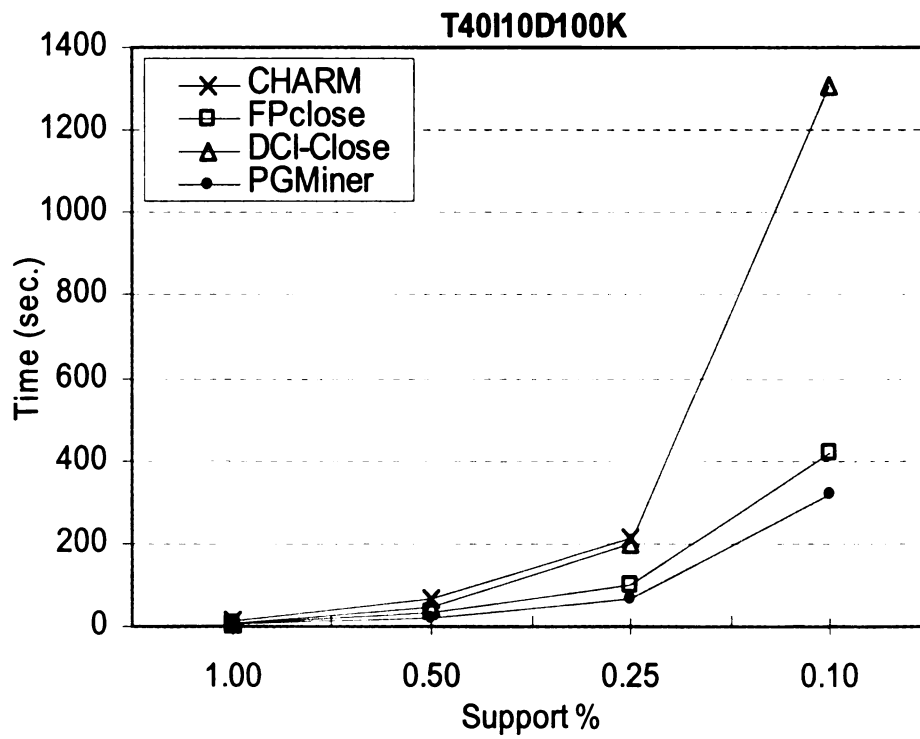


Figure 3.14. Execution time (in seconds) for T40I10D100K



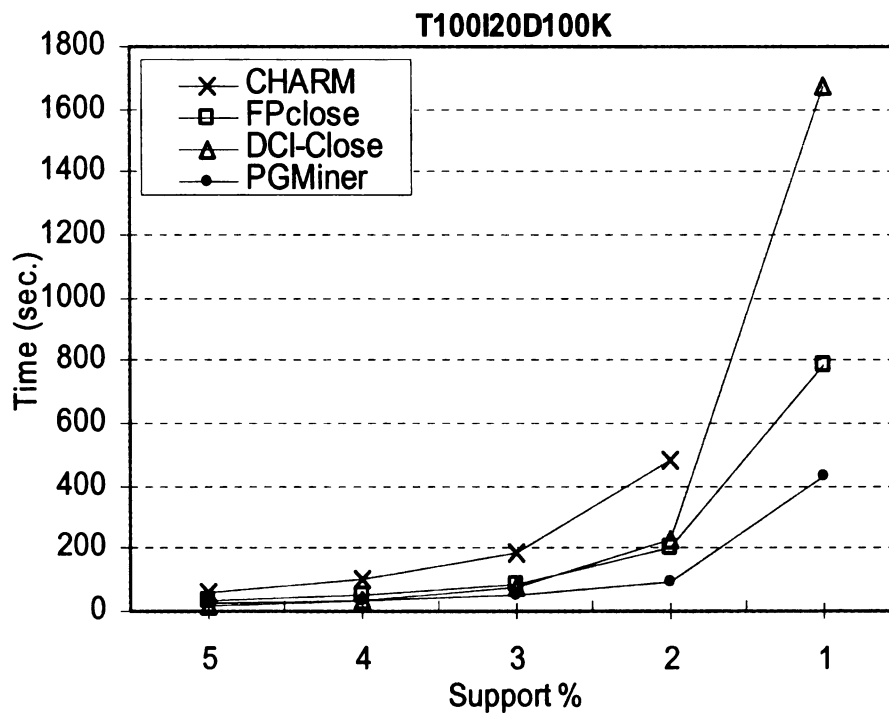


Figure 3.15. Execution time (in seconds) for T100I20D100K

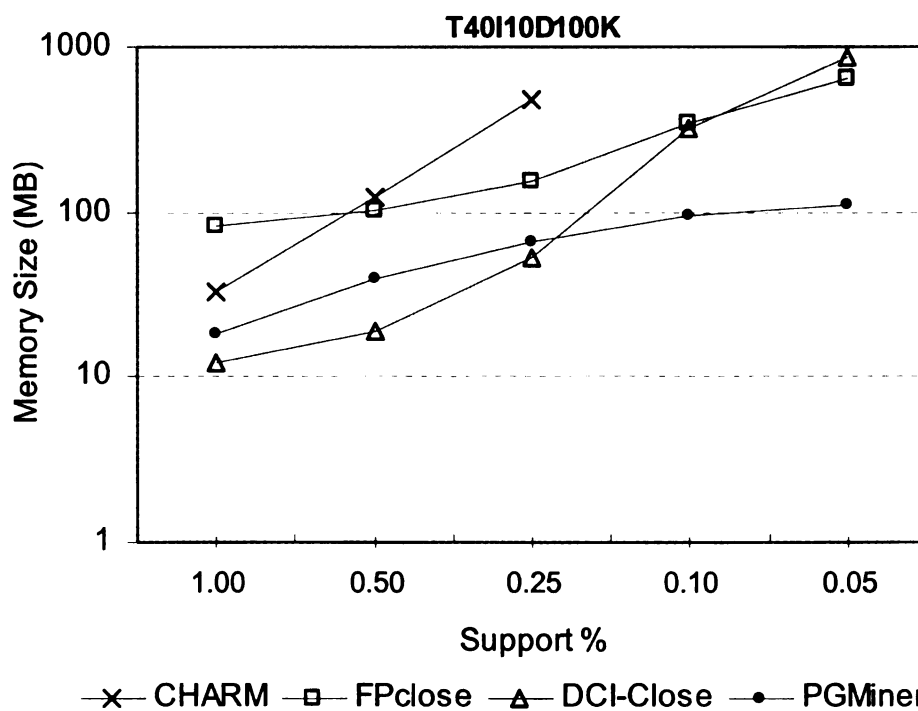


Figure 3.16. Amount of memory (in MB) required for T40I10D100K

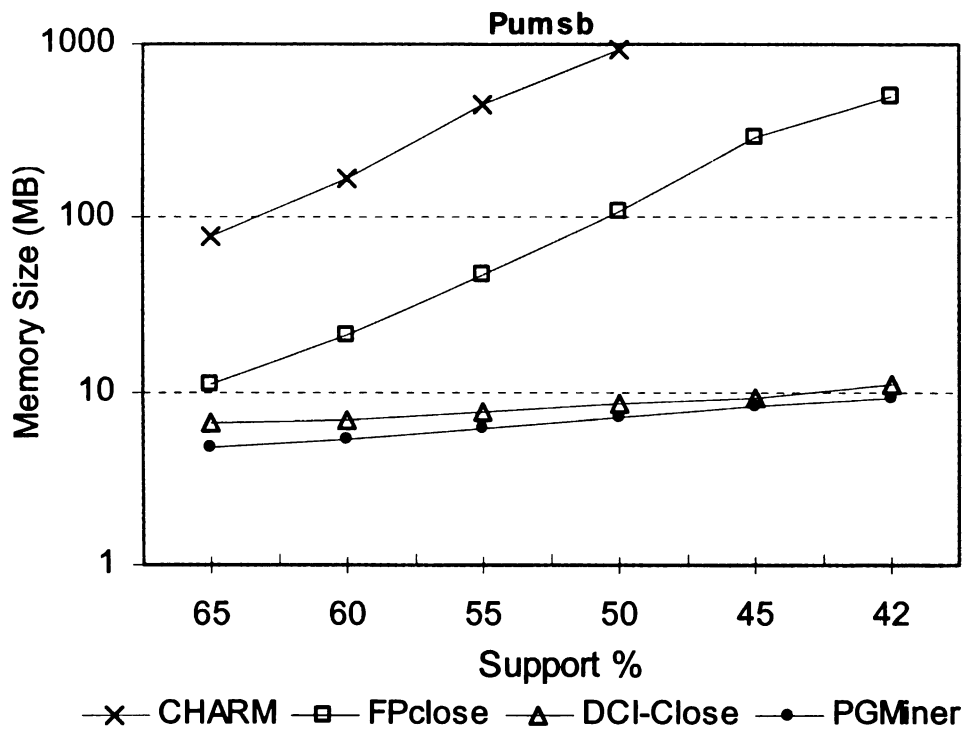


Figure 3.17. Amount of memory (in MB) required for Pumsb

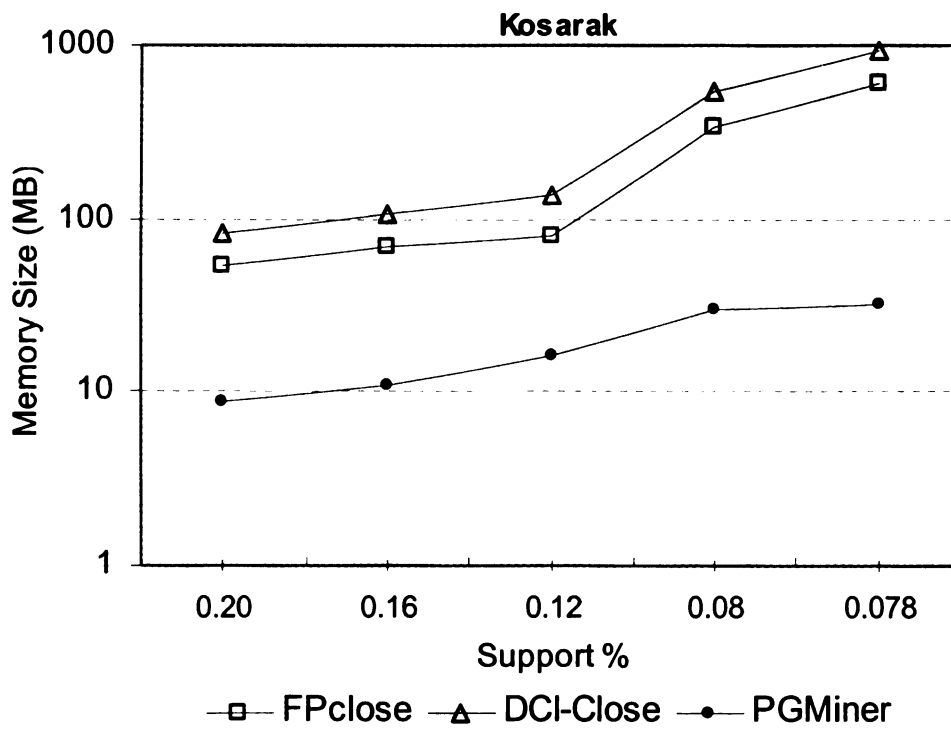


Figure 3.18. Amount of memory (in MB) required for Kosarak

### 3.5.4 Scalability

We have also measured the execution time of all the algorithms by increasing the number of transactions gradually. We use the *T50I10DxK* data set, where  $x$  is varied from 25,000 transactions (DB size 6.9MB) to 50 million transactions (DB size 13.9GB), with minimum support threshold 0.1%. When experimenting with these databases we used a server (2 GHz) with 4GB of memory, since these databases are of gigabyte size. Execution time for all of the algorithms is shown in Figure 3.19. The experimental results revealed that *CHARM*, *FPclose*, and *DCI-Close* could not reach more than 1 million transactions (1000K) of this database set. *FPclose* and *DCI-Close* crashed for the *T50I10D5000K* dataset, and *CHARM* did not finish even after 2 hours.

Analysis of memory usage for these algorithms revealed that they consume high memory space. In the *5000K* dataset, the *FPclose* algorithm fails because it has consumed all the available memory space and it was killed by the system when trying to allocate more memory. See Figure 3.20.

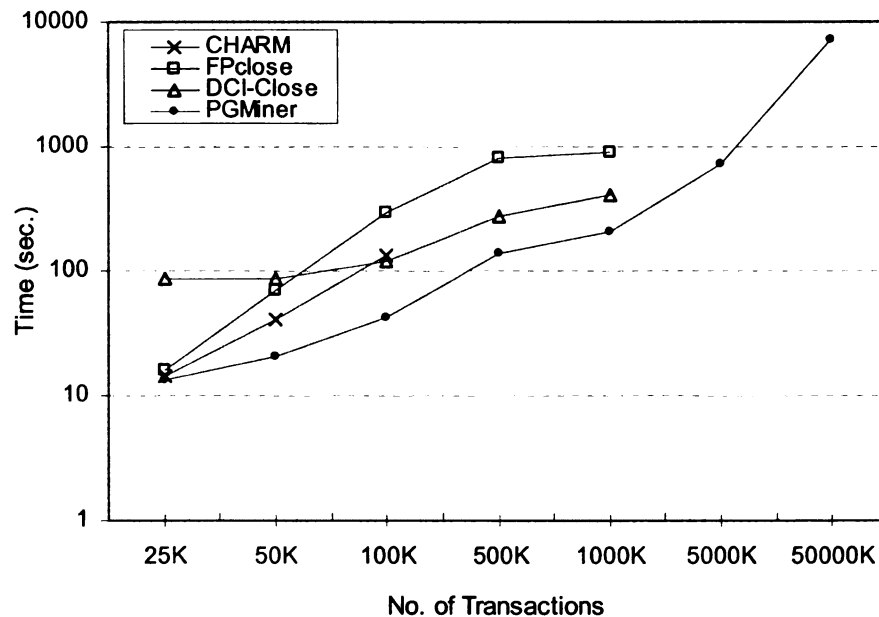


Figure 3.19. Execution time versus number of transactions ( $K=1000$ )

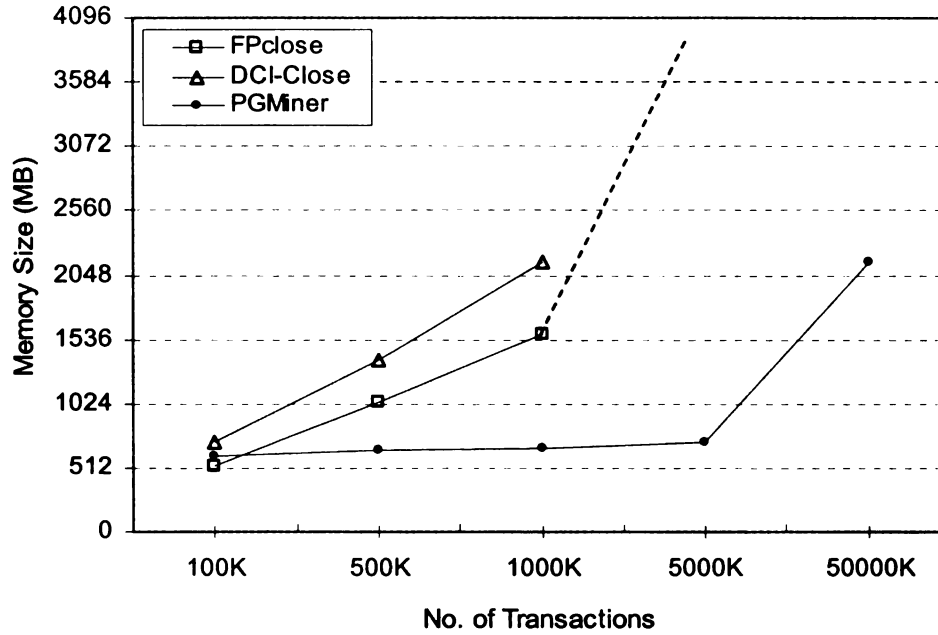


Figure 3.20. Memory usage of algorithms for large databases ( $K=1000$ )

Memory consumption of *DCI-Close* is remarkably high even for the *1000K* dataset, and it was also killed by the system when trying to allocate a larger block in the *5000K* case. Note that *PGMiner* was able to reach 50 million transactions easily showing remarkably low memory usage. As shown in Figure 3.19, *PGMiner* shows impressive scalable performance when mining larger databases.

### 3.5.5 Effectiveness of the Flow Based Pruning

In our algorithm, when a local closed itemset is discovered, we first apply Theorem 3.2 and then if it cannot discover the closedness of the itemset, we apply Theorem 3.3. In Table 3.4 we have shown the percentage of itemsets discovered by both these theorems. For example, In *WebDocs* dataset we were able to discover 94.1% of the total local closed itemsets as either globally closed or not by using Theorem 3.2. From the remaining percentage (i.e. 5.9%), 85.4% of itemsets were discovered by Theorem 3.3. Table 3.4 clearly shows that both Theorem 3.2 and Theorem 3.3 are capable of detecting global closedness of many local closed

itemsets of the database. Moreover, these two techniques can be easily implemented and it is one of the key factors to achieve faster performance in our algorithm.

Table 3.4. Evaluation of the global closedness techniques

Data Set (min. sup.)	Theorem 3.2	Theorem 3.3
Chess (30%)	91.6%	8.0%
WebDocs (10%)	94.1%	85.4%
Pumsb (45%)	91.9%	30.5%
Kosarak (0.08%)	65.5%	68.1%
T20I8D500K (0.01%)	91.7%	26.4%
T40I10D100K (0.1%)	67.6%	40.3%

### 3.6 Summary

In this chapter, we introduced a novel data representation called *PrefixGraph*, which leverages some of the positive aspects of existing representations. From FP-tree, it borrows the idea of projecting a database onto different nodes of a graph—but without the extra cost of traversing multiple branches of the tree to collect frequency information. A *PrefixGraph* also uses bit vectors to encode the database projection at each node. However, the length of its bit vector is considerably shorter than that used by existing vertical bit vector representations.

The size of the *PrefixGraph* structure is quite moderate and its memory requirements do not grow as rapidly as other algorithms. Our proposed algorithm called *PGMiner* employs several effective itemset pruning strategies derived from network flow analysis. These strategies can be adapted to other existing algorithms (such as *CLOSET* [PHM00]) that use projected databases to prune their non-closed itemsets.

Furthermore, *PGMiner* outperforms *FPclose* [GZ03], *DCI-Close* [LOP06a] and *CHARM* [ZH02], three state-of-the-art closed itemset mining algorithms, by an order of magnitude both in time and memory requirements.

## 4 *OutRank*: Mining Anomalous Data

This chapter explores the use of stochastic graph-based method for anomalous pattern mining. The main contributions of this work are as follows:

- We investigate the effectiveness of *random-walk* approach for anomaly detection and show that our proposed framework, called *OutRank* (**Outlier Ranking**), is capable of detecting small clusters of outliers, which is hard to detect by the existing approaches as discussed in Section 1.2.
- We investigate two different approaches for constructing the graph based representation of objects upon which the random walk model is applied.
- We also analyze different choices of similarity measures on the random walk model and compare the performance with existing anomaly detection methods.

The remainder of this chapter is organized as follows. In section 4.1, we discuss issues in the existing anomaly detection methods introduce our solution. Section 4.2 presents our proposed anomaly detection model. Section 4.3 presents several anomaly detection algorithms. We perform an extensive performance evaluation in Section 4.4

### 4.1 Anomaly Detection and its Issues

Anomalies (or outliers) are aberrant observations whose characteristics deviate significantly from the majority of the data. Anomaly detection has huge potential benefits in a

variety of applications (e.g. computer intrusions, surveillance and auditing, failures in mechanical structures, to name a few). Many innovative anomaly detection algorithms have been developed, including statistical-based [Esk00][Lew94], depth-based [JKN98][PS88], distance-based [BS03][JTH01][KNT00][RRS00], and density-based [BKN+00].

These approaches, however, focus mostly on the efficiency of anomaly detection rather than the quality of solution. Therefore, when they are applied to real-world applications across many domains, they show high false alarm rates. For instance, in intrusion detection [KV03], the small clusters of outliers often correspond to interesting events such as denial-of-service or worm attacks. Although existing density-based algorithms show high detection rate over distance-based algorithms for datasets with varying densities, they can be less effective when identifying small clusters of outliers. This is because these algorithms consider the density of a predefined neighborhood for anomaly detection, and in some cases small clusters of outliers with similar density to normal patterns cannot be distinguished.

This chapter explores the use of random walk models as an alternative to previously used anomalous pattern mining algorithms. The heart of this approach is to represent the underlying dataset as a weighted undirected graph, where each node represents an object and each (weighted) edge represents similarity between objects. By transforming the adjacency matrix of the graph into transition probabilities, we model the problem as a *Markov* chain process and find the *dominant eigenvector* of the transition probability matrix. The values in the eigenvector are then used to determine the *outlierness* of each object.

