

MINING LOCAL AND GLOBAL PATTERNS FOR COMPLEX DATA CLASSIFICATION

By

Geng Li

A Dissertation Submitted to the Graduate
Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: COMPUTER SCIENCE

Examining Committee:

Mohammed J. Zaki, Dissertation Adviser

Mark Goldberg, Member

Malik Magdon-Ismail, Member

Mark J. Embrechts, Member

Rensselaer Polytechnic Institute
Troy, New York

July 2013
(For Graduation August 2013)

© Copyright 2013
by
Geng Li
All Rights Reserved

CONTENTS

| | |
|---|------|
| LIST OF TABLES | vi |
| LIST OF FIGURES | viii |
| ACKNOWLEDGMENT | ix |
| ABSTRACT | xi |
| 1. INTRODUCTION | 1 |
| 1.1 Motivation | 1 |
| 1.2 Frequent Pattern Mining (FPM) | 2 |
| 1.2.1 Preliminaries | 3 |
| 1.2.1.1 Database | 3 |
| 1.2.1.2 Pattern Partial Order Graph | 4 |
| 1.2.2 Applications | 6 |
| 1.2.3 Algorithms | 7 |
| 1.2.3.1 Frequent Pattern Mining | 7 |
| 1.2.3.2 Frequent Pattern Summarization | 8 |
| 1.3 Graph Theory | 9 |
| 1.3.1 Preliminaries | 9 |
| 1.3.2 Algebraic Graph Theory | 11 |
| 1.4 Challenges | 12 |
| 1.5 Contributions | 13 |
| 2. RELATED WORK | 15 |
| 2.1 Frequent Pattern Summarization | 15 |
| 2.2 Frequent Pattern Sampling | 16 |
| 2.2.1 Itemset Sampling | 16 |
| 2.2.2 Graph Sampling | 19 |
| 2.2.3 Input Space versus Output Space Sampling | 19 |
| 2.2.4 Single versus Parallel Threads | 22 |
| 2.3 Pattern-based (Itemset) Classification | 23 |
| 2.4 Graph Classification | 23 |
| 2.4.1 Graph Kernels | 24 |
| 2.4.2 Frequent Subgraph Mining for Classification | 26 |

| | | |
|---------|---|----|
| 3. | SAMPLING MINIMAL FREQUENT BOOLEAN (DNF) PATTERNS . . . | 27 |
| 3.1 | Introduction | 27 |
| 3.1.1 | Preliminaries | 28 |
| 3.2 | Minimal Boolean Expressions | 31 |
| 3.3 | minDNF Sampling Algorithm | 33 |
| 3.3.1 | Sampling in Clause-wise State Space | 34 |
| 3.3.2 | Sampling in Item-wise State Space | 35 |
| 3.3.3 | Optimizations | 38 |
| 3.3.4 | Sampling Minimal AND-clauses | 41 |
| 3.4 | Experiments | 43 |
| 3.4.1 | Classification Performance | 43 |
| 3.4.2 | Sampling Evaluation | 49 |
| 3.5 | Conclusions | 54 |
| 4. | GRAPH CLASSIFICATION USING GLOBAL TOPOLOGICAL PAT- TERNS | 56 |
| 4.1 | Introduction | 56 |
| 4.2 | Graph Attributes for Classification | 58 |
| 4.3 | Experiments | 65 |
| 4.3.1 | Experimental Setup | 65 |
| 4.3.2 | Datasets | 68 |
| 4.3.3 | Graph Kernel Comparison | 69 |
| 4.3.3.1 | Chemical Compound Datasets | 69 |
| 4.3.3.2 | Protein Datasets | 72 |
| 4.3.3.3 | Cell-Graph Datasets | 74 |
| 4.3.3.4 | SVM Training Times | 76 |
| 4.3.3.5 | Scalability Study | 77 |
| 4.3.4 | Frequent Subgraph Features | 78 |
| 4.3.5 | Feature Importance Study | 79 |
| 4.3.6 | Augmented Topological Features | 86 |
| 4.4 | Conclusions | 89 |
| 5. | FUTURE WORK | 91 |
| 5.1 | Parallel MCMC Sampling | 91 |
| 5.2 | Subspace Sampling for Multiview Clusterings | 92 |