The *random-walk* model is designed to find nodes that are most “central” to the graph. To illustrate, consider graph-*A* shown on the top left panel of Figure 4.1, which consists of 4 nodes connected to a central node labeled as node 1. Upon applying the random walk model to the transition matrix constructed from graph-*A*, the probability score

of each node is plotted on the right hand panel of Figure 4.1. Clearly, node 1 has the highest score compared to other remaining nodes. To illustrate the effect of outliers on the random walk model, consider the graph-*B* shown in Figure 4.1, which is obtained by removing the edges between nodes (3,5) and nodes (4,5) of graph-*A*. Node 5 can be considered as an outlier (anomaly) of the graph. As can be seen from the probability score distribution for graph-*B*, the random walk model assigns the lowest score to the outlying node.

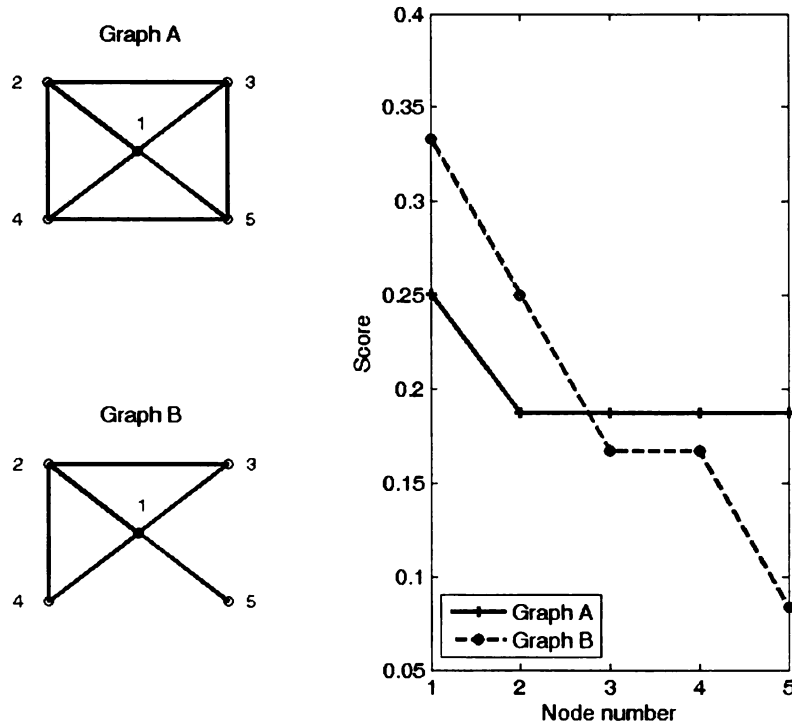


Figure 4.1. Outlier detection with random walk

A major advantage of using our random walk approach is that it can effectively capture not only the outlying objects scattered uniformly, but also small clusters of outliers. This is because random walk based model defines the outlierness of an object with respect to the entire graph of objects; i.e. it views the outlierness from a global perspective. In contrast, existing methods consider a neighborhood to define the outlierness as discussed ear-



lier. Nevertheless, one potential challenge of using the random walk approach is to determine the neighborhood graph from which the outliers can be detected. In the next section, we will show how objects can be modeled as a graph to apply the random walk approach.

## 4.2 Modeling Anomalies Using a Graph

In this section, we develop our framework to discover anomalous objects in a database.

Most anomaly detection schemes adopt Hawkin’s definition [Haw80] of outliers and thus assume that outliers are isolated points far away from other normal points. As such, these outliers can be easily detected by existing distance or density based algorithms. However, in this work we focus on outliers that might be concentrated in certain regions, thus forming small clusters of outliers.

We take a graph based approach to solve this problem. Here we model objects in the database as a graph, where each node represents an object and each edge represents a similarity between them. Each edge is also assigned a weight, which is equal to the similarity between the nodes of the corresponding edge. There are two major issues that need to be addressed: first, how to determine the link structure of the graph based on the similarity of nodes; second, how to discover the outlying objects using this graph model. The following sections describe these issues in detail.

### 4.2.1 Graph Representation

In order to determine the link structure of the graph we compute the similarity between every pair of objects. Let  $X = (x_1, x_2, \dots, x_d)$  and  $Y = (y_1, y_2, \dots, y_d)$  be the vector representation of any two objects drawn from a  $d$ -dimensional space  $R^d$ . While there are many possible choices of similarity measures, we experiment with the following metrics:

**Cosine Similarity.** The similarity between  $X$  and  $Y$  is defined as follows:

$$\text{cosine\_similarity}(X, Y) = \begin{cases} 0 & \text{if } X = Y \\ \frac{\sum_{k=1}^d x_k y_k}{\sqrt{\sum_{k=1}^d x_k^2} \cdot \sqrt{\sum_{k=1}^d y_k^2}} & \text{otherwise} \end{cases} \quad (4.1)$$

**RBF Kernel.** The similarity between  $X$  and  $Y$  is defined as follows (where  $\sigma$  is a user specified kernel width parameter):

$$\text{rbf\_similarity}(X, Y) = \begin{cases} 0 & \text{if } X = Y \\ \frac{1}{\sqrt{2\pi}\sigma^2} e^{-\frac{\|X-Y\|^2}{2\sigma^2}} & \text{otherwise} \end{cases} \quad (4.2)$$

Note that the similarity between an object to itself is set to zero to avoid self loops in the underlying graph representation. Such loops are ignored since they are common to every node, and therefore it is not very useful to distinguish normal objects from outliers.

The relationship between all pairs of objects in the database is represented by the  $n \times n$  similarity matrix  $Sim$ , where  $n$  is the number of objects. We use the similarity matrix to represent the adjacency matrix of the graph. In the graph representation, each node corresponds to an object in the database. Two nodes  $X$  and  $Y$  are connected by an edge if their similarity is greater than zero, and the weight of the edge is taken as  $Sim(X, Y)$ .

#### 4.2.2 Markov Chain Model

Based on the graph representation, we model the problem of outlier detection as a Markov chain process. The Markov chain modeled here corresponds to a random walk on a graph defined by the link structure of the nodes. We hypothesize that under this representation, if an object has a low connectivity to other objects in the graph, then it is more likely to be an outlier.

Connectivity is determined in terms of the weighted votes given by other nodes in the graph. Here higher connectivity nodes convey votes with more weight than that conveyed by the lesser connectivity nodes. The weight of the vote from any node is scaled by the number of nodes adjacent to the source node. The connectivity of a node is computed iteratively using the following expression.

**DEFINITION 4.1 (Connectivity)** Connectivity  $c(u)$  of node  $u$  is defined as follows:

$$c_t(u) = \begin{cases} a & \text{if } t = 0 \\ \sum_{v \in \text{adj}(u)} (c_{t-1}(v) / |v|) & \text{otherwise} \end{cases} \quad (4.3)$$

where  $a$  is its initial value,  $t$  is the iteration step,  $\text{adj}(u)$  is the set of nodes linked to node  $u$ , and  $|v|$  denotes the degree of node  $v$ .

Given  $n$  nodes,  $v_1, v_2, \dots, v_n$ , we can initially assign each node a uniform connectivity value (e.g.  $c_0(v_i) = 1/n$ ,  $1 \leq i \leq n$ ) and recursively apply Equation (4.3) to refine its value, taking into account the modified connectivity values computed for its neighboring nodes. This iterative procedure is known as the power method and is often used to find the dominant eigenvector of a stochastic matrix. Upon convergence, Equation (4.3) can be written in matrix notation as follows:

$$c = S^T c \quad (4.4)$$

where  $S$  is the transition matrix and  $c$  is the stationary distribution representing connectivity value for each object in the dataset. For a general transition matrix, neither the existence nor the uniqueness of a stationary distribution is guaranteed, unless the transition matrix is *irreducible* and *aperiodic*. These properties follow from the well-known *Perron-Frobenius* theorem [IM76].

The transition matrix ( $S$ ) of our Markov model is obtained by normalizing the rows of our similarity matrix ( $Sim$ ) defined earlier:

$$S[i, j] = \frac{Sim[i, j]}{\sum_{k=1}^n Sim[i, k]} \quad (4.5)$$

This normalization ensures that the elements of each row of the transition matrix sum to 1, which is an essential property of a stochastic matrix. It is also assumed that the transition probabilities in  $S$  do not change over time. In general, the transition matrix  $S$  computed from data might not be *irreducible* or *aperiodic*. To ensure convergence, instead of using Equation (4.4), we may compute the steady state distribution for the following modified matrix equation:

$$c = d \cdot \mathbf{1} + (1 - d) S^T c \quad (4.6)$$

where  $S$  is the row normalized transition matrix,  $d$  is known as the damping factor, and  $\mathbf{1}$  is the unit column vector  $[1 \ 1 \dots 1]^T$ . For the proof of convergence of this equation, readers should refer to [S98]. Intuitively, the modification can be viewed as allowing the random walker to transit to any nodes in the graph with probability  $d$  even though they are not adjacent to the currently visited node. As an example, consider the 2-dimensional data with 11 objects shown in Figure 4.2. Clearly object 1 and object 2 are outliers while the rest of the objects are normal.

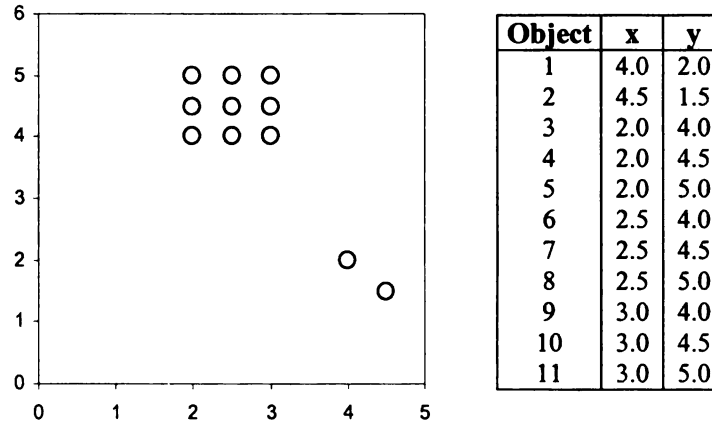


Figure 4.2. Sample 2-D data set

Using uniform probabilities as the initial connectivity vector and after applying Equation (4.6), the connectivity vector converges to its stationary distribution after 112 iterations (where  $d$  is chosen to be 0.1). The final connectivity values and the rank for each object are shown in Table 4.1. Note that object-1 and object-2 are correctly identified as the most outlying objects.

Table 4.1. Outlier rank for sample 2-D dataset

Object	Connectivity	Rank
1	0.0835	2
2	0.0764	1
3	0.0930	5
4	0.0922	4
5	0.0914	3
6	0.0940	9
7	0.0936	7
8	0.0930	6
9	0.0942	10
10	0.0942	11
11	0.0939	8

## 4.3 Anomaly Detection Algorithms

This section describes our proposed algorithm based on the above random walk model for outlier detection. Two variants of the *OutRank* algorithms are presented.

### 4.3.1 *OutRank-a*: Using Object Similarity

In *OutRank-a* algorithm, we form the transition matrix for the Markov chain model using the similarity matrix discussed earlier. We then use the power method to compute the stationary distribution of its connectivity vector. The connectivity vector is initialized to be a uniform probability distribution ( $1/n$ , where  $n$  is the total number of objects) and the

damping factor is set to 0.1. The pseudo code for the OutRank-*a* algorithm is shown below.

---

**Algorithm 3 (OutRank-*a*)**

---

**Input:** Similarity matrix  $\text{Sim}_{n \times n}$  with  $n$  objects, error tolerance  $\varepsilon$ .

**Output:** Outlier ranks  $c$ .

**Method:**

```

1: for i=1 to n do           // forms transition matrix S
2:   let totSim=0.0;
3:   for j=1 to n do
4:     totSim=totSim+Sim[i][j];
5:   end
6:   for j=1 to n do
7:     S[i][j]=Sim[i][j]/totSim;
8:   end
9: end
10: let d=0.1                // damping factor
11: let t=0;
12: let  $c_0 = (1/n) \cdot \mathbf{1}$  // arbitrary assignment,  $\mathbf{1}$ =column vector
13: repeat
14:    $c_{t+1} = (d/n) \cdot \mathbf{1} + (1-d)S^T c_t$ 
15:    $\delta = \|c_{t+1} - c_t\|_1$ 
16:    $t = t+1$ ;
17: until ( $\delta < \varepsilon$ )
18: rank  $c_{t+1}$  from  $\min(c_{t+1})$  to  $\max(c_{t+1})$ 
19: return  $c_{t+1}$ ;

```

---

The space complexity of this algorithm depends on the size of the similarity matrix and the vectors  $C_{t+1}$  and  $C_t$ . So, the space requirement with  $N$  objects is  $(N^2+2N)$ ; i.e.  $O(N^2)$  space complexity. The time complexity of the algorithm depends on the computation of similarity matrix, matrix vector multiplication in the power method, and the sorting method used to sort the stationary vector to rank the outliers. Computation of similarity values takes  $O(N^2/2)$  time complexity for the upper triangular part of the matrix. Power method takes  $O(N^3)$  multiplications and the use of *quicksort* requires  $O(N \log N)$  time complexity. Therefore, *OutRank* has total time complexity  $O(N^3)$ . However, fast computation of random walk method has been proposed in [TFP06]. These techniques can be employed instead of using naïve power method. In this thesis, we focus mostly on the

effectiveness of the anomaly detection task and therefore we use the simplest implementation in our anomaly detection algorithms.

### 4.3.2 *OutRank-b*: Using Shared Neighbors

In *OutRank-a*, nodes are considered adjacent if their corresponding similarity measure is non-zero. So even nodes with low similarity values might be considered adjacent, and that similarity value is used as the weight of the link. In this section, we propose an alternative algorithm called *OutRank-b*, which uses a similarity measure defined in terms of the number of neighbors shared by the objects. For example, consider two objects,  $v_1$  and  $v_2$ . Suppose  $v_1$  has a set of neighbors  $\{v_3, v_4, v_5, v_7\}$  and  $v_2$  has a set of neighbors  $\{v_3, v_4, v_6, v_7\}$  (see Figure 4.3). The set of neighbors shared by both  $v_1$  and  $v_2$  is  $\{v_3, v_4, v_7\}$ . In this algorithm, we take the cardinality of this set as the similarity measure.

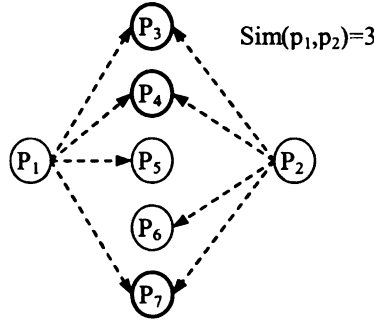


Figure 4.3. Similarity based on the number of shared neighbors

In order to define the shared neighbors we need to find the neighbors of a given object (i.e. adjacent nodes of the graph representation). Here we limit the neighbors only to a set of nodes having high similarity values by using a threshold to cutoff low similarity neighbors. By doing so, outliers will have a fewer number of nodes than the normal ob-

jects in general, and this further helps to isolate outliers. In short, the similarity measure used in OutRank-*b* corresponds to the number of *high-similarity* shared neighbors.

Finding a suitable threshold  $T$  is vital to achieve a higher outlier detection rate. If the threshold is too small, many objects including both outliers and normal ones will have a higher number of shared neighbors, and therefore it will be harder to distinguish outliers. On the other hand, if the threshold is too high, then even the normal objects might have fewer shared neighbors and the algorithm will yield a high false alarm rate. In order to find a suitable threshold, we consider the distribution of similarity values of the corresponding data set. Let  $X$  be the set of similarity values. Let  $\mu$  and  $\sigma$  be the mean and standard deviation of  $X$  respectively. Experimentally, we observe that any  $T$  value within the interval  $[\mu - \sigma, \mu)$  gives higher detection rate; i.e.  $T$  should be chosen to be any value within one standard deviation below the mean.

As can be seen, the choice of the threshold  $T$  depends only on the dataset (i.e. mean and standard deviation of the corresponding data set) and can be automatically derived. Existing algorithms such as *LOF* [BKN+00] and *k-dist* [JTH01] use a threshold called minimum points ( $k$ ) to define the neighborhood. Selection of threshold  $k$  for these approaches is non-trivial and must be specified by the user. Also, unlike in previous approaches where detection rate is sensitive to the threshold parameter, the detection rate of our algorithm is not that sensitive to the value of  $T$ . We will demonstrate this further in the experimental section.

The pseudo code for the OutRank-*b* algorithm is summarized below. The algorithm forms the transition matrix based on the number of shared neighbors between every pair of objects. After computing its transition matrix, the rest of the computation is similar to that of OutRank-*a*.



---

**Algorithm 4 (OutRank-*b*)**

---

**Input:** Cosine similarity matrix  $M_{n \times n}$ , threshold  $T$ ,  
error tolerance  $\varepsilon$ .

**Output:** Outlier ranks.

**Method:**

```
1: for i=1 to n do // discretize M using T
2:   for j=i+1 to n do
3:     if  $M[i][j] \geq T$ 
4:        $M[i][j] = M[j][i] = 1$ ;
5:     else
6:        $M[i][j] = M[j][i] = 0$ ;
7:     end
8:   end
9: end
10: for i=1 to n do // compute new similarity scores
11:   for j=i+1 to n do
12:     let  $X = \{M[i][1] \text{ to } M[i][n]\}$ ;
13:     let  $Y = \{M[j][1] \text{ to } M[j][n]\}$ ;
14:      $\text{Sim}[i][j] = \text{Sim}[j][i] = |X \cap Y|$ ;
15:   end
16:    $\text{Sim}[i][i] = 0$ ;
17: end
18: call OutRank-a ( $\text{Sim}, \varepsilon$ )
```

---

The time and space complexities of OutRank-*b* is similar to those of OutRank-*a*, since we change only the similarity measure in OutRank-*b* compared to OutRank-*a*.

## 4.4 Experimental Evaluation

We have performed extensive experiments on both synthetic and real data sets to evaluate the performance of our algorithms. The experiments were conducted on a SUN *Sparc* 1GHz machine with 4 GB of main memory. The data sets used for our experiments are summarized in Table 4.2. Here  $D$  is the dimension of databases and  $C$  is the number of clusters. Thresholds  $T$ ,  $K_L$ , and  $K_D$  are parameters used with OutRank-*b*, LOF [BKN+00], and K-dist [JTH01] algorithms respectively.

*2D-Data* is the synthetic data set. The rest of the data sets are obtained from the UCI KDD archive. Some of these datasets contain multiple clusters of normal objects. For example, dataset *Zoo* contains 2 normal clusters and a smaller outlier cluster of 13 objects.

We employ several evaluation metrics to compare the performance of our algorithms: *Precision* (P), *False Alarm rate* (FA), *Recall* (R), and *F-measure* (F). In all of our experiments, the number of actual outliers is equal to the number of predicted outliers and therefore  $P=R=F$ .

Table 4.2. Characteristics of the datasets

Data Set	D	C	No. of outliers	No. of instances	T	K <sub>L</sub>		K <sub>D</sub>	
						<i>euc</i>	<i>cos</i>	<i>euc</i>	<i>cos</i>
2D-Data	2	2	20	482	0.93	10	20	10	16
Austra	14	2	22	400	0.25	4	2	5	12
Zoo	16	3	13	74	0.45	40	40	20	24
Diabetic	8	2	43	510	0.80	30	40	30	32
Led7	7	5	248	1489	0.55	330	390	330	220
Lymph	18	3	4	139	0.90	2	2	2	20
Pima	8	2	15	492	0.70	20	2	10	2
Vehicle	18	3	42	465	0.95	50	56	30	14
Optical	62	8	83	2756	0.65	10	2	20	40
KDD-99	38	2	1000	11000	0.35	500	500	5	200

The remainder of this section is organized as follows. In subsection 4.4.1, we evaluate our proposed algorithms against existing approaches, including a distance-based outlier detection algorithm called K-dist [JTH01] and a density-based algorithm known as LOF [BKN+00]. In subsection 4.4.2 we analyze the performance of our algorithms by varying the percentage of outliers. In subsection 4.4.3, we discuss the effect of the shared neighbor approach and in subsection 4.4.4 we analyze RBF kernel for outlier detection. Subsection 4.4.5 presents the effect of threshold selection for OutRank-*b*.

#### 4.4.1 Comparison with Other Approaches

Table 4.3 shows the results of applying various algorithms to synthetic and real life data sets. The K-dist algorithm uses the distance between an object to its *k*-th nearest neighbor

to determine the outlier score. The LOF algorithm, on the other hand, computes the outlier score in terms of the ratio between the densities of an object to the density of its  $k$  nearest neighbors. When experimenting with LOF and K-dist, we used 2 different distance measures: Euclidean distance and  $(1 - \text{cosine})$  distance. In LOF, we have experimented with various  $K_L$  threshold values for each dataset and selected the best  $K_L$  value that maximizes the precision. We did the same for the K-dist algorithm when choosing the threshold  $K_D$  (Table 4.2 shows all such thresholds selected under Euclidean (*euc*) and  $1 - \text{cosine}$  (*cos*) distance measures). So the results reported in this work for LOF and K-dist represent the optimal precision that these algorithms can achieve.

For OutRank-*a*, we chose cosine as the similarity measure. We show in subsection 4.3.4 that the choice of similarity measure (cosine or RBF kernel) does not affect the performance significantly. Meanwhile, the threshold  $T$  for OutRank-*b* is determined empirically from the data using the approach described in Section 3.

Table 4.3. Experimental results

Data Set		OutRank		Euclidean		1 – Cosine	
		<i>a</i>	<i>b</i>	K-dist	LOF	K-dist	LOF
2D-Data	P	1.0000	1.0000	0.8500	0.5500	0.9500	0.5500
	FA	0.0000	0.0000	0.0064	0.0195	0.0022	0.0195
Austra	P	0.7727	0.9545	0.0454	0.1363	0.4545	0.0909
	FA	0.0132	0.0026	0.0555	0.0502	0.0317	0.0529
Zoo	P	0.9230	1.0000	0.7692	0.9230	1.0000	1.0000
	FA	0.0163	0.0000	0.0491	0.0163	0.0000	0.0000
Diabetic	P	0.8837	0.8139	0.7209	0.5813	0.5116	0.4884
	FA	0.0107	0.0171	0.0256	0.0385	0.0450	0.0471
Led7	P	0.9516	0.9799	0.8467	0.2217	0.9758	0.7419
	FA	0.0096	0.0040	0.0306	0.1555	0.0048	0.0516
Lymph	P	0.5000	1.0000	1.0000	0.7500	1.0000	0.7500
	FA	0.0148	0.0000	0.0000	0.0074	0.0000	0.0074
Pima	P	1.0000	1.0000	0.9333	0.9333	0.1333	0.1333
	FA	0.0000	0.0000	0.0020	0.0020	0.0273	0.0273
Vehicle	P	0.6190	0.6428	0.1666	0.3095	0.6190	0.2619
	FA	0.0378	0.0354	0.0827	0.0685	0.0378	0.0733
Optical	P	0.5300	0.6024	0.1686	0.0722	0.2530	0.0482
	FA	0.0145	0.0123	0.0258	0.0288	0.0232	0.0296
KDD-99	P	0.8880	0.8990	0.0080	0.2520	0.2810	0.3510
	FA	0.0112	0.0101	0.0992	0.0748	0.0719	0.0649

First, let us analyze the 2D synthetic dataset, which is designed to view the difference between existing outlier detection schemes and our random walk based method. This dataset (see Figure 1.2) has two clusters ( $C_1$ ,  $C_2$ ) of normal patterns and several small clusters of outlier objects ( $O_1$  to  $O_5$ ). Note that cluster  $C_1$  has a similar density to some of the outlying objects. Both our algorithms successfully captured all of the outliers and delivered a precision of 1.0. On the other hand, LOF was unable to find some of the outlying clusters. Figure 1.2 shows the outlier objects detected by LOF (denoted with ‘+’ symbol). Many of the outlying objects in  $O_1$ ,  $O_2$ , and  $O_3$  regions were undetected. Even worse, it identified some of the normal objects in  $C_1$  and  $C_2$  as outliers.

We have experimented with various  $K_L$  values, but LOF always had problems finding the outliers. When the  $K_L$  value is less than the maximum size of the outlier clusters, then LOF may miss some of the outlying objects, and it identifies some of the normal points in cluster  $C_1$  as outliers. This is because LOF computes the outlier score of an object by taking the ratio between the densities of an object and its  $K_L$ -th nearest neighbor, and if the neighborhoods under consideration for an object in  $C_1$  and in some outlying cluster (say  $O_3$ ) have similar densities, then it is difficult to distinguish outliers since in this case the outlier scores computed by LOF can be similar for both objects. Also, when a larger  $K_L$  value is used, it can possibly identify normal objects in cluster  $C_2$  as outliers. On the other hand, distance based algorithms such as K-dist suffer from local density problem as described in [BKN+00]. Therefore, they fail to identify outlier clusters such as  $O_4$ . As a result, these existing approaches break down and deliver a higher false alarm rate.

When considering the real life data sets such as *Optical* (hand written data), *kdd-99* (intrusion data) and *Austra*, both density and distance based algorithms performed poorly. We speculate that there may be many small clusters of outliers in those datasets and since these algorithms are less effective in identifying such outliers, their performance becomes low. Our algorithm performed significantly better than both LOF and K-dist with a lower false alarm rate, since as expected the random walk model can handle this situation well.

Also, when analyzing the datasets with several clusters of objects such as *Led7* and *Optical*, performance of density based algorithms became low. In both these datasets, our algorithm showed better performance. Also, as shown in Figure 4.5, our algorithm shows a lower false alarm rate in *Led7*.

When OutRank-*b* is compared against OutRank-*a*, we found that, on average, it delivers a 20% improvement in precision. Also, a significant reduction in false alarm rate for datasets such as *Austra*, *Zoo*, and *Lymph* can be seen. In *Diabetic* dataset OutRank-*b* shows somewhat low precision compared to OutRank-*a*, and it is because of the choice of threshold.

#### 4.4.2 Effect of the Percentage of Outliers

In this section, we compare the performance of various algorithms when the percentage of outliers is varied. We have compared OutRank-*b* against both LOF and K-dist with Euclidean distance on three of the larger data sets as shown in Figure 4.4 to Figure 4.9. The performance of K-dist algorithm on *kdd-99* dataset was not graphed because its precision and false alarm rate are considerably worse than that for LOF and OutRank-*b*. In general, the precision values for OutRank-*b* did not change significantly compared to LOF and K-dist when the percentage of outliers was varied. OutRank-*b* also shows a comparably lower false alarm rate, whereas other approaches deliver typically unacceptable rate for datasets such as *Led7* and *kdd-99*.

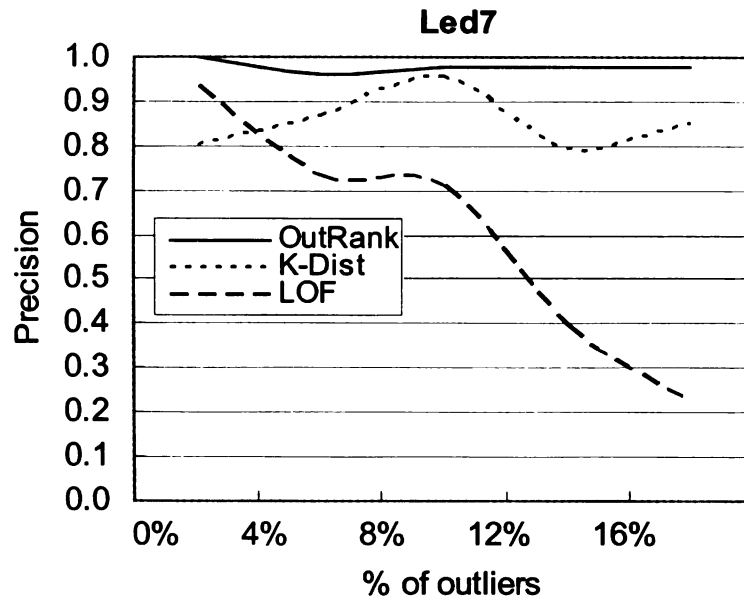


Figure 4.4. Precision while varying the % of outliers in Led7

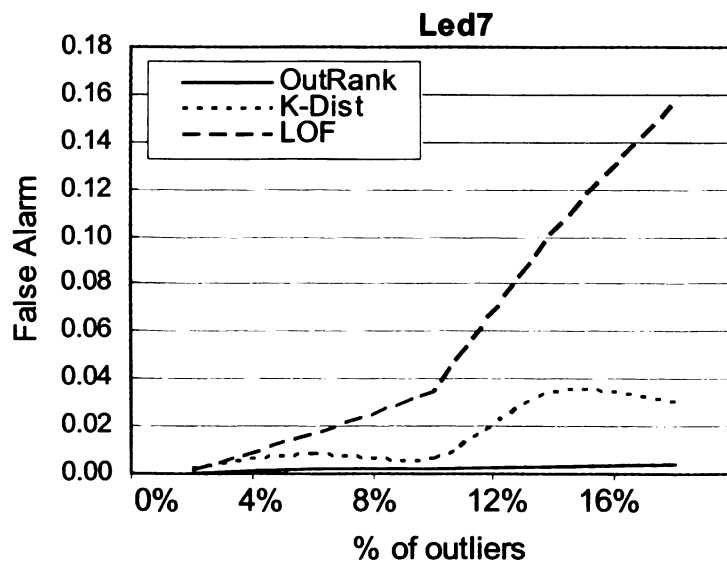


Figure 4.5. False alarm rate while varying the % of outliers in Led7

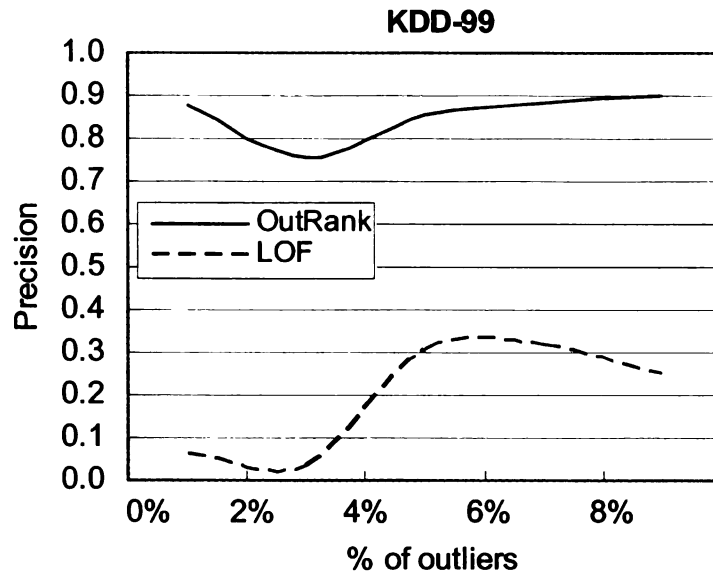


Figure 4.6. Precision while varying the % of outliers in KDD

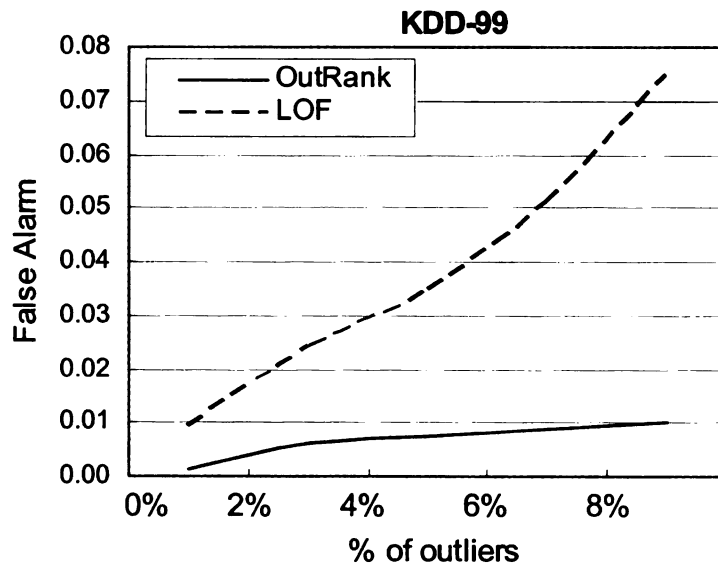


Figure 4.7. False alarm rate while varying the % of outliers in KDD

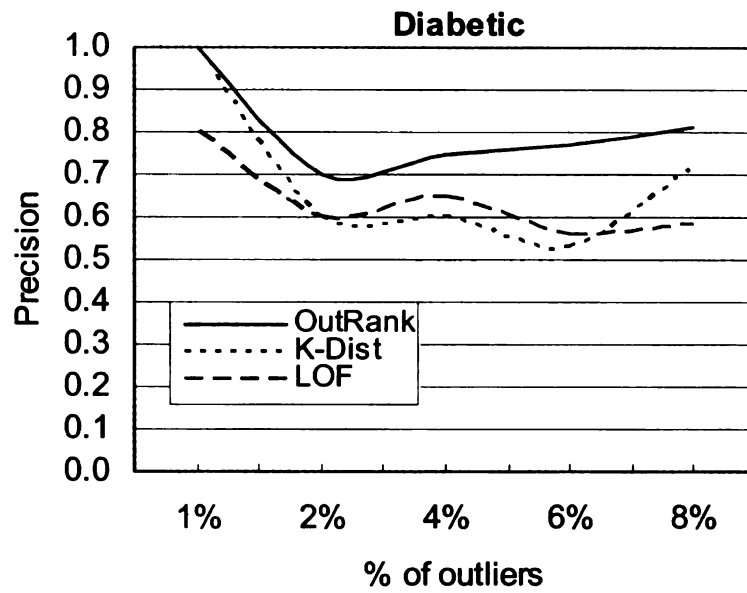


Figure 4.8. Precision while varying the % of outliers in Diabetic

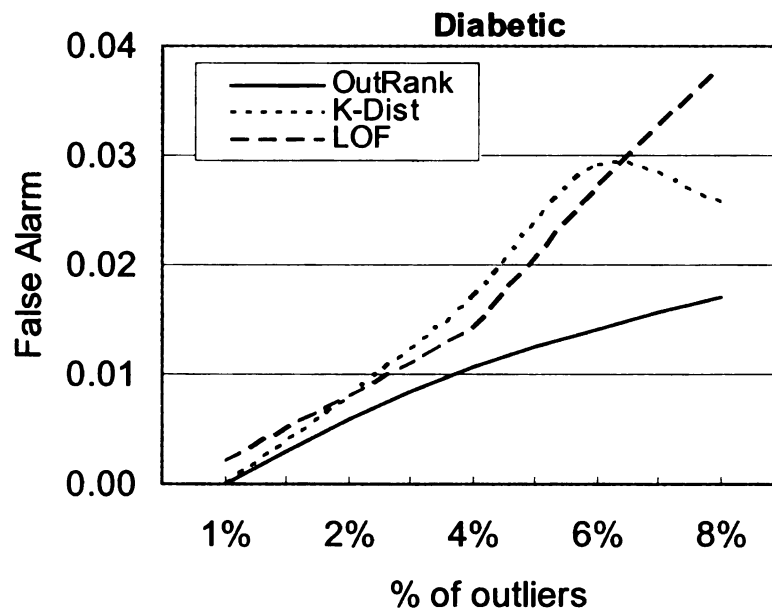


Figure 4.9. False alarm rate while varying the % of outliers in Diabetic



### 4.4.3 Effect of the Shared Neighbor Approach

Here we analyze the effect of the shared neighbor approach used by OutRank-*b* on outlier detection. First, we analyze the precision achieved by the shared neighbor approach. For this analysis, we use 3 versions of OutRank algorithm. Apart from OutRank *a* and *b*, we design a new algorithm called OutRank-*c*, which is similar to OutRank-*a*, but we apply a threshold  $T$  on its similarity matrix to cutoff the low similarity nodes. Note that OutRank-*b* computes shared neighbors using this matrix. So, the only difference between OutRank-*b* and OutRank-*c* is that the former uses shared neighbors for outlier detection. This way we can see whether the performance achieved by OutRank-*b* results from the threshold effect or the shared neighbors. Figure 4.10 shows the precision for the three OutRank algorithms on the *Optical* dataset, while varying the percentage of outliers.

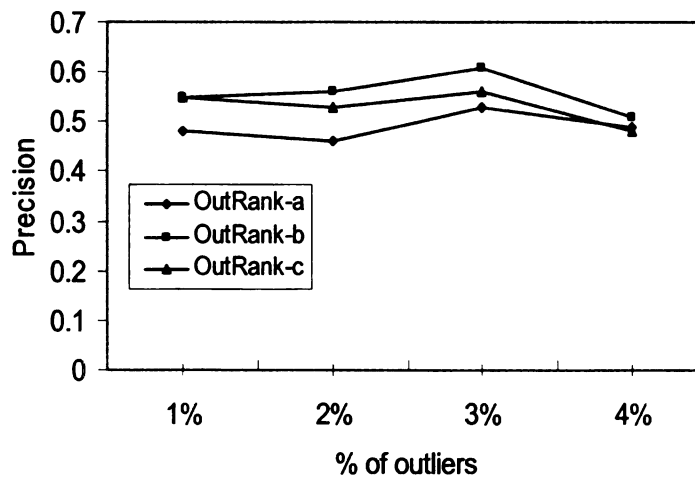


Figure 4.10. Precision of algorithms on optical dataset

In most cases, we can see significant improvement of precision with OutRank-*b* over other algorithms. Furthermore, in situations where the percentage of outliers is sufficiently large, applying a threshold on the similarity matrix can have an adverse effect as shown in Figure 4.10 for the 4% case (here, OutRank-*c* shows a slightly lower precision

than OutRank-*a*), whereas the shared neighbor approach can minimize the thresholding effect.

To further understand the effect of using shared neighbor approach, we have also analyzed the values of the dominant eigenvector (*connectivity*) produced by the random walk model. As shown in Figure 4.11 and Figure 4.12, there is a sharp distinction between the connectivity values of nodes identified as outliers from those identified as normal. The presence of such a rising slope is vital towards detecting outliers using the random walk approach. When comparing the connectivity values of OutRank-*a* to OutRank-*b*, it is clear that the shared neighbor approach tends to push down the connectivity values for outliers and pull them up for normal points. This shows that the shared neighbor approach helps to improve the distinction between outliers and normal nodes, which makes this approach more suitable for the outlier detection task.

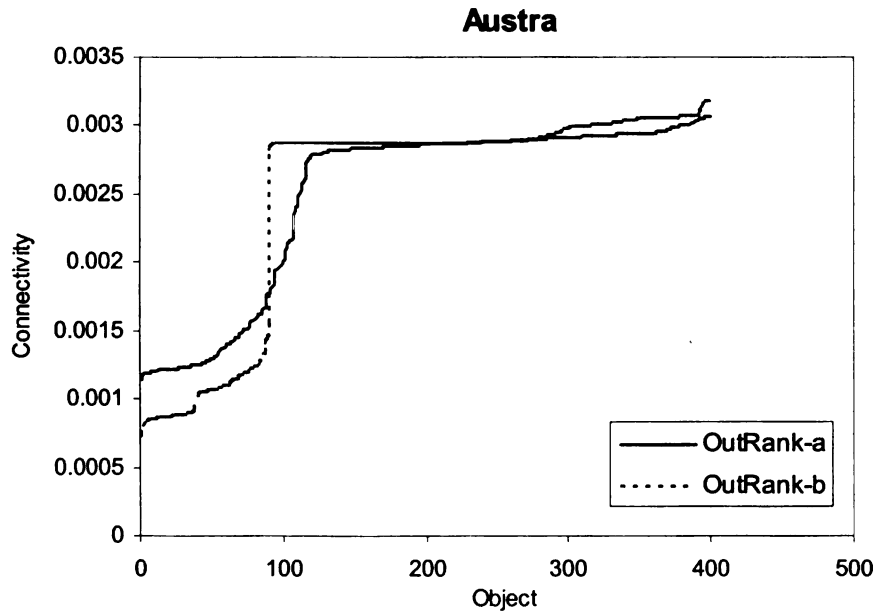


Figure 4.11. Connectivity of objects in Austra dataset

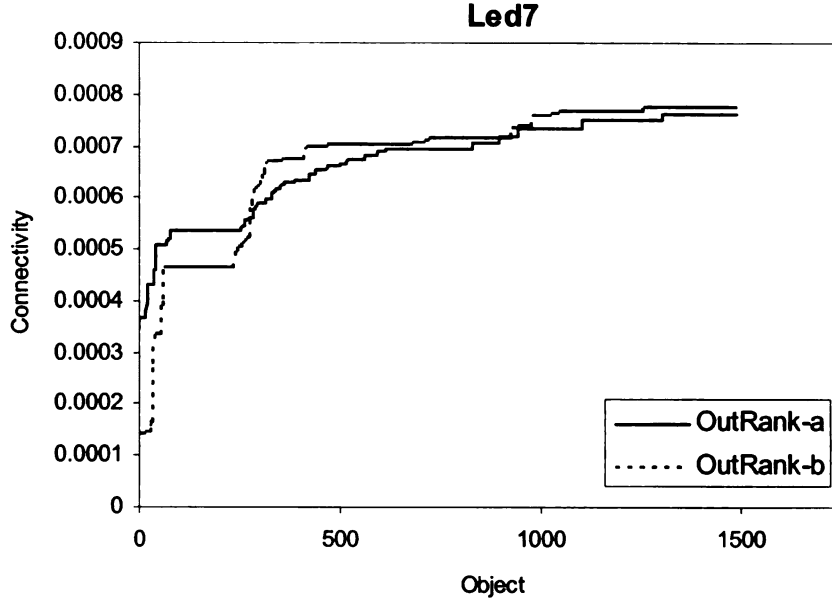


Figure 4.12. Connectivity of objects in Led7 dataset

#### 4.4.4 Choice of Similarity Measures

This section examines the choice of similarity measure for our proposed random walk framework. The cosine similarity measure that we used in most of our experiments cannot effectively handle outliers that are co-aligned with other normal points. As an alternative, we consider using the RBF kernel function given in Equation 4.2 to define the transition probabilities of our random walk model. Table 4.4 compares the precision and false alarm rates of the OutRank-*a* algorithm using RBF kernel and cosine similarity measures.