| | | |
|-------|--|-----|
| 5.2.1 | Introduction | 92 |
| 5.2.2 | Fundamentals | 93 |
| | 5.2.2.1 Subspace and Multiview Subspaces | 93 |
| | 5.2.2.2 Mean Shift Clustering Algorithm | 95 |
| 5.2.3 | Subspace Sampling Algorithm | 98 |
| | 5.2.3.1 Subspace Quality | 98 |
| | 5.2.3.2 Multiview Subspace Sampling | 99 |
| 5.2.4 | Optimizations | 103 |
| 5.2.5 | Preliminary Experiments | 104 |
| | LITERATURE CITED | 109 |

LIST OF TABLES

| | | |
|------|--|----|
| 1.1 | Dataset \mathcal{D} (a) and its transpose \mathcal{D}^T (b) | 3 |
| 3.1 | Classification Performance: Accuracy \pm Standard Error: cls denotes #classes | 42 |
| 3.2 | Time (in sec) for minDNF and minDNF* | 45 |
| 3.3 | Averaged Clauses Length & Items Length of minDNF and minAND | 47 |
| 3.4 | Classification Performance (Augmented Experiments): Accuracy \pm Standard Error | 50 |
| 3.5 | Datasets | 52 |
| 3.6 | Sampling Statistics: IBM100 | 52 |
| 3.7 | Effect of Parameters: IBM100 | 53 |
| 3.8 | Running Time: minDNF and minDNF* | 54 |
| 4.1 | Global graph feature vector for the example (F: Feature, V: Value) | 60 |
| 4.2 | Benchmark Datasets | 67 |
| 4.3 | Accuracy (\pm Standard Deviation): Chemical Compound Datasets (bold <i>t</i> -statistic means statistically significant) | 70 |
| 4.4 | Running Times on Chemical Compound Datasets (h:hours, m:minutes, s:seconds) | 70 |
| 4.5 | Accuracy (\pm Standard Deviation): Chemical Compound Datasets Without Labels | 70 |
| 4.6 | Accuracy (\pm Standard Deviation): (a) Protein Datasets, (b) Protein Datasets without Labels. (bold values indicate best performance, and bold <i>t</i> -statistic means statistically significant) | 72 |
| 4.7 | Running Times on Protein Datasets (h:hours, m:minutes, s:seconds) | 73 |
| 4.8 | Accuracy (\pm Standard Deviation) on Cell-Graph Datasets – E: Breast, O: Bone, and A: Brain. (bold <i>t</i> -statistic means statistically significant) | 75 |
| 4.9 | Running Times on Cell-Graph Datasets (h:hours, m:minutes, s:seconds) | 76 |
| 4.10 | Feature rankings – range normalization | 80 |

| | | |
|------|--|----|
| 4.11 | Feature rankings – z normalization | 81 |
| 4.12 | Frequent Subgraphs as Binary Features | 87 |
| 4.13 | Accuracy (\pm Standard Deviation) on Cell-Graph Datasets (E: Breast, O: Bone, and A: Brain. GF*: features f_{11} , f_{12} removed. The best results for GF and GF* are shown in bold) | 87 |
| 4.14 | Accuracy (\pm Standard Deviation) on selected Datasets | 89 |
| 4.15 | Running Times on selected Datasets (h:hours, m:minutes, s:seconds) . . | 89 |

LIST OF FIGURES

| | | |
|-----|--|-----|
| 1.1 | Itemset Partial Order Graph [1] | 5 |
| 1.2 | Closed and Maximal Itemsets in Table 1.1 [2] | 6 |
| 1.3 | A graph example G_e | 9 |
| 3.1 | Clause- and Item-wise State Space | 34 |
| 3.2 | minDNF Sampling Algorithm | 37 |
| 3.3 | Time (in sec) for minDNF (white dots) and minDNF* (black dots) . . . | 44 |
| 3.4 | Sampling Quality (RWRJ): IBM100 | 51 |
| 3.5 | Sampling Quality (RWR): IBM100 | 51 |
| 4.1 | A triangle with its three triples | 58 |
| 4.2 | A chemical compound from PTC dataset (with implicit hydrogens) . . . | 62 |
| 4.3 | The graph representation for Figure 4.2 (labels on node/atoms: O:Oxygen, H:Hydrogens, N:Nitrogen, C:Carbon. Labels on edges/bonds: A:Aromatic, S:Single, D:Double) | 62 |
| 4.4 | SVM Training Times | 77 |
| 4.5 | Scalability Study: GF(z) Time for various Replication Factors | 78 |
| 4.6 | Breast Cell-Graphs: Class Specific Average Degree (a) and Clustering Coefficient Distribution (b) | 85 |
| 5.1 | Search space for subspaces when $d = 4$ | 94 |
| 5.2 | A synthetic dataset | 104 |
| 5.3 | Experiments on Synthetic Datasets | 106 |

ACKNOWLEDGMENT

I am be indebted to many people for their support and help when I was pursuing my Ph.D degree.

First of all, I would like to extend my sincere gratitude to my research adviser, Prof. Mohammed J. Zaki for his helpful advice and endless support. He has always been there and ready to help. His stringent academic attitude, rich source of expertise, solid fundamental skills and broad research vision always inspire and guide me. He taught me how to think independently and critically. He is a very nice person and I enjoyed many pleasant and fruitful discussions with him. Without his continuous guidance, the completion of this thesis would not be possible.

Second, I would like to thank my doctoral committee. Prof. Mark Goldberg taught me the course Computability and Complexity. Prof. Malik Magdon-Ismael taught me Machine Learning and Computational Finance. They provided me much career guidance beyond the class scope. Prof. Mark J. Embrechts comes from Industrial and Systems Engineering Department, and he gave me many useful suggestions for my thesis from new perspectives.

Thank are also due to my former Data Mining lab members: Dr. Mohammad Al Hasan and Dr. Vineet Chaoji. They graduated soon after I became a member in the lab and have already started in their new careers. Dr. Mohammad Al Hasan gave me many valuable suggestions for research in Data Mining and C++ template coding. Dr. Vineet Chaoji was my co-author for the ABACUS arbitrary shape clustering paper. He was really a great collaborator with lots of insightful feedback. I also want to thank Dr. Hilmi Yildirim, who is now in Google, for his generous sharing of academic and industrial experience constantly. I finally want to thank the newer lab members: Pranay Anchuri, Nilothpal Talukder and Haiqiong Li. It was pleasant to share the lab with you guys.

I have benefited from my interactions with with several others that deserve mention. I would like to thank Dr. Garret Swart from Oracle. I was interning under his supervision in Oracle in summer 2012 and I really learnt a lot from him.

Also I want to thank Prof. Bülent Yener, for our collaboration on graph classification paper. A special thank must go to our department staff, in particular Chris Coonrad and Terry Hayden, who have been providing administrative support and organizing department activities constantly.

Last but not the least, my genuine gratefulness must go to my family. To my parents who raised me up in China and supported me with all their heart during my PhD, from the beginning to the end, in the United States. This thesis is dedicated to you.

ABSTRACT

Pattern mining is an important data mining technique that involves extracting interesting (e.g., frequent) patterns from complex data. There is abundant work published in this research area in the past and lots of progress has been made, ranging from itemset mining, to sequence mining and graph mining. One significant application of pattern mining lies in its use for effective classification, since pattern mining can help discover structure in complex domains. The basic idea is that interesting patterns can be defined as new features, which can then be used to build a classifier. However, with the growing complexity of the data as well as the types of patterns and groups sought, traditional methods based on complete enumeration of all interested patterns suffers in terms of either the time complexity or memory constraint problem, which usually make the computation very expensive or even intractable. This is especially common for dense datasets and complex graph data.