When comparing the precision achieved by OutRank-*a* using RBF, we can see in six out of nine data sets, its precision is significantly lower than OutRank-*a* using cosine similarity. Nevertheless, when comparing the performance of OutRank-*a* using RBF against LOF and K-dist (using Euclidean distance), the RBF approach still shows better performance in most datasets. K-dist shows better performance in *Diabetic*, *Lymph*, and *Pima* datasets and LOF shows better performance in *Austra*, *Zoo*, and *Lymph* datasets. Even in these datasets, the performance of the RBF approach is not very low.

Table 4.4. Performance comparison with different similarity measures

Data Set		OutRank- <i>a</i> (RBF-Kernel)	OutRank- <i>a</i> (Cosine)
Austra	P	0.1000	0.7727
	FA	0.0529	0.0132
Zoo	P	0.7692	0.9230
	FA	0.0491	0.0163
Diabetic	P	0.6511	0.8837
	FA	0.0321	0.0107
Led7	P	0.9597	0.9516
	FA	0.0080	0.0096
Lymph	P	0.5000	0.5000
	FA	0.0148	0.0148
Pima	P	0.8000	1.0000
	FA	0.0062	0.0000
Vehicle	P	0.4762	0.6190
	FA	0.0520	0.0378
Optical	P	0.6265	0.5300
	FA	0.0115	0.0145
KDD-99	P	0.2710	0.8880
	FA	0.0729	0.0112

#### 4.4.5 Effect of Threshold on the Quality of Solution

Let us analyze the effect of threshold  $T$  on OutRank- $b$ . Note that OutRank- $a$  does not use any threshold. Figure 4.13 and Figure 4.14 show precision for *led7* and *kdd-99* datasets when  $T$  is varied from 0.00 to 0.90. As expected, for higher and lower  $T$  values, precision becomes low. Notice the interval  $[\mu - \sigma, \mu)$ , where our algorithm delivers the highest performance. Also, any  $T \in [\mu - \sigma, \mu)$  shows similar performance in precision, and therefore our algorithm does not exhibit any unexpected sensitivity to the choice of threshold  $T$ .

### 4.5 Discussion

This chapter investigated the effectiveness of a stochastic graph based approach for anomalous pattern mining. Experimental results using both real and synthetic data sets confirmed that this approach is generally more effective at ranking most understandable outliers that previous approaches cannot capture.

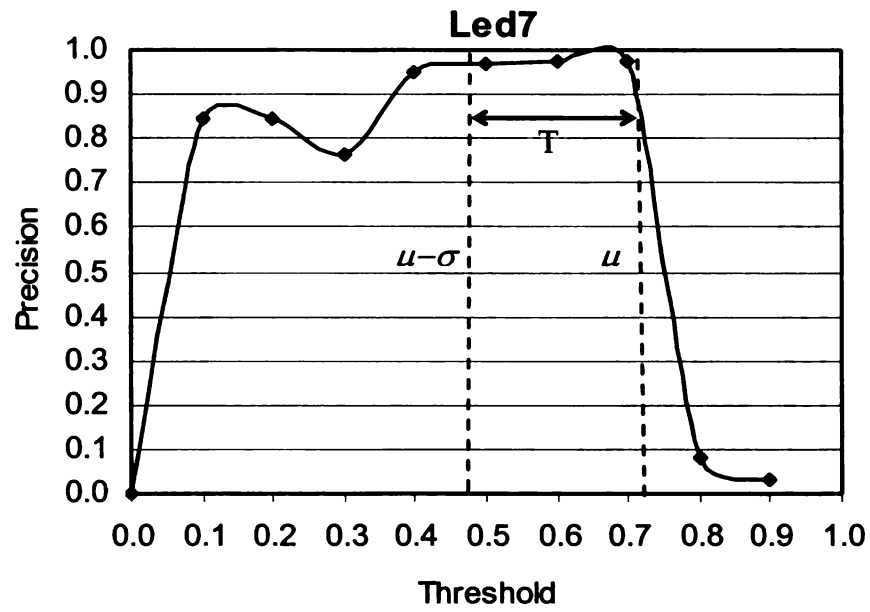


Figure 4.13. Precision for different threshold values in Led7 dataset

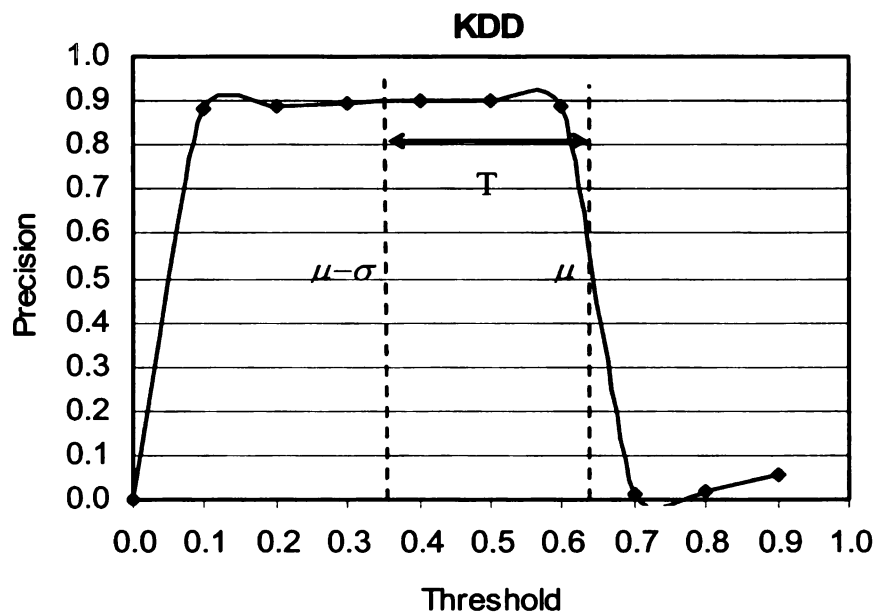


Figure 4.14. Precision for different threshold values in KDD dataset

A key advantage of using our approach is that it can effectively capture not only the outlying objects scattered uniformly, but also small clusters of outliers. Although they show high detection rate over distance-based algorithms for datasets with varying densities, existing density-based algorithms become less effective when identifying small clusters of outliers. This is because these algorithms consider the density of a predefined neighborhood for outlier detection, and in some cases small clusters of outliers with similar density to normal patterns cannot be distinguished. Our approach solves this problem by defining the outlierness of an object with respect to the entire graph of objects; i.e. it views the outlierness from a global perspective.

Also, the results revealed that our outlier detection model tends to do better when the percentage of outliers is gradually increased. In outlier detection algorithms, false alarm rate is considered the limiting factor of its performance and the algorithms proposed here achieved the lowest false alarm rate using an unsupervised learning scheme.

There are several aspects of our framework that can still be improved. First, the cosine similarity measure that we used has both advantages and disadvantages. For example, it cannot effectively handle outliers that are co-aligned with other normal points. Despite this limitation, both *OutRank-a* and *OutRank-b* still outperform standard distance-based and density-based algorithms. We are currently investigating other similarity measures, such as those based on Euclidean-based distances, as the transition probabilities of our random walk model. While these measures work well in low dimensional data, they do not work well in high dimensions.

# 5 Mining Anomalous Graphs

In this chapter, we explore both unsupervised and supervised anomaly detection techniques for graph based data. The main contributions of this work are summarized below:

- We extend *OutRank* to graph based data and propose an effective anomaly detection algorithm, called *gRank*, to determine anomalous graphs.
- We develop a probability-based approach for unsupervised anomaly detection in graphs using the *maximum entropy* principle. The proposed algorithm is called *gEntropy*.
- We extend the *maximum entropy* approach to supervised anomaly detection and propose an effective algorithm, called *gEntropySuper* for classifying anomalous graphs.

The rest of the chapter is organized as follows: In Section 5.1, we discuss some potential applications of mining graph based data. In Section 5.2, we discuss the issues of applying existing anomaly detection methods to graph based data. Section 5.3 presents the *gRank* algorithm, which detects anomalous graphs based on subgraph similarity. In Section 5.4, we present maximum *entropy* based unsupervised anomaly detection scheme called *gEntropy*. Section 5.5 extends the approach to supervised graph anomaly detection. Experimental results on real-world data sets are reported in Section 5.6.

## 5.1 Mining Graph Based Data

The mining of graph-structured objects have received significant interest due to the emergence of applications that generate massive amount of data in the form of graphs. For example, web click-stream data, program execution traces, chemical compounds, and secondary structures in proteins are examples of graph-based data. As such, pattern mining in this type of data becomes important and will be beneficial to many scientific and commercial applications.

The aim of graph mining is to discover interesting patterns within the graph database. For example, patterns derived from the chemical compound database can be used to discover new drugs for treating diseases. Figure 5.1 illustrates an example chemical compound database consisting of compounds used for evidence of anti-HIV activity.

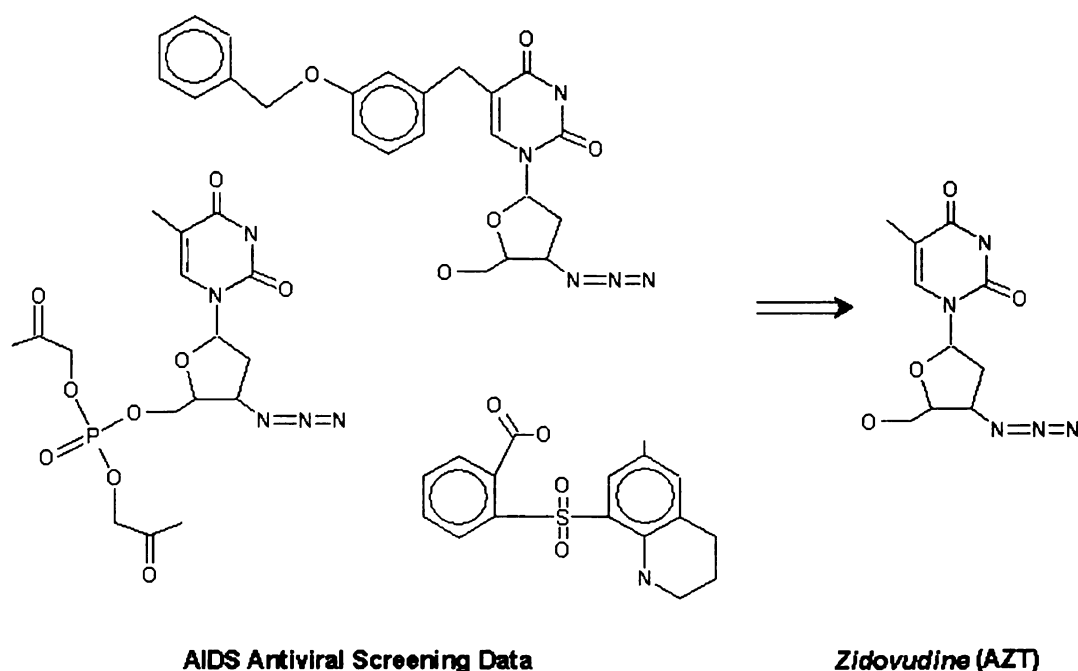


Figure 5.1. AIDS antiviral screening data<sup>4</sup> and its interesting fragments

<sup>4</sup> [http://dtp.nci.nih.gov/docs/aids/aids\\_data.html](http://dtp.nci.nih.gov/docs/aids/aids_data.html)



The chemical compound (*AZT*) shown in the right-hand side of Figure 5.1 is an interesting fragment that is present in some of the compounds in the database. This compound has been shown to provide capability to protect human cells from HIV-1 infections. Finding other graph fragments that might have such an effect is an important step in drug design. Graph mining is one important approach that can be used when mining such datasets.

There are two types of pattern mining problems for graph based data: frequent subgraph mining and anomalous graph mining. The problem of frequent subgraph mining is to discover frequently occurring substructures in a given database. Several efficient subgraph mining algorithms such as *FSG* [KK01], *gSpan* [YH02], and *Gaston* [NK04] have been developed for this task.

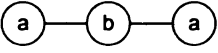
In this thesis, we focus on the anomalous graph mining problem and develop several algorithms for this task.

## 5.2 Anomaly Detection in Graphs and its Issues

Anomaly detection on graph based data has huge potential benefits in a variety of applications. For example, detecting spam web pages has become an important research problem because of the dramatic growth of spam web sites and the adverse effect it causes to degrade the quality of search results produced by the web search engines. Other examples of anomaly detection on graph based data include software bug detection based on program execution traces.

Despite its wide applicability, very few unsupervised anomaly detection algorithms have been developed for graph based data. A state of the art approach that has been proposed recently is the compression based anomaly detection scheme by *Nobel* and *Cook* [NC03]. Their approach uses an algorithm called *Subdue* [CH00] to discover repetitive graph patterns. More specifically, *Subdue* is run in multiple iterations on the graph data-

set and after each run the graph dataset is compressed using the best substructure discovered (using MDL principle [CH00]). Then, every instance of the substructure is replaced by a single new vertex representing it and the new vertices are connected to form the compressed graph. An anomaly score is computed for each graph in the dataset based on the percentage of subgraphs that is compressed away. A major advantage of this method is because of the compression of the graphs, subsequent mining becomes much faster and the compressed graphs are easier to manipulate. Since the vertices need to be connected to form the compressed graph, it can possibly cause two completely different graphs to have the same representation after the connection. Situations such as this are likely to return many false positive answers because of the loss of important structural information during the connection phase. The advantage mentioned above for compression based schemes now becomes a major weak point for detecting anomalous graphs.

The following example illustrates the disadvantage of this approach. Consider a sample dataset of three graphs shown in Figure 5.2 (1). As one can see graphs (i) and (ii) are more similar and hence graph (iii) should be the anomalous graph. Based on the MDL principle [CH00], subgraph  is the best substructure that can be used to compress this dataset. The graph dataset after the compression is shown in Figure 5.2 (2); i.e. each instance of the best substructure is replaced by a new vertex (x). Now graphs (ii) and (iii) have the same structure although originally they were significantly different. Therefore, compression based methods will never identify graph (iii) as anomalous.

The above analysis shows the problems with compression based unsupervised methods. In order to address these issues, we believe a solution should consider the maximal substructure similarity among the graphs. For example graphs (i) and (ii) have a large common substructure which makes them more similar to each other than to graph (iii). In section 5.3, we discuss such an approach to detect anomalous graphs by addressing these existing issues.

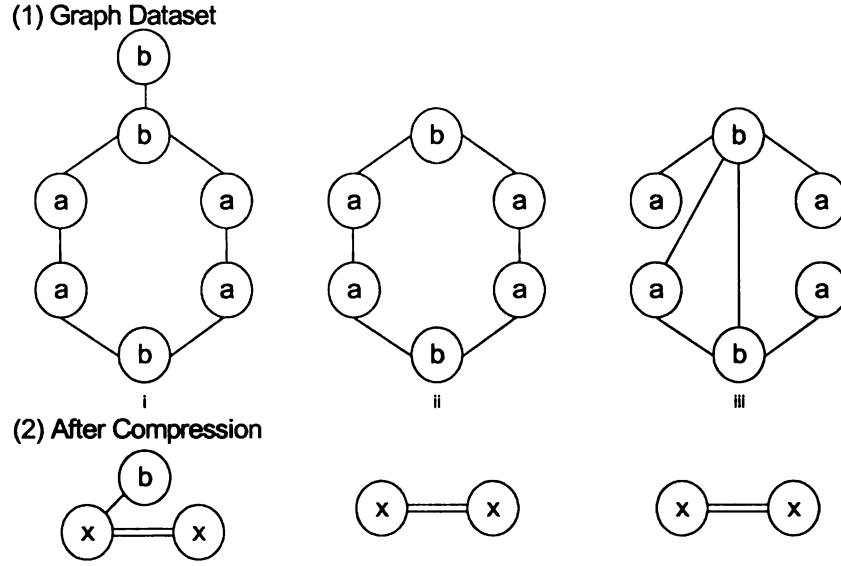


Figure 5.2. Compression based graph similarity

## 5.3 Extending *OutRank* for Graphs

In this section, we present an anomaly detection framework, called *gRank* that extends our previous *OutRank* approach for tabular data.

### 5.3.1 *gRank*: Graph Ranking

A major challenge in extending *OutRank* for graph based data is to determine the *similarity* between the graph objects. Unlike tabular data, where each object is a record, graphs are complex data objects with varying sizes. Therefore, it is difficult to directly compare two graph objects to determine their similarities. In order to address this, we use a feature based scheme to represent each graph object. In feature based schemes, a set of features or invariants is extracted from the graph structures and then used to define the feature vectors. The similarities among graph objects are then computed by applying a similarity measure on their respective feature vectors.

In this method, we apply a frequent subgraph mining algorithm [KK01][YH02][HWP03][NK04] to generate frequent subgraphs for a given minimum support threshold. Once the frequent subgraphs are generated, we build a frequent subgraph based feature vector (FV) for each graph in the dataset. Each component of this feature vector corresponds to the presence or absence (1 or 0) of that feature (frequent subgraph) in the graph. Now the graph database is represented by a set of binary feature vectors.

Once the feature vectors are built, a similarity measure such as *cosine* similarity can be used to obtain the similarities among graph objects. Subsequently, we use the *random walk* approach described in Chapter 4 to determine their anomaly scores. The following algorithm, *gRank*, gives an outline of our anomaly detection framework for graphs:

---

**Algorithm 5 (*gRank*)**

---

**Input:** Graph database  $G$ , error tolerance  $\epsilon$

**Output:** Outlier ranks.

**Method:**

- 1: construct features using frequent subgraph mining on  $G$
  - 2: construct the similarity matrix –  $Sim$  // use graph similarity
  - 3: call *OutRank* ( $Sim, \epsilon$ )
- 

The space complexity of *gRank* depends on the space required by the underlying frequent subgraph mining algorithm, feature vectors, and the *OutRank* algorithm. Let  $S_f$  be the space requirement of the underlying subgraph mining method for storing intermediate subgraphs in order to generate frequent patterns. Let  $|F_I|$  be the frequent subgraphs generated. Then with a graph database of  $N$  objects, it takes  $|F_I|N$  memory space to store feature vectors. Moreover, *OutRank* has  $O(N^2+2N)$  space complexity. Therefore, the space complexity of the algorithm is  $O(\max(S_f, |F_I|N + (N^2+2N)))$ .

The time complexity of this algorithm depends on the time to generate frequent subgraphs, construction of similarity matrix, and the *OutRank* method. Let  $T_f$  be the time complexity of any existing subgraph mining method. In order to construct binary feature

vectors it requires  $T_s = N|F_I|$  isomorphism checks, where subgraph isomorphism has exponential time complexity. Algorithm OutRank has  $O(N^3)$  time complexity. Therefore, the total time complexity can be stated as  $O(T_f + T_s + N^3)$ .

In the next section, we discuss feature vector construction and the different similarity measures used to define the similarities among graphs.

### 5.3.2 Similarity Measures for Graphs

In this section, two variants of the *gRank* algorithm are presented based on the choice of similarity measure for graphs. The first approach, *gRank-cos*, uses *cosine* similarity while the second approach, *gRank-cg*, introduces a new similarity measure based on the concept of maximal common frequent subgraphs.

#### 5.3.2.1 *gRank-cos*

Let  $g_1 = (x_1, x_2, \dots, x_d)$  and  $g_2 = (y_1, y_2, \dots, y_d)$  be any two feature vectors of the graphs drawn from a  $d$ -dimensional feature space  $\mathbb{R}^d$ . The similarity between  $g_1$  and  $g_2$  is defined by the *cosine* between two corresponding vectors:

$$\text{cosine\_similarity}(g_1, g_2) = \begin{cases} 0 & \text{if } g_1 = g_2 \\ \frac{\sum_{k=1}^d x_k y_k}{\sqrt{\sum_{k=1}^d x_k^2} \cdot \sqrt{\sum_{k=1}^d y_k^2}} & \text{otherwise} \end{cases} \quad (5.1)$$

Note that the similarity between a graph object to itself is set to zero to avoid self loops in the underlying representation. Such loops are ignored since they are common to every node, and therefore it is not very useful to distinguish normal objects from outliers.