In the first part of this thesis, we tackle the challenging problem of mining the simplest Boolean patterns from categorical datasets, which are subsequently used as features for classification. Enumerating all possible frequent Boolean patterns is prohibitive in most real-world datasets, so instead of complete enumeration, which is typically infeasible for this class of patterns, we propose the first approach to generate a near-uniform sample of the minimal Boolean expressions. We make both theoretical and practical contributions. Our method, based on Markov Chain Monte Carlo (MCMC) sampling, yields a succinct subset of the simplest frequent Boolean patterns. In our method, each Markov state can be taken to be a Boolean expression, with transitions allowed only between parent and child expressions. Starting from the empty expression, we use Monte Carlo methods to perform random walks with designed probability transition matrix P in the expression space to sample the minimal expressions. We propose a novel theoretical characterization of the minimal Disjunctive Normal Form (DNF) expressions, which allows us to prune the MCMC search space effectively. When combined with other optimization techniques, our method is also practically effective. For instance, we are able to sample interesting

support-less patterns, i.e., where minimum frequency threshold is set to one. In order to demonstrate the effectiveness of our method, we perform an extensive set of experiments. In particular, we classify a variety of datasets from the UCI Machine Learning Repository, and show that minimal DNF patterns make very effective features for classification; much more so than purely conjunctive features. We also study the sample quality of our approach, as well as its scalability.

In the second part of this thesis, we study the challenging problem of graph classification. With the proliferation of graph data, there has been a lot of interest in recent years to develop effective methods for classifying graph objects. Various graph kernel methods have been proposed recently. These methods have proven to be effective, but they tend to have high computational overhead. We propose an alternative approach to graph classification that is based on easily computable feature-vectors constructed from different global topological pattern attributes, as well as global label-based features. The main idea is that the graphs from the same class should have similar topological patterns and label attributes. Our method is simple to implement, and via a detailed comparison on real benchmark datasets, we show that our topological and label feature-based approach delivers competitive classification accuracy, with significantly better results on those datasets that have large unlabeled graph instances. Our method is also substantially faster than most other graph kernels.

CHAPTER 1

INTRODUCTION

With the fast development of the digital communication and software techniques, an overwhelming amount of data is being generated nowadays. At the same time, the rapid progress of computer hardware has made the storage of “big data” a reality. People are no longer satisfied with the information obtained from simple database queries and statistical analyses. Often they are more interested in finding interesting and hidden knowledge from the data. Data mining and related techniques have become an increasingly popular computer science discipline. Pattern mining is an important data mining technique that involves extracting interesting (e.g., frequent) patterns from the complex data (e.g., data items are graph structured or have complex logical relations). The two most prominent frequent pattern mining branches are *Itemset Mining* and *Graph Mining*, which both could be used to find interesting patterns to capture regions of high contrast for construction of effective classifiers.

1.1 Motivation

Traditional pattern mining techniques usually lack of scalability. The inherent algorithmic mechanism usually makes them costly or prohibitive on most large real world datasets, since the combinatorial search space of solutions grows exponentially. One straightforward way to tackle this challenging problem is to apply sampling methods. In the first part of this thesis, our main concern is on how sampling methods can be designed to effectively extract interesting patterns from large datasets. Several interesting questions arise, such as: How to design good sampling methods for model space exploration and estimation? How to effectively sample a

Portions of this chapter previously appeared as: G. Li and M. Zaki, “Sampling Minimal Frequent Boolean (DNF) Patterns,” In *Proceedings of the 18th SIGKDD International Conference on Knowledge Discovery and Data Mining*, Beijing, China, 2012, pp. 87–95, and G. Li, M. Semerci, B. Yener and M. Zaki, “Effective Graph Classification based on Topological and Label Attributes,” *Statistical Analysis and Data Mining*, vol. 5, no. 1, pp. 265–283, August 2012.

representative subset of patterns, which can in turn be used as features to build classifier? How to measure the goodness of a sampling method and what is the target distribution? Those questions shall be formally answered in Chapter 3.