### 5.3.2.2 *gRank-cg*: using maximal common frequent subgraphs

In this section, we develop a new graph similarity measure that uses *maximal common frequent subgraphs* as an alternative to the previously defined similarity measure. A maximal common frequent subgraph can be defined as follows:

**DEFINITION 5.1 (Maximal Common Frequent Subgraph)** Let  $g_1, g_2 \in G$  be any two graphs and  $F$  be the set of frequent subgraphs discovered in  $G$ . Let  $F_{g_1 \cap g_2}$  be the set of frequent subgraphs contained in both  $g_1, g_2$ . The *maximal common frequent subgraph*  $S_{g_1 \cap g_2} \subseteq F_{g_1 \cap g_2}$  such that if  $f_1 \in S_{g_1 \cap g_2}$  then there is no  $f_2 \in S_{g_1 \cap g_2}$ , where  $f_1 \subseteq f_2$ .

Note that each maximal common frequent subgraph of the two graphs under consideration contains frequent subgraph components that are edge disjoint and can be disconnected. For example, Figure 5.3 shows two graphs and maximal common frequent subgraph (highlighted edges) for each of them.

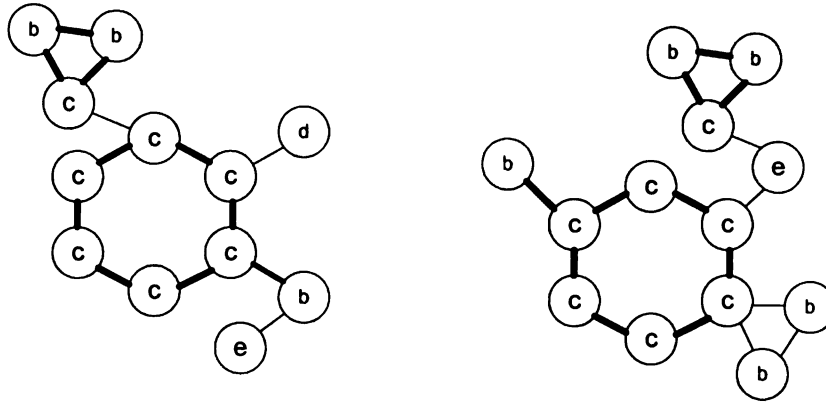


Figure 5.3. Maximal common frequent subgraph of the two graphs

In this method, the key idea is to capture approximately a *maximal common frequent subgraph* between the graphs being compared. Use of frequent subgraphs allows us to discriminate the anomalous graphs as they cannot have very many frequent components.

On the other hand, discovering maximal common frequent subgraphs allows us to determine the structural similarity between the graphs more accurately than the previous method based on cosine similarity. In that method every feature has the same weight on similarity computation whether it is a larger substructure or a one-edge subgraph. But, as we know, if graphs have a large common substructure then they should have high similarity when compared to graphs having lots of common one-edge substructures.

In order to determine maximal common frequent subgraph, we develop an efficient methodology based on the frequent subgraph lattice that can be generated by existing graph mining algorithms, such as FSG [KK01]. In the following sections, we will discuss this in detail for the two types of graph datasets: graphs containing unique vertex labels, and graphs with repeating vertex labels.

**Graphs with Unique Vertex Labels:** Here we consider graphs with unique vertex labels. Some of the real world examples containing such a graph data includes web click stream data and program execution traces. In web click stream data, vertices correspond to unique web pages and edges correspond to the navigation pattern of the user, whereas in program execution data, vertices corresponds to functions and procedures in the program and the control transfer between them represents edges.

In our method, we use a frequent subgraph mining algorithm to generate frequently occurring subgraphs. These subgraphs can be arranged in a lattice structure, where the nodes represent the subgraphs and edges represent the subgraph and super graph relationship of the nodes. For example, consider the lattice of frequent subgraphs shown in Figure 5.4. Here subgraph 3-1 (here  $l-i$  denotes subgraph  $i$  at level  $l$ ) is a superset of subgraphs 2-1 and 2-2.

A lattice  $L$ , generated using frequent subgraphs, has several properties that can be illustrated as follows:

- i.  $\forall g \in L \quad \sigma(g) \geq \xi$

ii. If  $\exists e=(g_i, g_j) \in L$  such that  $|g_i| \leq |g_j|$  then  $g_i \subset g_j$   $0 < i, j \leq N$

where  $N$  is the number of subgraphs in  $L$  and  $g, g_i$ , and  $g_j$  are frequent subgraphs.

We define subgraph  $g$  as a *common ancestor* of two graphs  $g_1$  and  $g_2$  if  $g$  is a subgraph of both  $g_1$  and  $g_2$ . For example, subgraph 1-2 is a common ancestor of both subgraphs 3-1 and 2-3.

**Theorem 5.1.** For any given two subgraphs  $g_1, g_2 \in L$  if there is no common ancestor between them, then  $g_1$  and  $g_2$  are edge disjoint subgraphs.

**PROOF.** Let  $g_1, g_2 \in L$  be any given two subgraphs with a common ancestor  $g$ ; i.e. there is at least one edge  $e$  that is common to both of these graphs. If such an edge exists, by the definition of edge disjoint of two graphs,  $g_1$  and  $g_2$  become non edge-disjoint, which is a contradiction. ■

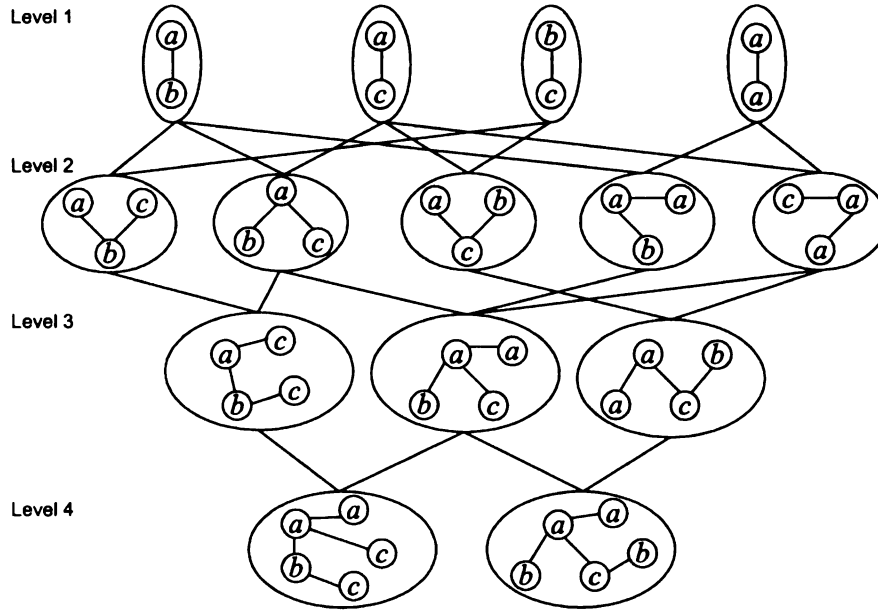


Figure 5.4. Frequent subgraph lattice

This theorem is used to build the *maximal common frequent subgraphs* in our anomaly detection framework. In this approach, for a given graph  $g$  and a set  $F$  of frequent subgraphs present in  $g$ , we can easily identify the edge disjoint frequent subgraph in  $g$  by



simply checking the common ancestor between each of them in lattice  $L$ . Algorithm 6 summarizes the process for finding maximal common frequent subgraphs between two given graphs.

---

**Algorithm 6 (*MaxCommonFrequentSubgraph\_unique*)**

---

**Input:** Feature vectors  $FVg_1$ , and  $FVg_2$ , Lattice  $L$

**Output:** Max common frequent subgraph  $S$

---

**Method:**

- 1: let  $C$  = set of all common frequent subgraphs  $FVg_1 \cap FVg_2$
  - 2: let  $k_{max} = \max_{|g|} \{g \in C\}$
  - 3: for  $k = k_{max}$  down to 1 do
  - 4:     if  $f$  is edge disjoint with all of the graphs in  $S$    // using Theorem 5.1
  - 5:          $S = S \cup \{f\}$
  - 6: return  $S$
- 

In this algorithm, we first determine the set of all common frequent subgraphs ( $C$ ) between  $g_1$ , and  $g_2$  by intersecting their corresponding feature vectors. Then the graphs are selected from  $C$  in the decreasing order of their size and inserted into the maximal common frequent subgraph  $S$ , if they are edge disjoint to each other. The set  $S$  represents the maximal common frequent subgraph between  $g_1$  and  $g_2$ .

**Graphs with Repeating Vertex Labels:** Now let us consider how to determine the maximal common frequent subgraphs in graphs with repeated vertex labels. Since the vertex labels for this type of graphs can be repeated, it is possible that a frequent subgraph occurs more than once in a given graph. Therefore, our previous method is not directly applicable here as the lattice does not provide any frequency information of the occurrence of frequent subgraphs. On the other hand, searching the graph for determining the existence of frequent subgraphs is costly since it involves a subgraph isomorphism test, which is NP complete. Therefore, in this work we propose an approximate method that can be used to find a maximal common frequent subgraph between any given two graphs.

In this method, we determine the maximal common frequent subgraph  $S$  between any given two graphs in such a way so that  $S$  covers maximum number of frequent subgraphs of size 1 (in the graph database, denoted as set  $L1$ ). In order to do this, we first find the set of all frequent subgraphs ( $C$ ) common to both  $g_1$  and  $g_2$  by intersecting their corresponding feature vectors ( $FVs$ ). Then, the subgraph  $f (\in C)$  with the maximum size is selected and  $S$  is initialized to contain  $f$ . Then,  $f$  becomes the first component of the maximal common frequent subgraph. Also, an edge cover ( $COVER$ ) is initialized to contain all the frequent subgraphs of size 1 that belong to  $f$ . When selecting the next frequent subgraph that needs to be added into  $S$ , we pick the best graph that covers the maximal number of frequent subgraphs (edges) in  $L1 \setminus COVER$ ; i.e. we always select the subgraph with highest edge coverage to be added into  $S$ , thus making this method an approximate method to discover maximal common frequent subgraphs.

The complete method for generating a maximal common frequent subgraph for any given two graphs is presented in Algorithm 7.

---

**Algorithm 7 (*MaxCommonFrequentSubgraph\_repeating*)**

---

**Input:** feature vectors  $FVg_1$ , and  $FVg_2$ , Set of frequent subgraphs  $F$ , lattice  $L$ , set of frequent subgraphs of size =1  $L1$

**Output:** Maximal common frequent subgraph  $S$

**Method:**

- 1: let  $C$  = set of all frequent subgraphs of  $F$  belongs to  $FVg_1 \cap FVg_2$
  - 2: let  $f$  = maximum size subgraph in  $C$  and let  $S = \{f\}$
  - 3: let  $COVER = \{\text{frequent subgraphs of size =1 present in } f\}$
  - 4: repeat
  - 5:   remove subgraphs of  $f$  from  $C$    // use lattice  $L$  to determine subgraphs
  - 6:   select frequent subgraph  $f$  from  $C$  s.t. it covers maximum number of edges in  $L1 \setminus COVER$
  - 7:    $COVER = COVER \cup \{\text{edges of } f\}$
  - 8:    $S = S \cup \{f\}$
  - 9: until  $COVER = L1$  or  $|C| = 0$
  - 10: return  $S$
- 

Next, we will describe how the similarity between two graphs can be computed using the maximal common frequent subgraph.

**Common Subgraph Similarity:** Let  $F = \{f_1, f_2, \dots, f_k\}$  be the  $k$  frequent subgraph components of the maximal common frequent subgraph between any given two graphs  $g_1$  and  $g_2$ . Then, the similarity between  $g_1$  and  $g_2$  can be defined as:

$$CG\_Similarity(g_1, g_2) = \frac{\left( \sum_{i=1}^k N(f_i) - C \right)^2}{N(g_1) \cdot N(g_2)} \quad (5.2)$$

where  $N(g)$  denotes the sum of the number of vertices and edges of a graph  $g$ , and  $C$  is the number of vertices that may possibly be in common among the subgraphs in  $F$ .

## 5.4 *gEntropy*: Maximum Entropy Based Unsupervised Anomaly Detection

In the preceding section, we discussed an anomaly detection framework using substructures to define the similarity measure for graphs. In this section, we use substructures to build a probability model and use this model to determine the anomaly score of graphs. Our probability model is developed based on the *maximum entropy* principle.

### 5.4.1 Maximum Entropy Model

Maximum entropy modeling synthesizes a set of local patterns into a global model by choosing a probability model that is consistent with the constraints imposed by the local patterns, but otherwise, it is as uniform as possible. Mathematically, the approach corresponds to finding a probability model  $P$  that minimizes the *Kullback-Leibler* divergence  $D(P||Q)$ , subject to a set of constraints  $\wp$ , with  $Q$  chosen to be a uniform probability distribution. The resulting model can be shown to be equivalent to a model  $P^*$  that maximizes the entropy:

$$P^* = \arg \max_P H(P) \quad (5.3)$$

where  $H(P)$  is the entropy of  $P$ .

*Maximum entropy* modeling has been used in a variety of areas such as *machine learning* and *Natural Language Processing* (NLP). For example, it is used to build language translation models [BPP96, ZR01]. Khudanpur and Wu [KW99] have used the *maximum entropy* principle to conversational speech recognition. On the other hand, Mannila et al. [MS99] have applied the principle to query selectivity and protein sequence modeling applications. In [NLM99] Nigam et al. illustrated how the *maximum entropy* principle could be used for text classification.

Unlike the previous works, this thesis employs the *maximum entropy* principle to build a probability model for anomalous graph detection. Once the model has been derived, it can be used to detect anomalies by examining the probability assigned by the model to each graph in the database. This probability represents how likely the graph is generated from the same random process that generates the majority of the graphs in the database; the lower the probability, the more anomalous the graph is.

We first introduce the notations used throughout this section. Let  $G = \{g_1, g_2, \dots, g_N\}$  be the set of  $N$  graphs in the graph database and  $X = \{x_1, x_2, \dots, x_m\}$  be the set of  $m$  frequent subgraphs discovered in  $G$  for a given support threshold  $\xi$ . Furthermore, let  $S = \{s_1, s_2, \dots, s_m\}$  be the support count of each frequent subgraph. To apply the *maximum entropy* principle we define a set of features  $F = \{f_1, f_2, \dots, f_m\}$  from the frequent subgraphs, where  $f_i(g_j) = 1$  if the graph  $g_j$  contains the frequent subgraph  $x_i$ . Now, our objective is to derive a probability model  $p(g)$  that satisfies the following two constraints:

- i.  $p(g)$  should maximize entropy  $\left[ - \sum_{g' \in G} p(g') \log p(g') \right]$
- ii.  $\sum_{g'} p(g') f_i(g') = s_i$

where  $s_i$  is the support count of frequent subgraph  $x_i$  observed in the graph database  $G$ . Note that constraint (ii) ensures that the expected value for every feature to be identical to the support of its corresponding subgraph. It can be shown that the model that solves the constrained optimization problem has the following exponential distribution:

$$P(g) = \frac{1}{Z} \exp \left[ \sum_{i=1}^m \lambda_i f_i(g) \right] \quad (5.4)$$

where  $Z = \left( \sum_g \exp \left[ \sum_{i=1}^m \lambda_i f_i(g) \right] \right)$  is the normalization factor and  $\lambda$ 's are the parameters to be estimated from the graph database.

Once the parameters ( $\lambda$ ) have been estimated the model can be used to compute the anomaly score of a graph,  $A_g$ , in the following way:

$$A_g = -\log P(g) \quad (5.5)$$

$$= \sum_{i=1}^m \lambda_i f_i(g) \quad (5.6)$$

where the lower the probability, the higher the anomaly score. The graphs are then ranked in increasing order of their anomaly score.

The first step of the anomaly detection algorithm is to generate frequent subgraphs using any available frequent subgraph mining algorithm. Once the frequent subgraphs have been generated, binary feature vectors are constructed for each graph in the graph database. The graph anomaly detection process has two major steps: (1) learn the model by estimating the optimal weight-set ( $\lambda$ ); (2) rank the graphs according to their anomaly score. The following algorithm (*gEntropy*) describes the procedure we used to detect anomalies.

---

**Algorithm 8 (*gEntropy*)**

---

**Input:** Graph database  $G$ , min support threshold  $\xi$

**Output:** Outlier ranks

**Method:**

- 1: generate features  $f_1, f_2, \dots, f_n$  using a frequent subgraph mining algorithm
  - 2: construct set of feature vectors  $FV$  for each graph  $g \in G$
  - 3: initialize the parameter set  $\{\lambda_i \mid \forall i \in \{1, 2, \dots, n\}\}$
  - 4: repeat                    *// learn  $\lambda$ 's*
  - 5:     generate a graph sample of size  $k$  with  $Metropolis(G, FV, \lambda, k)$
  - 6:     for  $i = 1$  to  $n$  do
  - 7:         update  $\lambda_i$  by solving Equation (5.7)
  - 8:     until convergence of  $\lambda$ 's
  - 9:     for each graph  $g$  in  $G$  do
  - 10:         compute  $A_g$  by solving Equation (5.6)
  - 11:         rank graphs from  $\max(A_g)$  to  $\min(A_g)$
  - 12: return the rank of the graphs
- 

The time complexity of this algorithm has 4 components: generation of frequent subgraphs, construction of feature vectors, calculation of optimal model, and the calculation of anomaly score. Let  $T_f$  be the time complexity of frequent subgraph generation, which depends on a number of factors such as support threshold, database size, the number of database scans, the data structures used by the mining algorithm as well as the pruning strategies used. Constructing  $m$  binary feature vectors for  $N$  graph objects requires  $T_s = mN$  subgraph isomorphism checks, where subgraph isomorphism has exponential time complexity. Optimal model computation takes  $(T_M + mN)T_{steps}$ , where  $T_{steps}$  is the maximum number of iterations required for model convergence and  $T_M$  is the time requirement for *Metropolis* sampling approach. Once the parameters have been determined, computing the anomaly scores for the graphs takes  $O(N)$  and sorting their scores takes  $O(N \log N)$ . Therefore, the overall time complexity is  $O(T_f + mN + (T_M + mN)T_{steps} + N + N \log N)$ . According to this *Metropolis* sampling will be the dominant term.

Similarly, space complexity of these four steps can be analyzed. Let  $S_f$  be the space requirement of the underlying subgraph mining method. The storage requirement for the binary feature vectors is  $mN$ . Also, the optimal parameter set require  $m$  memory space

and let  $S_M$  be the space requirement of *Metropolis* sampling approach. Finally,  $N$  spaces are needed to store the anomaly score of each graph. Therefore, the overall space complexity of the algorithm is  $O(\max(S_f, (nN + n + S_M + N)))$ .

### 5.4.2 Parameter Estimation

We will estimate parameter  $\lambda$ 's using the the *Generalized Iterative-Scaling (GIS)* algorithm [DR72], which is a well known iterative technique that converges to the solution  $p(g)$  for problems of the form (5.3). To find the parameter set, the *GIS* algorithm will iteratively adjust each parameter  $\lambda_i$  by an amount as described below:

$$\lambda_{i+1} \leftarrow \lambda_i + \frac{1}{C} \log \left[ s_i / \sum_g p(g) f_i(g) \right] \quad (5.7)$$

where  $s_i$  is the support of subgraph  $i$  in  $G$ ,  $C$  is a constant (maximum count of frequent subgraphs present in  $g$ ), and  $\sum_g p(g) f_i(g)$  is the expected support. When the expected

support is equal to  $s_i$ ,  $\lambda_i$  will not change and the solution is said to converge.

However, computation of  $\sum_g p(g) f_i(g)$  over all  $g$  is a challenge since it requires us

to consider all the possible graphs. In this approach, we compute the expected value using the *Metropolis* sampling approach. The approach generates a sample of graphs that satisfy the probability model  $p(g)$  and will be discussed in the next section.

### 5.4.3 Metropolis Sampling Approach

The *Metropolis* sampling algorithm is used to generate a sample of graphs according to some probability model  $p(g)$ . In this algorithm, we first select a graph  $g \in G$  randomly and modify it in such a way that each modification generates a new graph  $g'$ . A modification of  $g$  may involve any of the following operators:

- i. **ADD\_EDGE**: add an edge between any two vertices of  $g$ .

- ii. ADD\_VERTEX: add a vertex and connect it to an existing vertex of  $g$  by adding an edge between them.
- iii. DEL\_EDGE: remove an existing edge  $e$  attached to any two vertices of  $g$  in such a way so that the resultant graph  $g'$  does not become *disconnected*. Furthermore, if any of the vertices attached to  $e$  has *degree* equals to 1, then the vertex and its edge will be removed.

The selection of the operator is carried out according to the distribution of vertices and edges in each graph of  $G$ . We assume that the distribution of vertices and edges in  $G$  have a *Poisson* distribution. We calculate the average number of vertices ( $\lambda_v$ ) and average number of edges ( $\lambda_e$ ) of the graphs in  $G$ . Then, the probability that there is exactly  $k$  vertices corresponds to  $p(V=k) = \frac{(e^{-\lambda_v} \lambda_v^k)}{k!}$ . Similarly,  $p(E=k) = \frac{(e^{-\lambda_e} \lambda_e^k)}{k!}$  corresponds to the probability of exactly  $k$  edges. For a graph with  $n$  vertices and  $m$  edges we define the probabilities of the three operators as follows:

$$\text{i. } p(\text{ADD\_VERTEX}) = p(V=n+1) \times p(E=m+1) \quad (5.8)$$

$$\text{ii. } p(\text{ADD\_EDGE}) = p(V=n) \times p(E=m+1) \quad (5.9)$$

$$\text{iii. } p(\text{DEL\_EDGE}) = p(V=n) \times p(E=m-1) \quad (5.10)$$

Once the individual probabilities are computed, they are normalized to 1 and an operator is selected with its corresponding probability. Once an operator is selected, graph  $g$  can be modified by applying the underlying operation to form a candidate graph  $g'$ . However, the acceptance of  $g'$  as the next graph is decided in a probabilistic manner (using *Metropolis* algorithm). Given  $g'$ , we calculate the following ratio  $\alpha$  between  $g'$  and current graph  $g$ .

$$\alpha = \frac{P(g')}{P(g)} = \frac{\exp \left[ \sum_i \lambda_i f_i(g') \right]}{\exp \left[ \sum_i \lambda_i f_i(g) \right]} \quad (5.11)$$



Note that the constant  $Z$  cancels out. Now if  $\alpha \geq 1$  we accept the candidate  $g'$ . If  $\alpha < 1$  we accept  $g'$  with probability  $\alpha$ , else we reject the candidate and generate a new candidate graph starting from  $g$  by applying the operators. Once  $g'$  is accepted it becomes the current graph  $g$  and this process is repeated to generate more graphs.

This process generates a *Markov* chain  $(g'_0, g'_1, \dots, g'_k, \dots)$  as the transition probabilities from  $g'_i$  to  $g'_{i+1}$  depend only on  $g'_i$  and not  $(g'_0, g'_1, \dots, g'_{i-1})$ . In order to generate a sample of graphs using this method, we initially allow a sufficient number of graphs to be generated (say  $k$ ) during the *burn-in* period. Following this burn-in period the chain approaches its stationary distribution and samples from the vector  $(g'_{k+1}, \dots, g'_{k+n})$  are samples from our probability distribution  $p(g)$ . To further distinguish the graphs generated, once  $g'_{k+1}$  is generated we allow a smaller *burn-in* interval ( $k' < k$ ) before selecting the next graph  $g'_{k+2}$ . We can carryout this process a given number ( $n \times k'$ ) of times to generate a sample of  $n$  graphs. Based on the above discussion, we present our sample generation algorithm, called *Metropolis*.

---

**Algorithm 9 (*Metropolis*)**

---

**Input:** Graph database  $G$ , feature vectors  $FV$ , parameter set  $\lambda$ 's, size of the sample  $k$

**Output:** set of  $k$  graphs and corresponding  $FV$ s

---

**Method:**

- 1: select a graph  $g$  randomly from database  $G$  and its  $FV_g$
  - 2: let initial burn-in period = const-1 and subsequent burn-in period = const-2
  - 3: for  $i = 1$  to  $k$  do
  - 4:     repeat
  - 5:         compute probability of each operator using Equations (5.8)-(5.10)
  - 6:         select the operator  $op$  (ADD\_EDGE/ADD\_VERTEX/DEL\_EDGE)
  - 7:         call *GenNextGraph* ( $g, FV_g, op$ ) to get a new graph  $g'$  and its feature vector  $FV_{g'}$
  - 8:         compute  $\alpha$  by solving Equation (5.11)
  - 9:         if  $\alpha \geq 1$  let  $g = g'$      // accept or reject  $g'$
  - 10:        else if  $\alpha < 1$  let  $g = g'$  with probability  $\alpha$
  - 11:     until end of burn-in period
  - 12:     add the graph  $g$  and its  $FV$
  - 13: return the set of  $k$  graphs and corresponding  $FV$ s
-

This algorithm has two *burn-in* periods as described earlier. In general, the initial *burn-in* period (const-1) is set to high value (e.g. 100 iterations) and the subsequent *burn-in* periods (const-2) can be set to lower value (e.g. 20 iterations). At the end of each *burn-in* period the graph is picked to be inserted into the sample. In the next section, we discuss how the feature vector is generated for newly generated graph (Algorithm *GenNextGraph*).

#### 5.4.4 Feature Vector Generation

A key challenge in the sampling process is compute the feature vector,  $f_i(g)$  associated with each frequent graph  $i$ , of the newly generated graph  $g'$ . Since  $g$  has been modified its feature vector may no longer be valid for  $g'$ .

A trivial solution to find the affected frequent graphs once an operator is applied to  $g$  can be easily determined. In the case of ADD\_EDGE (or ADD\_VERTEX), we can consider all the non-present frequent graphs of  $g$  (i.e.  $F_{np} = \{\text{graph } i \mid f_i(g)=0\}$ ), and search whether they are in  $g'$ , after applying the operator to  $g$ . Similarly, we can consider all the frequent graphs of  $g$  (i.e.  $F_p = \{\text{graph } i \mid f_i(g)=1\}$ ), and search whether they are in  $g'$  after applying the operator DEL\_EDGE. However, when we analyze the relationship between a graph and the number of frequent subgraphs not present in it (i.e.  $|F_{np}|$ ), obviously that number is quite large when compared to the number of frequent subgraphs present (i.e.  $|F_p|$ ). Since searching  $g'$  involves sub-graph *isomorphism* testing and because  $|F_{np}|$  is quite large, this method can possibly be computationally expensive for operators like ADD\_EDGE, and ADD\_VERTEX. We therefore propose a more efficient technique to compute the feature vector  $f_i(g')$  for these operators without incurring a significant cost.

In this approach, we use the subgraph *lattice* ( $L$ ) that is generated during subgraph mining. This *lattice* contains information such as subgraph-supergraph relationship for

each frequent subgraph discovered. Since  $|F_p|$  is smaller, we start with the graphs in  $F_p$ , instead of  $F_{np}$  as in the trivial case, and construct the new feature vector gradually.

Here, for each frequent graph in  $F_p$  we obtain its super graph  $f'$  from  $L$  s.t.  $f_{sg} \notin F_p$ , and checks whether  $f'$  is contained in  $g'$ . If  $f'$  is present, we set the corresponding bit of the feature vector to 1, and it will be added to  $F_p$  so that subsequently its super graphs will also be checked. Using the *lattice* information, we give the following set of propositions that will speedup the existence check of  $f'$  in  $g'$  by pruning many unnecessary checks.

**PROPOSITION 5.1:** *Given the operator DEL\_EDGE, a graph  $g$  and its modified graph  $g'$ , along with a frequent subgraph  $f$ , if  $f$  is not present in  $g$ , then all of its super graphs in  $L$  will not be present in  $g'$  and therefore can be pruned.*

**PROOF.** Since  $f$  is not present in  $g$  all of its super graphs in  $L$  cannot be present in  $g$ . Since the subgraph  $g'$  is generated from  $g$  by the operator DEL\_EDGE (i.e.  $g' \subset g$ ) and therefore none of the super graphs should not be present in  $g'$  and can be pruned. ■

**PROPOSITION 5.2:** *Given the operator ADD\_EDGE, a graph  $g$  and its modified graph  $g'$ , a frequent subgraph  $f \in g$  along with its super graph  $f' (\notin g)$  in  $L$ , if the number of vertices of  $f'$  is greater than that of  $f$ , then subgraph  $f'$  can be pruned.*

**PROOF.** Since the operator ADD\_EDGE only adds an edge between any given two vertices of  $g$  to generate  $g'$ , the number of vertices of  $g'$  is equal to  $g$ . Since  $f'$  does not present in  $g$  and has extra vertices than  $f$ ,  $f'$  cannot be present in  $g'$  and therefore can be pruned. ■

**PROPOSITION 5.3:** *Given the operator ADD\_VERTEX, a graph  $g$  and its generated graph  $g'$ , a frequent subgraph  $f \in g$  along with its super graph  $f' (\notin g)$  in  $L$ , if the number of vertices of  $f'$  is equal to that of  $f$ , then subgraph  $f'$  can be pruned.*

**PROOF.** The operator ADD\_VERTEX only adds a vertex to  $g$  in order to generate  $g'$ ; i.e.  $V(g') > V(g)$ . Since  $f'$  does not present in  $g$  and has same number of vertices as  $f$ ,  $f'$  cannot be present in  $g'$  and therefore can be pruned. ■

**PROPOSITION 5.4:** *Given a graph  $g$  and its generated graph  $g'$ , a frequent subgraph  $f$  along with its complete set of immediate super graphs  $S = \{f'_1, f'_2, \dots, f'_k, \dots, f'_w\}$  in  $L$ , if there is a super graph  $f'_k$  known to be present in  $g'$ , then the rest of the super graphs in  $S$  cannot be present in  $g'$  and thus can be pruned.*

**PROOF.** Suppose two super graphs,  $f'_k$  and  $f'_m \in S$  are present in  $g'$ . Since both  $f'_k$  and  $f'_m$  are immediate super graphs of  $f$ , they can be considered as generated by operators and since  $f'_k \neq f'_m$ , both these operators should be different. But graph  $g'$  is generated from  $g$  by applying only a single operator. Therefore,  $f'_k$  and  $f'_m$  cannot be present in  $g'$  at the same time. Therefore, if one super graph-  $f'_k$  present in  $g'$  then all the graphs in  $S \setminus \{f'_k\}$  can be pruned. ■

Proposition 5.1 is useful since a set of super graphs can be eliminated completely from the *isomorphism* check if its parent subgraph is not contained in  $g$ . Propositions 5.2 and 5.3 are useful to eliminate some of the frequent subgraphs when using ADD operation. Proposition 5.4 is also beneficial and avoids many *isomorphism* checks if a super graph of a frequent subgraph  $f$  is found to be present in  $g'$ , as the rest of the super graphs do not need to be checked in  $g'$ . Also, this proposition can be used to eliminate more super graphs. For example, suppose frequent subgraph  $f_1$  has a super graph  $f'$  found to be present in  $g'$ . Now if there is another frequent subgraph  $f_2$  that happens to have the same  $f'$  as its super graph, there is no need to check it again, and also all the super graphs of  $f_2$  can be safely pruned. This makes this proposition highly effective at reducing the amount of *isomorphism* check. Based on this discussion, we present the algorithm for generating a graph and a feature vector below:

---

**Algorithm 10 (*GenNextGraph*)**

---

**Input:** Graph  $g$ , frequent feature vector  $FV_g$ , operator  $op$ **Output:** Generated graph  $g'$  and corresponding feature vector  $FV_{g'}$ **Method:**

```
1: let  $FV_{g'} = FV_g$ 
2: if  $op = \text{ADD\_EDGE}$  or  $op = \text{ADD\_VERTEX}$ 
3:   select edge  $(v1, v2)$  to be added      //  $v2$  is a new vertex in  $op \text{ ADD\_VERTEX}$ 
4:   modify  $g$  by adding  $(v1, v2)$  to form  $g'$ 
5:   for each graph  $f \in FV_g$ 
6:     select immediate super graph  $f'$  in subgraph lattice  $L$ 
7:     prune  $f'$  using Propositions 5.1-5.4
8:     if not pruned check if  $f'$  is isomorphic to  $g'$       //check presence of  $f'$ 
9:     update  $FV_{g'}$  to reflect the presence of  $f'$  in  $g'$ 
10: else if  $op = \text{DEL\_EDGE}$ 
11:   select edge  $e \in g$  s.t.  $g$  will not become disconnected when  $e$  is removed
12:   if edge  $e$  exists
13:     modify  $g$  by removing  $e$  to form  $g'$ 
14:     for each graph  $f \in FV_g$ 
15:       check if  $f$  is isomorphic to  $g'$       //check presence of  $f$ 
16:       update  $FV_{g'}$  to reflect the presence of  $f$  in  $g'$ 
17:   else let  $g' = g$       //  $op \text{ DEL\_EDGE}$  is not successful
18: return  $g'$  and the feature vector  $FV_{g'}$ 
```

---

Algorithm *GenNextGraph* is used to generate a new graph  $g'$  from the current graph  $g$  by applying the operators as described in the previous section. Also, the feature vector is updated for  $g'$  to reflect the presence of frequent subgraphs. In the case of operator *DEL\_EDGE*, edges are deleted in such a way so that the resultant graph does not become disconnected. If such an edge is not present, then the original graph  $g$  is not modified.

## 5.5 Maximum Entropy Based Supervised Anomaly Detection

Over the years, several innovative supervised classification algorithms for graph-based data have been developed [DKN+05][WK06][HGW04][KMM04][LYY+05] [BB02]. Most of the algorithms are based on the underlying assumption that the intrinsic properties of a graph are rendered by its underlying substructures (nodes, edges, paths, strongly connected components, trees, subgraphs, etc). These substructure-based algorithms iden-

tify the important components present in each graph and subsequently use them to discriminate graphs from different classes. Most of the early works in graph-based classification have focused on the use of heuristic based search techniques to discover such discriminative components present in the graph database [BB02]. More recently, however, frequent subgraph based approach was proposed in which frequent subgraph mining algorithms are employed to generate all the subgraphs that occur a sufficiently large number of times in the graph database and to construct feature vectors based on the extracted subgraphs [DKN+05]. By transforming each graph into its corresponding feature vector, we can subsequently apply any of the existing classification algorithms to build a classification model for the graph database.

Among the state of the art classification algorithms includes support vector machines (SVM) [J99], *Adaboost* [SS99] [FHT00] [VV05], and *maximum entropy* model. The SVM approach has been widely applied to the classification of graph objects. Cai et al. use SVM to classify protein sequences [CWS+03]. Dobson et al. applied SVM to distinguish enzyme from non-enzyme proteins [DD03]. Much of the recent work on applying SVM to graph-based application focuses on how to build efficient and valid kernel functions for graphs. Most of these approaches are based on constructing a feature space by decomposing a graph into its underlying subgraphs and counting the number of occurrences of these subgraphs. Watkins [W99] shows that the scores produced by certain dynamic alignment algorithms can be considered as valid kernel functions. Kashima et al. [KTI03] use the counts of paths produced by *random-walks* on graph to generate the kernel. Borgwardt et al. in [BOS+05] construct the kernel by combining the similarity measures based on different data types for different source of information including sequential, structural, and chemical.

In this section, we present a principled approach for building a global classification model from the local patterns using the *maximum entropy* principle. The idea behind this approach is to learn a conditional probability distribution of the class for a graph given its

underlying features (i.e., frequent subgraphs) by using the support of the features as constraints imposed on the probability model. Using an iterative technique known as *the improved iterative-scaling* algorithm [B97], the parameters of the probability model can be estimated from the data. While the idea of using the *maximum entropy* principle for constructing a global model based on its underlying local patterns is not new [MS99], to the best of our knowledge, this approach has not been applied to graph-based data.

### 5.5.1 Maximum Entropy Model in Supervised Setting

In Section 5.4.1, we discussed how *maximum entropy* can be used in an unsupervised setting. Here we discussed the supervised case, where each graph is assigned a class label,  $y$ , chosen from a set of discrete labels  $Y$ . In this work, we assume that the classes are binary, even though the approach can be generalized to multi-class problems using one-versus-one, one-versus-all, or error correcting output coding (ECOC) methods.

Our objective is to find a global probability model  $P(y|X_g)$  using the maximum entropy principle. To apply the *maximum entropy* principle, we first define a set of features constructed from the frequent subgraphs. We then create a database of binary transactions, where each transaction corresponds to a graph  $g \in G$ . Each transaction also contains a set of binary features, also called *items*, each of which is defined as follows:

$$f_g(X_i, y) = \begin{cases} 1 & \text{if } X_i \subseteq g \\ 0 & \text{otherwise} \end{cases} \quad (5.12)$$

As previously noted, the *maximum entropy* principle seeks to find a probability model  $P^*$  that maximizes the following objective function:

$$P^* = \max_P \left[ - \sum_{g \in G} P(y|X_g) \log P(y|X_g) \right] \quad (5.13)$$

subject to the following constraints:

$$\sum_g P(y | X_g) f_g(X_i, y) = s_i \quad (5.14)$$

where  $s_i$  corresponds to the support of the subgraph  $X_i$  in the database  $G$ . Note that Equation (5.14) states that the expected value for every feature is constrained to be identical to the support of the corresponding subgraph. It can be shown that the probability model that maximizes Equation (5.13) subject to the linear constraints in Equation (5.14) has the following exponential form:

$$P^*(y | X_g) = \frac{1}{Z(X_g)} \exp \left[ \sum_i \lambda_i f_g(X_i, y) \right] \quad (5.15)$$

where  $Z(X_g)$  is a normalization factor, and  $\lambda$ 's are the parameters to be estimated.

### 5.5.2 Parameter Estimation

Let  $\Lambda = \{\lambda_1, \lambda_2, \dots, \lambda_d\}$  be the set of parameters to be estimated. The  $\Lambda$  vector can be interpreted as the significance or weight of the corresponding features and can be estimated using the maximum likelihood approach. Specifically, the likelihood function for the training data  $G$  is:

$$L(\Lambda) = \prod_{g \in G} P^*(y | X)^{\tilde{p}(X,y)} \quad (5.16)$$

where  $\tilde{p}(X,y)$  is the estimated counts of  $(X,y)$  in the training data  $G$ .

The conditional maximum likelihood model above is solved using the *Improved Iterative Scaling algorithm (IIS)* [BPP96]. *IIS* can be designed to prevent overfitting in the resulting model  $P^*(y/X)$ , by incorporating a *Gaussian Prior*. Here, instead of maximizing the likelihood function we maximize the function:

$$L(\Lambda) = \prod_{g_i \in G} P^*(y_i | X_i)^{\tilde{p}(y_i/X_i)} \times P(\Lambda) \quad (5.17)$$



where the parameters  $\Lambda$  are assumed to have a *Gaussian Prior* with zero mean and variance  $\sigma^2$ :

$$P(\Lambda) = \prod_i \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{\lambda_i^2}{2\sigma^2}\right] \quad (5.18)$$

To maximize the likelihood, we initialize  $\Lambda = \{\lambda_1, \lambda_2, \lambda_3, \dots, \lambda_m\}$  with some arbitrary values. The algorithm then seeks for a new set of parameters that will yield a higher log likelihood.

$$L'(\Lambda) = L(\Lambda + \Delta) - L(\Lambda) \quad (5.19)$$

Solving the equation above gives us a model in terms of  $\Delta$ , so taking the derivative with respect to a certain  $\delta_i$  yields:

$$\sum_{g \in G} \tilde{p}(X, y) f_i(X, y) - \sum_{g \in G} \tilde{p}(X) p^*(y/X) f_i(X, y) \exp(\delta_i f^\#(X, y)) = 0 \quad (5.20)$$

where  $f^\#(X, y) = \sum_{i=1}^n f_i(X, y)$  for a particular graph  $g$ . We solve this for each  $\delta_i$  and increment the corresponding  $\lambda_i$  iteratively. We continue with this search procedure until a stationary point of the vector  $\Lambda$  is found.

### 5.5.3 *gEntropySuper*: Classifying Anomalous Graphs

This section describes our proposed algorithm (*gEntropySuper*) based on the above framework for classifying graphs in a given graph dataset. In our approach, we first generate frequent subgraphs for a given minimum support threshold ( $\xi$ ) using a frequent subgraph mining algorithm. Once the frequent subgraphs are generated, binary feature vectors are constructed for each graph in the graph database. The  $i^{\text{th}}$  entry of the feature vector is set to one if that feature (frequent sub graph) occurs in the graph, and it is set to zero if the feature is not present.

Using these feature vectors, we repeatedly compute the model described in Equation (5.20) until it converges to a stationary point. The following algorithm describes the procedure we used to classify graphs.

---

**Algorithm 12** (*gEntropySuper*)

---

**Input:** Graph database  $G$ , min support threshold  $\xi$

Empirical distribution  $\tilde{p}(X, y)$

**Output:** Optimal parameters set  $(\lambda_i) \forall i \in \{1, 2, \dots, n\}$   
Optimal model  $p^*$

**Method:**

- 1: generate feature functions  $f_1, f_2, \dots, f_n$  for  $G$
  - 2: construct binary feature vector for each  $g \in G$
  - 3: initialize  $\lambda_i = 0 \forall i \in \{1, 2, \dots, n\}$
  - 4: repeat
  - 5:   for  $i = 1$  to  $n$  do
  - 6:     compute  $\delta_\alpha$  by solving Equation (20)
  - 7:     update  $\lambda_i \leftarrow \lambda_i + \delta_i$
  - 8:      $\forall g \in G$  compute  $P^*(y|X)$  using  $\Lambda$ .
  - 9: until convergence
- 

The time complexity of the algorithm depends on three components: generation of frequent subgraphs, construction of feature vectors, and calculation of optimal model. Let  $T_f$  be the time complexity of any existing frequent subgraph mining method. In order to construct binary feature vectors using  $n$  features with  $N$  graph objects it requires  $T_s = nN$  isomorphism checks, where subgraph isomorphism has exponential time complexity. Optimal model computation takes  $nNT_{steps}$ , where  $T_{steps}$  is the number of steps required to converge the model. Therefore, the total time complexity can be stated as  $O(T_f + nN + nNT_{steps})$ . Similarly, space complexity of the three steps can be analyzed. Let  $S_f$  be the space requirement of the underlying subgraph mining method. Let  $n$  be the frequent subgraphs generated. Then it takes  $nN$  computations to generate feature vectors. Also, the optimal parameter set require  $n$  memory space. Therefore, the space complexity of the algorithm is  $O(\max(S_f, (nN + n)))$ .