Besides, in the past few years, graph classification problem has attracted lots of attention and some typical work includes graph kernel methods and subgraph mining based classification methods. However, as we shall see in Chapter 4, although many graph classification algorithms were proposed, efficiency and scalability still remain as a challenge. In the second part of this thesis, our main concern is on how to do scalable and effective graph classification using mined interesting graph patterns.

In this chapter, we will briefly review the related fundamental concepts in frequent pattern mining as well as its use in pattern-based classification. The research challenges for each topic are also highlighted. At the end of this chapter, we list our main contributions.

1.2 Frequent Pattern Mining (FPM)

FPM is one of the fundamental and active research problems in data mining. Its purpose is to find interesting patterns from the data, which could later be used in classification, clustering, prediction and related tasks. Normally, frequent patterns involve itemsets, sequences, trees and graphs. The notion of frequent itemset mining was proposed by [3], which uses Apriori level-wise property [4] for mining. From then on, the frequent pattern mining problems became very popular and many efficient and effective algorithms were proposed. Various frequent pattern mining techniques are now widely used in many disciplines like Biology and Chemo-informatics, Business Anaysis, Weather Prediction and Social Networking, in which the potential problems are large scale and the traditional analysis methods are incapable of dealing with them with high efficiency and effectiveness.

In this section, we will briefly review several fundamental concepts in itemset mining. Note that all the concepts mentioned can be easily extended to other abstract data types, e.g., trees and graphs.

1.2.1 Preliminaries

1.2.1.1 Database

Let $\mathcal{Z} = \{z_1, z_2, \dots, z_m\}$ be a binary-valued attributes set (or we call them *items*) and let $\mathcal{T} = \{t_1, t_2, \dots, t_n\}$ be a set of transactions identifiers (or we call them *tids*). We call a set $Z \subseteq \mathcal{Z}$ an *itemset*. If $|Z| = k$, i.e., the size or the cardinality of Z is k , we call Z a k -itemset. Also, a dataset \mathcal{D} is a binary relation $\mathcal{D} \subseteq \mathcal{Z} \times \mathcal{T}$. \mathcal{D} can also be considered as a set of tuples, i.e., in the form of $(t, t.X)$ where $t \in \mathcal{T}$ and $t.X \subseteq \mathcal{Z}$.

Given a dataset \mathcal{D} , we call \mathcal{D}^T the *vertical* or transposed dataset comprising tuples of the form $(z, z.Y)$ where $z \in \mathcal{Z}$ and $z.Y \subseteq \mathcal{T}$. Table 1.1 shows an example. The dataset \mathcal{D} has six transactions $\mathcal{T} = \{1, 2, 3, 4, 5, 6\}$ and five items $\mathcal{Z} = \{A, B, C, D, E\}$. For example, the tuple $(2, BCE) \in \mathcal{D}$ denotes the fact that tid 2 has three items B, C, E , whereas the tuple $(D, 1356) \in \mathcal{D}^T$ denotes the fact that item D is contained in transactions 1, 3, 5, 6. For convenience, we write subsets without commas. Thus $\{B, C, E\}$ is written as BCE, and so on.

Table 1.1: Dataset \mathcal{D} (a) and its transpose \mathcal{D}^T (b)

| tid | set of items | item | tidset |
|-----|--------------|------|--------|
| 1 | ABDE | A | 1345 |
| 2 | BCE | B | 12356 |
| 3 | ABDE | C | 2456 |
| 4 | ABCE | D | 1356 |
| 5 | ABCDE | E | 12345 |
| 6 | BCD | | |
| (a) | | (b) | |

Tidset & Support : Let 2^X denote the power set of X . We define the function $Z : 2^{\mathcal{T}} \rightarrow 2^{\mathcal{Z}}$:

$$Z(X) = \{z \mid \forall t \in X, t \text{ contains } z\} \quad (1.1)$$

where $X \subseteq \mathcal{T}$. In other words, $Z(X)$ is the set of items that are contained in X . We also define the function $T : 2^{\mathcal{Z}} \rightarrow 2^{\mathcal{T}}$:

$$T(Y) = \{t \mid t \in \mathcal{T}, Z(t) \supseteq Y\} \quad (1.2)$$

where $Y \subseteq \mathcal{Z}$. In other words, $T(Y)$ is the set of transactions which all contain the itemset Y . For example, $Z(34) = ABE$ and $T(BCE) = 245$ in Table 1.1. We also call a set $X \subseteq \mathcal{T}$ a *tidset*. Note that the tidset of an itemset Y is $T(Y)$. The *support* of Y in \mathcal{D} , denoted $sup(Y)$, is defined as the number of transactions which contain Y . Note that $sup(Y) = |T(Y)|$. For instance, $sup(BCE) = |245| = 3$. We say that Y is *frequent* if $sup(Y) \geq \sigma_{min}$, when given a user-specified minimum support threshold σ_{min} .

1.2.1.2 Pattern Partial Order Graph

In order theory, a partially ordered set is defined as the set that conforms to the partial order. In terms of partial order, suppose \mathcal{E} is a non-empty set with a relation R . If the relation R satisfies three properties: *reflexive*, *anti-symmetric* and *transitive*, we say that R is a partial order relation on \mathcal{E} , i.e. [5]:

- a) For $\forall e \in \mathcal{E}$, $(e, e) \in R$ (reflexive);
- b) If $(e, f) \in R$ and $(f, e) \in R$, then $e = f$ (anti-symmetric);
- c) If $(e, f) \in R$, $(f, g) \in R$, then $(e, g) \in R$ (transitive).

For example, the relation \leq over the real number set \mathbb{R} is a partially order relation and \mathbb{R} is a partial order set. Similarly, the frequent patterns that are sorted in accordance with the subset relation \subseteq form a *Pattern Partial Order Graph*. That is, in the graph, each node corresponds to a (frequent) pattern and two nodes that have an immediate subset-superset or parent-child relationship are connected by an edge. Put another way, each edge in the graph represents a possible extension from one (frequent) pattern to a larger (frequent) pattern by size one. The root node in the graph is usually the empty pattern, denoted as \emptyset or $\{\}$. The negative border of the pattern partial order graph is formed by the patterns which are infrequent, but all of their subsets are frequent. The immediate subsets of each pattern in negative border form the positive border. We can collect the set of all frequent patterns by traversing the partial order graph via breath-first or depth-first search methods, with the help of Apriori property (i.e., any subset of a frequent itemset must be frequent, and any superset of an itemset that is infrequent cannot be frequent) [4].