## 5.6 Experimental Evaluation

In this section, we describe the experimental environment used to evaluate our algorithms and the results obtained.

### 5.6.1 Datasets

We used four different real world datasets to evaluate the algorithms.

The first dataset, *Predictive Toxicology Data -PTC*<sup>5</sup>, contains 417 chemical compounds evaluated on four types of laboratory animals: male mouse (MM), female mouse (FM), male rat (MR), and female rat (FR). Each compound is assigned a class label indicating the toxicity (positive or negative) of the compound for that animal. Therefore, in this dataset we have four binary classification problems corresponding to each laboratory animal (MM/FM/MR/FR).

The second dataset, *Aids*<sup>6</sup>, contains 42,682 chemical compounds evaluated for evidence of anti HIV activity. We have formulated a binary classification problem, where a compound is labeled as active if it can provide protection to the human CEM cells from HIV infection, and as inactive otherwise.

The third dataset, *Cancer*<sup>7</sup>, contains 42,247 chemical compounds evaluated for evidence of the ability to inhibit the growth of human tumor cell lines. We have formulated a binary classification problem using this dataset.

The fourth dataset is the *WebSpam*<sup>8</sup> dataset containing 4,188 graphs generated for our purpose. This dataset contains a web graph, whose nodes correspond to hostnames, and edges correspond to the directed links between hosts. We first discretized this data set by calculating the average number of links between the nodes and applying a threshold to

---

<sup>5</sup> <http://www.predictive-toxicology.org/ptc/>

<sup>6</sup> [http://dtp.nci.nih.gov/docs/aids/aids\\_data.html](http://dtp.nci.nih.gov/docs/aids/aids_data.html)

<sup>7</sup> [http://dtp.nci.nih.gov/docs/cancer/cancer\\_data.html](http://dtp.nci.nih.gov/docs/cancer/cancer_data.html)

<sup>8</sup> <http://www.yr-bcn.es/webspam/datasets/>

form new edges. Then, for each node, its neighborhood is determined, and each such node and its corresponding neighborhood are used to form a single graph. This way we were able to generate multiple graphs and each graph is classified as normal or spam.

## 5.6.2 Evaluating Environment

In this section, we describe the machine environment used to evaluate our algorithms. We have performed extensive experiments on both synthetic and real data sets to evaluate the performance of our algorithms. The experiments were conducted on a SUN *Sparc* 1GHz machine with 4 GB of main memory running *Linux*.

## 5.6.3 Evaluation of the Unsupervised Frameworks

In this section, we describe the performance of our unsupervised anomaly detection algorithm – *gRank*, using the datasets discussed in Section 5.4.1. From these datasets, we selected a set of graphs randomly as shown in Table 5.1. Also, note that the anomalous graphs are also selected randomly from these datasets. We used FSG [KK01] frequent subgraph mining algorithm to generate frequent subgraphs using the minimum support thresholds as shown in Table 5.1 for each of the dataset.

Table 5.1. Characteristics of the datasets

<b>Dataset</b>	<b>Num. Instances</b>	<b>Num. Anomalies</b>	<b>Min. Support</b>
MM	215	15	10%
MR	210	20	10%
FM	215	15	10%
FR	245	20	10%
Aids	1,050	50	20%
Cancer	8,400	400	15%
WebSpam	1,050	50	30%

Table 5.2 shows the average *precision* ( $P$ ) and average *false Alarm rate* ( $FA$ ) obtained by each algorithm. Here algorithm *gRank-cos* uses *cosine* similarity of frequent subgraphs to define the similarity between graphs (Section 5.1.1.1), and *gRank-cg* uses *maximal common frequent subgraphs* to define the similarity between graphs (Section 5.1.1.2). Algorithm- *gEntropy* is the probabilistic approach based on the *maximum entropy*. We also compared our approaches with the existing anomaly detection scheme *Subdue* [NC03].

Table 5.2. Experimental results

Dataset	<i>gRank-cos</i>		<i>gRank-cg</i>		<i>gEntropy</i>		<i>Subdue</i>	
	P	FA	P	FA	P	FA	P	FA
MM	0.13	0.065	0.11	0.066	0.13	0.065	0.07	0.070
FM	0.05	0.071	0.16	0.062	0.16	0.062	0.05	0.071
MR	0.10	0.085	0.10	0.085	0.10	0.085	0.05	0.090
FR	0.05	0.084	0.05	0.084	0.05	0.084	0.05	0.084
Cancer	0.01	0.049	0.10	0.045	0.02	0.050	0.02	0.049
Aids	0.01	0.049	0.15	0.043	0.02	0.049	0.02	0.049
WebSpam	0.18	0.041	0.18	0.041	0.16	0.042	0.07	0.047

When analyzing the results of *gRank-cg* algorithm, it shows better precision than *Subdue* in all of the cases except for FR dataset, where the results are comparable. *gRank-cos* shows better precision than *gRank-cg* only in MM dataset. Furthermore, *gRank-cg* shows higher precision in *Aids* and *Cancer* datasets when compared with all other algorithms, including *gRank-cos*. Precision of *gEntropy* is comparable to that of *gRank-cos* but better than *Subdue*. In the *PTC* datasets, it showed better performance compared to *gRank-cg*. But *gRank-cg* shows better performance in all other cases. This shows the effectiveness of maximal common subgraph based features. When considering the complexity of these real-world data, the results we obtained from an unsupervised technique such as *gRank* is highly satisfactory.

### 5.6.4 Evaluation of the Supervised Frameworks

In this section, we describe the performance of our supervised anomaly detection algorithm – *gEntropySuper*, when compared with the existing methods. We analyze the performance using two different descriptor spaces: frequent subgraphs, and maximal frequent subgraphs.

In order to generate feature vectors, we used a frequent sub graph mining algorithm – FSG [KK01], and generated frequent subgraphs for a given minimum support threshold. Table 5.3 shows the selected minimum support threshold for each of the datasets. Then, binary feature vectors are constructed for each graph in the graph database as discussed in Section 5.3.2. Once the feature vectors are built, any of the existing classification algorithms (*SVM*, *Adaboost*) can potentially be used for classification.

Table 5.3. Characteristics of the datasets

<b>Dataset</b>	<b>Num. Instances</b>	<b>Min. Support</b>
MM	417	10%
MR	417	10%
FM	417	10%
FR	417	10%
Aids	42,682	20%
Cancer	42,247	10%
WebSpam	4,188	25%

In this work, classification is done by performing *5-fold cross validation* on the dataset; that is, we divide the datasets into five equal sized subsets and in each round we choose one subset as the test set and the remaining four subsets as the training set and report the average accuracy.

We used two state-of-the-art classification techniques for comparison. The first method, known as SVM (Support Vector Machine), chooses a maximum margin linear

classifier among all the existing linear classifiers. The margin is defined as the distance of the linear classifier to its nearest training samples, known as *support vectors*. This idea can be developed to non-linear case by mapping the samples to a high dimension space where they can be classified linearly. This mapping is performed by *kernel* function, which defines the distance between all pairs of samples in the new high dimensional space. Unlike traditional approach to classification, SVM minimizes the empirical classification error and maximizes the geometric margin at the same time and it has been shown that this increases the generalization power of the model (decreases the classification error of unseen examples).

The second method, known as *Adaboost*, constructs a series of weak classifiers and uses a linear combination of these classifiers as the final model. *Adaboost* uses an iterative process during which the error rate of the model decreases gradually. Given the model created from the linear combination of the current series of weak classifiers, it makes a new weak classifier by concentrating on the areas with many wrongly labeled samples. This is performed with a sampling procedure from a weighted training sample set with high weight for wrongly labeled samples and low weight for correctly labeled samples.

#### 5.6.4.1 Comparison with other approaches

We compared the performance of our algorithm (*gEntropySuper*) against two existing methods: *Adaboost* and *SVM*. For the SVM classifier, we used the SVM<sup>Light</sup><sup>9</sup> package described in [J99]. For *Adaboost*, we used the MATLAB<sup>10</sup> package. We set the number of iteration for *Adaboost* as 20. Table 5.4 shows the accuracy (Acc) and F-Measure (F) of each of these methods.

---

<sup>9</sup> <http://svmlight.joachims.org/>

<sup>10</sup> <http://research.graphicon.ru/machine-learning/gml-adaboost-matlab-toolbox.html>

Table 5.4. Experimental results (Accuracy and F-Measure)

Dataset	Adaboost		SVM		<i>gEntropySuper</i>	
	Acc	F	Acc	F	Acc	F
MM	0.584	0.58	0.561	0.50	0.582	0.59
FM	0.567	0.56	0.547	0.46	0.558	0.57
MR	0.585	0.53	0.569	0.42	0.586	0.63
FR	0.558	0.61	0.565	0.64	0.578	0.54
Cancer	0.654	0.59	0.669	0.60	0.661	0.58
Aids	0.974	0.00	0.964	0.00	0.965	0.07
WebSpam	0.935	0.68	0.906	0.00	0.934	0.67

As can be seen from the results, the *gEntropySuper* approach achieved a reasonably better accuracy compared to other approaches. In MR and FR datasets it shows the best performances. Overall, it achieved better performance than SVM in all of the datasets except in *Cancer* data. In the *WebSpam* dataset, performance of *gEntropySuper* is better than that of SVM and is comparable to *Adaboost*. *Adaboost* shows better performance in MM, FM, and *Aids* datasets when compared with *gEntropySuper*.

We have conducted a *paired t-test* to determine whether the performance of *gEntropySuper* differs from *Adaboost* in a significant way. For the t-test our null hypothesis is that the mean difference between paired observations is zero.

When comparing *Adaboost* and *gEntropySuper*, paired t-test gives  $t = -0.262$  with degrees of freedom = 6. The probability of this result, assuming the null hypothesis, is 0.802. The 95% confidence interval for mean is  $-1.0326E-02$  thru  $8.3260E-03$ . Therefore, we conclude that the accuracy of *gEntropySuper* is comparable to *Adaboost*.

On the other hand, when comparing *SVM* against *gEntropySuper*, paired- t test gives  $t = 2.58$  with degrees of freedom = 6. The probability of this result, assuming the null hypothesis, is 0.042. Also, 95% confidence interval for mean is  $6.2490E-04$  thru  $2.3089E-02$ . Therefore, a null hypothesis of no difference between the means can be rejected; the confidence interval is away from including zero. Based on this result *SVM* is different from *gEntropySuper* and as discussed earlier, it shows lower performance.



### 5.6.4.2 Classification using maximal subgraph-based descriptors

In the previous section, we used frequent subgraph based descriptor space for defining feature vectors. Instead of frequent subgraphs, here we analyze *maximal frequent subgraphs* as descriptors. For this analysis, we used 4 datasets (MM, FM, MR, FR) and generated maximal frequent subgraphs with the same support threshold (10%) used in the previous case. Table 5.5 shows the classification accuracy and F-Measure for each of the algorithms.

Use of maximal subgraph based descriptors changes the performance of different algorithms dramatically. However, while it improves the performance of SVM in terms of both accuracy and F-Measure, it reduces the performance of *Adaboost* and *gEntropySuper*. As the number of frequent subgraphs generated for these datasets is always higher than the number of maximal frequent subgraphs, these extra features provided by frequent subgraphs can be very effective for classification. *gEntropySuper* approach can automatically choose the best features for classification and thus it shows better performance with frequent subgraphs based descriptors than the maximal ones.

Table 5.5. Experimental results with maximal subgraphs

Dataset	Adaboost		SVM		<i>gEntropySuper</i>	
	Acc	F	Acc	F	Acc	F
MM	0.577	0.58	0.602	0.61	0.565	0.55
FM	0.553	0.54	0.547	0.49	0.542	0.54
MR	0.568	0.51	0.561	0.38	0.554	0.57
FR	0.603	0.63	0.592	0.68	0.563	0.50

## 5.7 Discussion

In this chapter, we investigated both unsupervised and supervised approaches for detecting anomalies in graph-based datasets.

First, we presented an unsupervised framework, called *gRank*, to determine the anomalousness of the graphs. We discussed two similarity measures to determine the similarity between graphs: (1) *cosine* similarity measure with frequent subgraph based features; (2) common subgraph based similarity measure with *maximal common frequent subgraphs*, utilizing an efficient lattice based approach for fast computation. Once the similarity between graphs is computed, a *random walk* model is used to assign an anomaly score to each graph object. Empirical studies conducted on both real and synthetic data sets showed that our proposed methods outperform previously developed anomaly detection schemes for graph based data and can effectively address the inherent problems of such schemes. Further, when analyzing the similarity between graphs, common subgraph similarity seems to be a viable solution. However our common subgraph similarity measure for graphs with repeating vertex labels is an approximation and still needs improvement to make it suitable for any graph database, such as graphs with many repeating vertices.

We have also proposed a *maximum entropy* based unsupervised anomaly detection framework, called *gEntropy*, to detect anomalies in graph based data. In this method, we assign each graph a probability value, which is then used to determine its anomalousness. The probability of a graph is determined by the weighted sum of its features in the corresponding feature vector. Using the *Generalized Iterative Scaling* algorithm optimal weights are calculated from the samples of graphs generated by *Metropolis* sampling. Experimental results using real-world data confirmed the effectiveness of this method for graph based data.

We also investigated a *maximum entropy* based supervised approach for the problem of graph classification. Similar to some of the existing methods, our algorithm is based on the frequent subgraph patterns extracted from the graph database. Instead of using these patterns directly to build feature vectors, we combined them into a coherent global model that can be used for prediction. This method, which is relatively unexplored in the context

of graphs, has the advantage of providing better accuracy and efficiency. Also, it offers coherent and consistent method for predicting the class of the graph objects.

Experimental results using real-world data sets confirmed that this approach is generally more effective at classifying most understandable graph objects. Also, the results revealed that the precision of our approach does not depend heavily on the support threshold used to generate frequent subgraph patterns, which makes our approach stable compared to existing SVM based graph classifiers.

# 6 Conclusions

In this chapter, we summarize the research contributions of this work and discuss some future research directions.

## 6.1 Summary of the Thesis

This thesis focuses on two kinds of pattern mining tasks: frequent itemset mining and anomalous pattern mining. Specifically, we developed a class of novel graph based algorithms.

### 6.1.1 Mining Frequent Itemsets

Mining frequent itemsets in databases has been studied extensively. But existing approaches have various characteristics that prohibit them from scaling-up to very large databases. In order to address this problem, we developed a graph-based approach for mining frequent closed itemsets. The main contributions of this work are:

- We introduced *PrefixGraph*, a new graph based data representation for storing compressed, important information about frequent itemset generation. This new representation has several advantages over the existing approaches: First, *PrefixGraph* uses *prefix 2-item* based bit vector projections to compress a large database

into a compact representation that avoids costly, repeated database scans. Second, a partitioned-based, divide-and-conquer technique decomposes the whole mining task into a set of smaller subtasks in such a way so that the information for frequent itemset generation is available locally, which avoids costly data structure traversals, especially when the database is very large. Finally, the variable length nature of the bit vectors, which enables fast bit vector intersections to generate frequent itemsets, makes *PrefixGraph* an effective data representation for fast frequent itemset mining.

- We also developed an efficient *PrefixGraph* based mining algorithm, *PGMiner*, for mining the *closed* set of frequent itemsets. In frequent closed itemset enumeration, checking the *closedness* of a newly generated frequent itemset is a major cost, since it requires searching the pattern-set already generated, which is in general a highly time consuming operation. To address this cost, *PGMiner* employs a graph based method, known as *network flow theory*, to efficiently detect the closedness of a frequent itemset. Unlike other existing approaches, this network flow based closedness check does not require us to store in memory the entire set of frequent closed itemsets that have been mined so far in order to check whether a candidate frequent itemset is closed. This dramatically reduces the memory usage of our algorithm, especially for low support thresholds. Also, our closedness check has minimal computational overhead when determining the closedness of a frequent itemset. The use of network flow theory with the link structure of the graph has enabled *PGMiner* to incorporate new, efficient ways of determining the closedness of frequent itemsets that would otherwise be costly, as can be seen in the existing approaches. For instance, following the *suffix-links* of the *PrefixGraph* we can easily determine the possible candidate patterns for closedness check, which is quite difficult for existing approaches without such link information. Our performance study shows that the memory usage for *PGMiner* does not

grow quite as rapidly as other existing approaches during the mining process. Furthermore, *PGMiner* is efficient and scalable to very large databases, and it outperforms existing state-of-the-art frequent closed itemset mining algorithms by an order of magnitude both in time and memory requirements.

We believe the frequent itemset mining approach developed in this thesis is a step forward towards making frequent pattern mining more scalable for large databases and it will be beneficial to many real world applications.

### 6.1.2 Mining Anomalous Patterns

Anomalous pattern mining is an important data mining problem with broad applications. Despite its importance, a major drawback of using anomaly detection algorithms in real-world applications is their high false alarm rate, which makes this an interesting research area. In this section, we discuss both unsupervised and supervised anomaly detection frameworks developed for efficient and effective mining of anomalies. The main contributions of this work are:

- We proposed *OutRank*, a stochastic-graph based model for anomalous pattern mining in an unsupervised setting for data objects that is described by a set of features. A distinct feature of this model is the representation of data as a *graph*. This representation has a key advantage over existing anomaly detection schemes, because with a graph based representation, we can consider the *outlierness* of an object with respect to the entire graph of objects. When detecting clusters of anomalies, viewing the outlierness from such a global perspective is crucial for the performance of the algorithm. We consider two approaches for constructing a graph representation of the data, based on the object similarity and number of shared neighbors between objects. The heart of this approach is the *Markov* chain model

(or *random walk*) that is built upon this graph, which assigns an anomaly score to each object. A major advantage of using this random walk on graph approach is that it can effectively capture not only the anomalous objects scattered uniformly, but also small clusters of anomalies. Using this framework, we showed that our algorithm is more robust than the existing anomaly detection schemes and can effectively address the inherent problems of such schemes. A performance study conducted on both real and synthetic data sets show that significant improvements in detection rate and a lower false alarm rate are achieved using the proposed framework.

- In many practical situations, data are not simply represented as sets of features, instead data can have more complex forms such as graphs. In this thesis, we systematically develop an anomaly detection framework to detect anomalies in such graph-based data. Existing algorithms for this task are based on the *compression* of frequently occurring substructures of the graph database. Such structural modifications can cause important information presents in the graph to be modified or even lost, which makes these algorithms show high false alarm rate. Instead, we believe a solution to this problem should consider the structural similarity among the graphs and therefore we developed an anomaly detection framework called *gRank* to detect anomalous graphs. The general idea is to use frequently occurring subgraphs to determine the *maximal common frequent subgraph* components present in the graphs and use them to compute the similarity between graph objects. Using these similarities, a graph representation of objects is constructed and modeled as a *Markov* chain, which assigns an anomaly score for each graph object. Use of frequent subgraphs enables us to discriminate normal objects from anomalous ones, since anomalous graphs are less common than the majority of the normal graphs. Also, we proposed an efficient subgraph *lattice* based technique that does not involve a costly *isomorphism* test to compute the maximal common fre-

quent components among graph objects. Empirical studies conducted on real-world graph data sets showed that our proposed anomaly detection framework based on maximal common frequent subgraphs outperforms previously developed anomaly detection schemes based on compression of substructures, by achieving higher detection rate. Furthermore, the results revealed that the maximal common frequent subgraph based similarity measure is a viable measure to determine the similarity between graphs.

- We have also studied a novel *maximum entropy* based unsupervised anomaly detection framework for graph based data. In this framework, anomalousness of a graph is determined by assigning a probability value for each graph in the database. Here, each graph is represented by a binary feature vector, where each feature corresponds to a presence or absence of a frequent subgraph discovered from the graph database. Then, the probability of a graph is computed based on the *weighted* sum of these features. The optimal *weight-set* is calculated using the *Generalized Iterative-Scaling* algorithm from the samples of data obtained by the *Metropolis* sampling approach. An efficient anomaly detection algorithm called *gEntropy* is proposed for this task. An experimental evaluation conducted using real-world datasets shows the effectiveness of this approach for anomaly detection. Also, when compared to the trivial case, *gEntropy* dramatically reduces the computational time.
- In some situations, unsupervised anomaly detection is difficult to perform and may result in lower precision. To address these situations, we proposed a probabilistic substructure-based *supervised* approach— *gEntropySuper*, for classifying graph based data. In this method, we use a frequent subgraph mining algorithm to extract frequent substructure based descriptors and apply *maximum entropy* principle to build a global classification model from these frequent subgraphs. This method, which is relatively unexplored in the context of graphs, has the advantage



of providing better accuracy and efficiency. Empirical studies conducted on real world data sets showed that the maximum entropy substructure-based approach often outperforms existing feature vector methods using *AdaBoost* and *Support Vector Machine*.