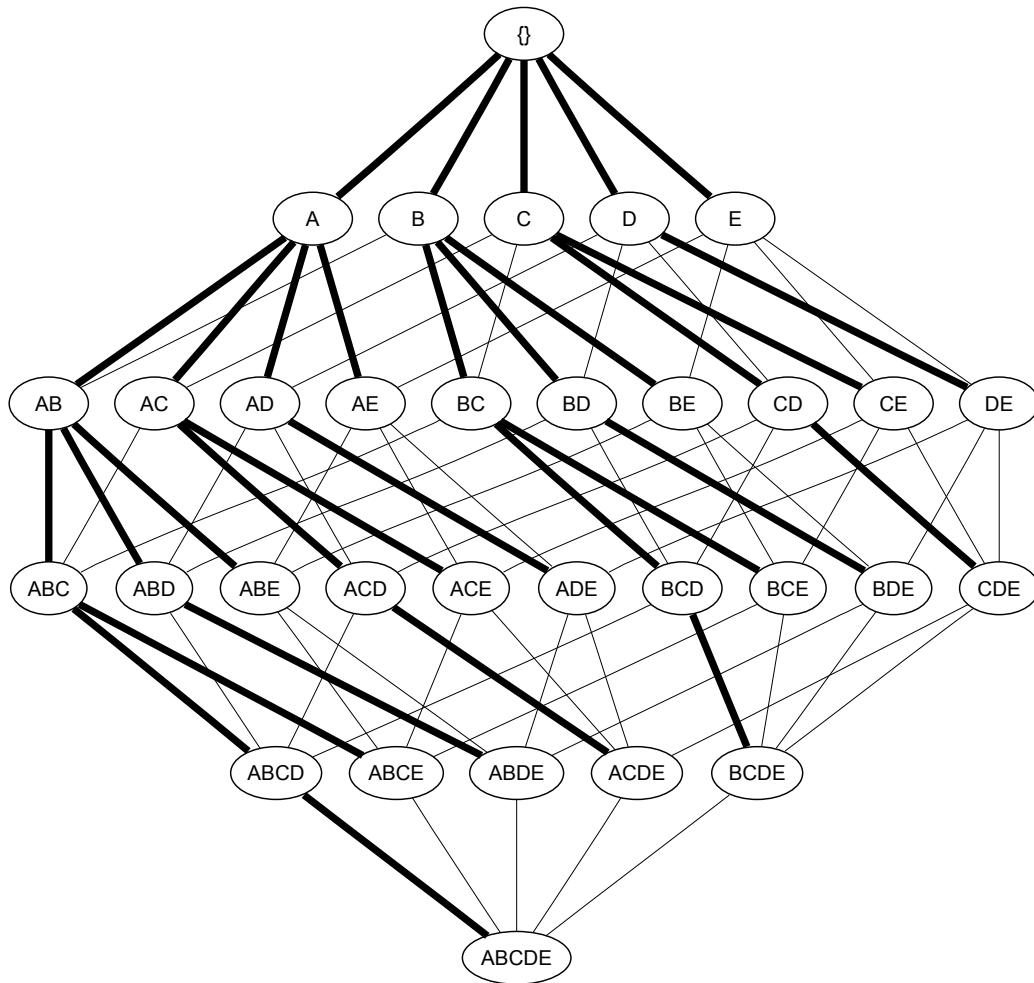


Figure 1.1: Itemset Partial Order Graph [1]

Figure 1.1 (source [1]) shows an example for itemset enumeration partial order graph [1]. It shows the generation of all the subsets of \mathcal{Z} in Table 1.1. The two nodes Z_1, Z_2 in the graph are connected iff $Z_1 \subset Z_2$ and $|Z_1| + 1 = |Z_2|$. For example, the itemsets ABC and $ABCD$ are connected since $ABC \subset ABCD$ and $|ABC| = 3$, $|ABCD| = 4$. The bold edges in Figure 1.1 represents a prefix-based enumeration approach.

Figure 1.2 (source [2]) gives another example showing the partial order structure containing closed and maximal itemsets (right) with frequent itemsets (left) for Table 1.1 [2]. In the left figure, the itemsets in each enclosed solid curve have the same support. The itemset in each curve that is not a subset of any other frequent itemsets is defined as *maximal* itemset, i.e., $ABDE$ and BCE . An itemset is defined

as *closed* if its superset does not have the same support. Note that the maximal itemsets ABDE and BCE make up the positive border of the graph. Also note that each maximal itemset is also a closed itemset, but the reverse is not true. The notion of partial order graph for itemsets can be easily extended to other frequent patterns, e.g., subgraphs.