We believe our graph-based anomaly detection framework will address the typical grievances voiced by users when applying existing anomaly detection techniques, and it will be beneficial to many applications in the real world.

## 6.2 Future Research

In this section, we present some of the related problems and extensions that can be targeted in the future:

- *Mining Sequential Patterns*: Sequential pattern mining is to find frequent sequences in a sequential database. Unlike mining frequent closed itemsets, there are very few methods (CloSpan [YH03] & BIDE [WH04]) proposed for mining closed sequential patterns. This is mainly due to the complexity of the problem. Our experiments with existing algorithms confirm that none of the methods are scalable when mining long sequences, such as bio-sequences. The *PrefixGraph* approach we proposed has some interesting features: first, it is a partitioned based divide and conquer algorithm built to improve scalability; second, it is a bit-vector based data structure that enables fast pattern generation. These advantages open the possibility of extending *PrefixGraph* for sequence mining. However, several issues need to be addressed when mining for long sequences. The projection of sequences to nodes of the graph should be modified so that the nodes at the far end of the *PrefixGraph* do not contain a larger database. Also, the use of flow-

based pruning strategy to prune the non-closed sequential patterns is an interesting issue to be addressed in future research.

- *Mining Very Large Disk Resident Databases*: The scalability of any in-core algorithm, such as *PGMiner*, is limited by the available memory capacity and, when mining large databases of hundreds of gigabytes, different approaches are needed. Therefore, it is interesting to develop closed itemset mining methods that can scale to very large databases using secondary memory. When developing a disk based mining approach for a closed itemset mining task, the following two key issues need to be addressed: (1) Partition strategy: Partitioning method involves partitioning of the database such that each such partition can be held in available memory. When designing a partitioning strategy, the number of partitions and the cost of search space enumeration for mining must be balanced. Otherwise, disk I/O cost can become high and will be a major bottleneck that affects the performance of the underlying algorithm; (2) Closed itemset enumeration: As mentioned earlier closedness of an itemset cannot be decided based on the knowledge available in a single partition. Therefore, efficient closed itemset enumeration from the local closed itemsets of each partition is an issue that needs to be addressed. *PGMiner* can be a good choice for this task because of the partitioned based nature of the *PrefixGraph*, and the independence nature of itemset enumeration task.
- *Mining Anomalous Substructures*: Real world graph datasets are not always a collection of graphs. Instead, the whole database can be a single large graph. The anomalous substructure detection is to examine the entire graph and mine anomalous substructures contained in it. These anomalous substructures are not simply the substructures occurring infrequently, since very large substructures occur once or twice. For instance, the entire graph can be considered as a substructure that occurs once in it. What we need to look for when mining such a large graph is the patterns that have an unusual occurrence inside the graph. Such patterns that devi-

ate from the majority of the substructures can be considered as anomalous. Care should be taken not to identify the large substructures as unusual. These challenges need to be addressed by any effective anomaly detection algorithm. Furthermore, graph based methods need to address the complexity of handling such a large graph, as the real-world graphs of this nature can have millions of nodes (e.g. telephone networks or social networks). Substructure similarity based techniques introduced in *gRank* open new opportunities when addressing these challenges.

- *Mining Anomalous Graphs by Incorporating Labeled Samples:* We have introduced an unsupervised framework for anomaly detection in graph based data. This method does not use any previous known knowledge about the dataset when detecting anomalies. However, if we have prior knowledge about that data, then the anomaly detection algorithm can be guided to detect normal and abnormal graphs in the data. This not only improves the detection rate, but also reduces the false alarm rate. In semi-supervised anomaly detection, a small amount of labeled data is mixed with a large amount of unlabeled data to obtain an improvement. With labeled information, some of the drawbacks present in the current unsupervised framework can be minimized. Determining the similarity of a small graph against a large graph and selecting the best discriminative substructures are some of the situations that can really be improved with labeled information. How to incorporate such information is a challenge that needs to be addressed in the future. This type of semi-supervised framework can also be extended to mining anomalous substructures present in a large graph.

## BIBLIOGRAPHY

- [AAP+98] R. Agarwal, C. Aggarwal, V. Prasad, and V. Crestana. A tree projection algorithm for generation of large itemsets for association rules. *IBM Research Report*, RC21341, November 1998.
- [AIS93] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. of the ACM SIGMOD conference on management of data*, pages 207-216, 1993.
- [Alm] IBM Almaden. *Synthetic Data Generation Code for Associations and Sequential Patterns*. <http://www.almaden.ibm.com/cs/quest/syndata.html>
- [AS94b] R. Agrawal and R. Srikant. Fast Algorithms for mining Association rules. In *Proceedings of VLDB*, September, 1994.
- [AS96] R. Agrawal and J. C. Shafer. Parallel mining of association rules. *IEEE Trans. On Knowledge and Data Engineering*, 8:962-969, 1996.
- [AW06] R. Angelova and G. Weikum. Graph-based text classification: learn from your neighbors. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, 2006.
- [B97] A. Berger. The Improved Iterative Scaling Algorithm: A gentle Introduction. Technical report, Carnegie Mellon University, 1997.
- [BB02] M.R. Berthold and C. Borgelt. Mining Molecular Fragments: Finding Relevant Substructures of Molecules, In *Proc. Int'l Conf. on Data Mining*, 2002.
- [BCG01] D. Burdick, M. Calimlim, and J. Gehrke. MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases. In *Proc. of ICDE*, 2001.
- [BKN+00] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. LOF: identifying density-based local outliers. In *Proc. of the 2000 ACM SIGMOD Int'l Conf. on Management of data*, pages 93–104, 2000.
- [BOS+05] K.M. Borgwardt, C.S. Ong, S. Schönauer, S. Vishwanathan, A.J. Smola, and H.P. Kriegel. Protein function prediction via graph kernels. *Bioinformatics* 21(1), 2005.
- [BPG06] G. Buehrer, S. Parthasarathy, and A. Ghoting. Out-of-core Frequent Pattern Mining on a Commodity PC. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, August 2006.

- [BPP96] A.L. Berger, V.J. Della Pietra, and S.A. Della Pietra. Maximum Entropy Approach for Natural Language Processing. *Computational Linguistics*. 22(1) P: 39 – 71, 1996.
- [BS03] S. D. Bay and M. Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Proc. of the ninth ACM SIGKDD Int'l Conf. on Knowledge discovery and data mining*, pages 29–38, 2003.
- [CH00] D. J. Cook , L. B. Holder, Graph-Based Data Mining, *IEEE Intelligent Systems*, v.15 n.2, p.32-41, 2000.
- [CHN+96] D. Cheung, J. Han, V. T. Ng, A. W. Fuan, and Y. Fu. A Fast Distributed Algorithm for Mining Association Rules. In *Proc. of 1996 Int'l Conf. on Parallel and Distributed Information Systems (PDIS'96)*, Miami Beach, Florida, USA, December, 1996.
- [CHX98] D. Cheung, K. Hu and S. Xia. Asynchronous Parallel Algorithm for Mining Association Rules on a Shared-memory Multi-processors. In *Proc. of the Tenth Annual ACM Symposium on Parallel Algorithms And Architectures (SPAA98)*, Puerto Vallarta, Mexico, June, 1998.
- [CWS+03] C.Z Cai., W.L. Wang, L.Z. Sun, and Y.Z. Chen. Protein function classification via support vector machine approach. *Math. Biosci.*, 185, 111–122, 2003.
- [CX98] D. Cheung and Y. Xiao. Effect of Data Skewness in Parallel Mining of Association Rules. In *Proc. of the Pacific-Asia Conf. Knowledge Discovery and Data Mining*, Lecture Notes in Computer Science, Vol. 1394, Springer Verlag, New York, 1998.
- [DD03] P.D. Dobson and A.J. Doig, Distinguishing enzyme structures from non-enzymes without alignments. *J. Mol. Biol.*, 330, 771–783, 2003.
- [DKN+05] M. Deshpande, M. Kuramochi, Nikil Wale, and G. Karypis, Frequent Substructure-Based Approaches for Classifying Chemical Compounds, *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 8, pp. 1036-1050, 2005.
- [DKW+05] M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis. Frequent Substructure-Based Approaches for Classifying Chemical Compounds. *IEEE Transactions on Knowledge and Data Engineering*, 2005.
- [DR72] J.N. Darroch and D. Ratcliff. Generalized iterative scaling for log-linear models. *Ann. Math. Statist.*, 43:1470--1480, 1972.

- [Esk00] E. Eskin. Anomaly detection over noisy data using learned probability distributions. In *Proc. of the 17th Int'l Conf. on Machine Learning*, pages 255–262, 2000.
- [FHT00] J. Friedman, T. Hastie, and Robert Tibshirani. Additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, 38(2):337–374, 2000.
- [FPS96] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, *Advances in Knowledge Discovery and Data Mining*, AAAI/MIT Press, 1996.
- [GZ03] G. Grahne, J. Zhu. Efficiently Using Prefix-trees in Mining Frequent Itemsets. In *Proc. of FIMI'03*, 2003.
- [GZ04] G. Grahne, and J. Zhu. Mining Frequent Itemsets from Secondary Memory. In *Proc. of IEEE International Conference on Data Mining*, 2004.
- [H98] D. J. Hand. Data mining: statistics and more?, *The American Statistician*, 52, 112--119, 1998.
- [Haw80] D. Hawkins. *Identification of outliers*. Chapman and Hall, London, 1980.
- [HCL07] J. He, J. Carbonell, and Y. Liu. Graph-Based Semi-Supervised Learning as a Generative Model, In *Proc. of International Joint Conference on Artificial Intelligence*, 2007.
- [HCH+98] R.J. Hilderman, C.L. Carter, H.J. Hamilton, and N. Cercone. Mining market basket data using share measures and characterized itemsets, In *Proc of Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 159–170, 1998.
- [HGW04] T. Horvath, T. Grtner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In *Proc. of the 10th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 158-167, 2004.
- [HIK06] Y. Haxhimusa, A. Ion, and W. G. Kropatsch. Evaluating Hierarchical Graph-based Segmentation. In *Proc of the 18th International Conference on Pattern Recognition (ICPR'06)*, 2006.
- [HKK00] E. H. Han, G. Karypis, and V. Kumar. Scalable parallel data mining for association rules. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 12, No. 3, 2000.
- [HLN99] J. Han, V.S. Lakshmanan, and R. T. Ng. Constraint-Based, Multidimensional Data Mining, *Computer*, IEEE Computer Society, vol. 32, no. 8, pp. 46-50, August, 1999.

- [HPY00] J. Han, J. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. In *proc. of ACM SIGMOD*, 2000.
- [HWL+02] J. Han, J. Wang, Y. Lu, and P. Tzvetkov. Mining Top-K Frequent Closed Patterns without Minimum Support, In *Proc. of Int. Conf. on Data Mining (ICDM'02)*, Maebashi, Japan, Dec. 2002.
- [HWP03] L. Huan, W. Wang, J. Prins. Efficient Mining of Frequent Subgraphs in the Presence of Isomorphism. In *Proceedings of the 2003 International Conference on Data Mining (ICDM2003)*, 2003.
- [HWP04] L. Huan, W. Wang, and J. Prins. SPIN: Mining Maximal Frequent Subgraphs from Graph Databases. In *Proceedings of the 2004 Conference on Knowledge Discovery and Data Mining (SIGKDD2004)*, 2004.
- [IK04] T. Ide and H. Kashima. Eigenspace-based Anomaly Detection in Computer Systems. In *Proc. 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD2004)*, 2004.
- [IM76] D. L. Isaacson and R.W. Madsen, *Markov chains: theory and applications*, Wiley, New York, 1976.
- [J99] T. Joachims. *Making large-Scale SVM Learning Practical*. Advances in Kernel Methods - Support Vector Learning, B. Schölkopf and C. Burges and A. Smola (ed.), MIT Press, 1999.
- [JKN98] T. Johnson, I. Kwok, and R. T. Ng. Fast computation of 2-dimensional depth contours. In *Proc. of the Fourth Int'l Conf. on Knowledge Discovery and Data Mining*, pages 224–228, 1998.
- [JTH01] W. Jin, A. K. H. Tung, and J. Han. Mining top-n local outliers in large databases. In *Proc. of the Seventh ACM SIGKDD Int'l Conf. on Knowledge discovery and data mining*, pages 293–298, 2001.
- [Kar] G. Karypis. *Synthetic Data Generation Code for graphs*. <http://glaros.dtc.umn.edu/gkhome/software>.
- [KHK99] G. Karypis, E.-H. Han, and V. Kumar. Chameleon: A Hierarchical Clustering Algorithm using Dynamic Modeling. *IEEE Computer: Special Issue on Data Analysis and Mining*, 32(8):68--75, 1999.
- [KK01] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proc of IEEE International Conference on Data Mining (ICDM)*, 2001.
- [KMM04] T. Kudo, E. Maeda, and Y. Matsumoto. An Application of Boosting to Graph Classification, *NIPS* 2004.

- [KNT00] E. M. Knorr, R. T. Ng, and V. Tucakov. Distance-based outliers: Algorithms and applications. *VLDB Journal*, 8(3-4):237–253, 2000.
- [KTI03] H. Kashima, K. Tsuda, and A. Inokuchi, Marginalized kernels between labeled graphs. In *Proceedings of 20th International Conference on Machine Learning (ICML 2003)*, Washington, DC, 2003.
- [KV03] C. Kruegel and G. Vigna. Anomaly Detection of Web-Based Attacks. In *Proc. of 10th ACM Conf. Computer and Comm. Security (CCS '03)*, pp. 251–261, Oct. 2003.
- [KW99] S. Khudanpur, and J. Wu. A Maximum Entropy Language Model Integrating N- Grams and Topic Dependencies for Conversational Speech Recognition. In *Proc. of ICASSP*, 1999.
- [Lew94] V. B. T. Lewis. *Outliers in statistical data*. John Wiley & Sons, Chichester, 1994.
- [LOP06a] C. Lucchese, S. Orlando, and R. Perego. Fast and Memory Efficient Mining of Frequent Closed Itemsets. *TKDE*, 2006.
- [LOP06b] C. Lucchese, S. Orlando, and R. Perego. Mining Frequent Closed Itemsets Out-Of-Core. In *Proceedings of the SIAM International Conference on Data Mining*, 2006.
- [LYY+05] C. Liu, X. Yan, H. Yu, J. Han, and P. S. Yu. Mining Behavior Graphs for Backtrace of Noncrashing Bugs. In *Proc. of SIAM Int. Conf. on Data Mining (SDM'05)*, 2005.
- [MS99] H. Mannila, and P. Smyth. Prediction with Local Patterns using Cross-Entropy. In *Proc. of the fifth ACM SIGKDD*, P: 357 – 361, ISBN:1-58113-143-7, 1999.
- [NC03] C. Noble and D. J. Cook, Graph-Based Anomaly Detection. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003.
- [NK04] S. Nijssen and Joost N. Kok. A quick start in frequent structure mining can make a difference. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2004.
- [NLM99] K. Nigam, J. Lafferty, and A. Maccallum. Using Maximum Entropy for Text Classification. *IJCAI-99 Workshop on Machine Learning for Information*, 1999.
- [OLP+03] S. Orlando, C. Lucchese, Paolo Palmerini, Raffaele Perego, and Fabrizio Silvestri: kDCI: a Multi-Strategy Algorithm for Mining Frequent Sets. In *Proc of FIMI*, 2003.



- [PB06] A. Pozdnoukhov, and S. Bengio. Graph-based transformation manifolds for invariant pattern recognition with kernel methods, In *18th International Conference on Pattern Recognition (ICPR'06)*, 2006.
- [PBM+98] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. *Technical report, Stanford University*, 1998.
- [PBT+99a] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *ICDT'99*, 1999.
- [PBT+99b] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Efficient mining of association rules using closed itemset lattices. *Information Systems*, 24(1):25-46, 1999.
- [PCY95] J. S. Park, M. Chen, and P. S. Yu. Efficient parallel data mining for association rules. In *ACM Intl. Conf. on Information and Knowledge Management*, 1995.
- [PHM00] J. Pei, J. Han, and R. Mao. CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets. In *Proc. of DMKD'00*, 2000.
- [PS88] F. Preparata and M. Shamos. *Computational Geometry: an Introduction*. Springer, 1988.
- [RRS00] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. In *Proc. of the ACM SIGMOD Int'l Conf. on Management of data*, pages 427-438, 2000.
- [S98] G. Strang. *Introduction to Linear Algebra*. Wellesley-Cambridge Press, 3rd edition, 1998.
- [SHR06] H. Shin, N. J. Hill and G. Rätsch. Graph Based Semi-Supervised Learning with Sharper Edges. *Lecture Note on Artificial Intelligence (LNAI) 4212*, 401-412, Springer-Verlag, 2006.
- [SK98a] T. Shintani and M. Kitsuregawa. Parallel mining algorithms for generalized association rules with classification hierarchy. In *Proc. of ACM SIGMOD Int'l Conf. on Management of Data*, pages 25--36, 1998.
- [SK96] T. Shintani, and M. Kitsuregawa. Hash based parallel algorithms for mining association rules. In *Proc. of 4th Int. Conf. on Parallel and Distributed Information Systems*, 1996.
- [SON95] A. Sarasere, E. Omiecinsky, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *Proc. 21st International Conference on Very Large Databases (VLDB)*, Zurich, Switzerland, Also Gatech Technical Report No. GIT-CC-95-04., 1995.

- [SS99] R.E. Schapire and Y. Singer Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297-336, December 1999.
- [SSM05] N. G. Singh, S. R. Singh, and A. K. Mahanta. CloseMiner: Discovering Frequent Closed Itemsets Using Frequent Closed Tidsets. In *Proc. of ICDM*, 2005.
- [SWH+05] L. K. Saul, K. Q. Weinberger, J. H. Ham, F. Sha, and D. D. Lee, Spectral methods for dimensionality reduction. In *Semi-Supervised Learning*, O. Chapelle, A. Zien, and B. Schölkopf, Eds. MIT Press, 2005.
- [TFP06] H. Tong, C. Faloutsos, and J.-Y. Pan. Fast Random Walk with Restart and Its Application. In *Proc of ICDM*, 2006.
- [TSK06] P. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*, Addison Wesley, 2006.
- [VV05] A. Vezhnevets, and V. Vezhnevets. Modest AdaBoost - Teaching AdaBoost to Generalize Better. In *Proc of Graphicon*, Novosibirsk Akademgorodok, Russia, 2005.
- [W99] C. Watkins. Dynamic Alignment Kernels, Department of Computer Science, Royal Holloway, University of London, Technical Report, CSD-TR-98-11, 1999.
- [WH04] J. Wang and J. Han. Bide: Efficient mining of frequent closed sequences. In *Proc. of ICDE*, 2004.
- [WHP03] J. Wang, J. Han, and J. Pei. CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets. In *Proc of ACM SIGKDD*, 2003.
- [WK06] N. Wale and G. Karypis. Acyclic Subgraph-based Descriptor Spaces for Chemical Compound Retrieval and Classification. In *Proc of IEEE International Conference on Data Mining (ICDM)*, 2006.
- [YH02] X. Yan, and J. Han. gSpan: Graph-based substructure pattern mining. In *Proc. of Int'l Conf. on Data Mining (ICDM)*, 2002.
- [YH03] X. Yan, J. Han. CloseGraph: Mining Closed Frequent Graph Patterns. In: *Proceedings of the 2003 Conference on Knowledge Discovery and Data Mining (SIGKDD2003)*, 2003.
- [YHA03] X. Yan, J. Han, and R. Afshar. CloSpan: Mining Closed Sequential Patterns in Large Databases. In *Proc. of SDM'03*, San Francisco, CA, 2003.
- [ZCM97] E. W. Zegura, K. L. Calvert, M.J. Donahoo. A quantitative comparison of graph-based models for Internet topology. *IEEE/ACM Trans. Networking*, 1997.

- [ZEL01] O. R. Zaïane, M. El-Hajj, and P. Lu. Fast Parallel Association Rule Mining Without Candidacy Generation. In *Proc. of the IEEE 2001 International Conference on Data Mining (ICDM'2001)*, San Jose, CA, USA, November, 2001.
- [ZG03] M. J. Zaki, and K. Gouda. Fast vertical mining using diffsets. In *Proc. of ACM SIGKDD*, 2003.
- [ZH02] M. J. Zaki, and C. J. Hsiao. CHARM: An Efficient Algorithm for Closed Itemset Mining. In *Proc of SDM'02*, 2002.
- [ZPO+98] M. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. Parallel algorithms for discovery of association rules. *Data Mining and Knowledge Discovery*, pages 343--373, 1998.
- [ZR01] X. Zhu and R. Rosenfeld. Improving Trigram Language Modeling with the World Wide Web. In *Proc of ICASSP*, P:533–536, 2001.

MICHIGAN STATE UNIVERS



3 1293 03062