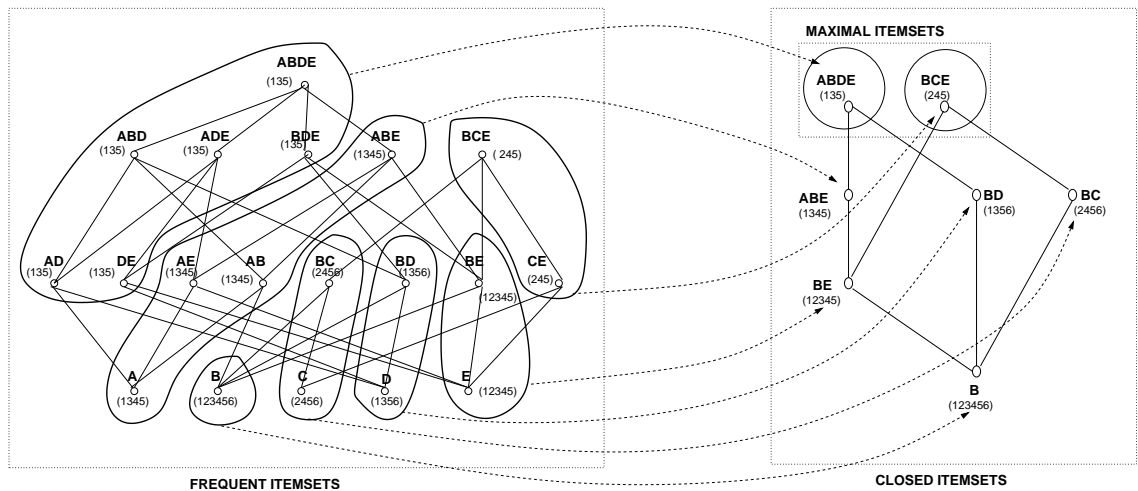


Figure 1.2: Closed and Maximal Itemsets in Table 1.1 [2]

1.2.2 Applications

FPM techniques have following typical applications:

- a) Market Analysis: By finding frequent transaction itemsets from customer data, business analysts can effectively divide the potential customers into several target groups. For each group, business people can specifically design a marketing strategy for profit based on the interested population's size, interests and needs.
- b) Science: FPM techniques can be used with other science disciplines, e.g., Chemistry and Biology. Usually the natural structures of proteins, cells and chemical compounds are complex. Efficiently indexing and finding the similarities among the proteins, chemical compounds and cells becomes an important research task. Because of the large potential database volume, simply applying a sequential search in the input database is very expensive or even infeasible [6].

- c) **Social Networking:** Graph pattern mining as well as network analytics is an active and critical FPM topic. The main purpose of graph pattern mining is to extract a set of interesting, e.g. discriminatory, subgraphs from a single large graph or from multiple graphs in a database. Graph examples include several social networks, e.g., Google+, Facebook, Twitter and Sina Blogger.
- d) **Decision Making and Planning:** Mining interesting patterns from stocks or inventory is crucial for financial decision making. Also, FPM mining techniques applied on annual census data could help government find out the details for the contemporary population, e.g. age, marriage, health and jobs. Then, the government can use population dynamics for better public facilities planning and maintenance.
- e) **Speech and language Recognition:** A typical application is that given a phoneme sequence, or a sequence of words and delimiters, try to predict next symbol of a sequence. Or given several utterances of a set of words from a person, try to classify a new utterance [7]. Or try to understand the potential information implied by the speech sequence directly.

1.2.3 Algorithms

1.2.3.1 Frequent Pattern Mining

We first briefly review the existing algorithms for FPM. For a comprehensive survey, please refer to [6].

Itemset Mining: The main purpose for itemset mining is to discover the items that occur together. One of the main applications for itemset mining is to analyze supermarket basket transaction data to investigate the customers' behavior for business. The notion of frequent itemset mining was proposed by [3], which uses Apriori level-wise property for mining: if an itemset is frequent, then all its subsets should be frequent; if an itemset is not frequent, thus all its supersets should not be frequent. From then on, frequent itemset mining became popular, and many algorithms [8, 9, 10, 11, 12, 13] have been proposed to solve the problem or similar problems more efficiently and effectively.

Sequence Mining: Sequence mining techniques aim to find frequent sequential patterns across time or positions. In [14], the authors introduce the notion of frequent sequential mining. Since then many other algorithms have been proposed [15, 16, 17, 18, 19, 20]. There are also several works [21, 19, 22] taking certain constraints into consideration, e.g., regular expressions, maximum/minimum gaps and so forth.

Graph Mining: Graph data is becoming increasingly important and ubiquitous. Examples include social networks like Twitter and Google+. One of the main purposes in graph mining is to find frequent common subgraphs. Early works in graph mining include [23, 24]. More recently, lots of algorithms have been proposed to boost the mining efficiency, including [25, 26, 27, 28, 29].

1.2.3.2 Frequent Pattern Summarization

One of main problems in FPM is that the search space for possible frequent patterns is exponentially large. Some techniques/algorithms have been proposed to summarize the frequent patterns qualitatively in order to save the storage space and computational time. In the context of frequent itemset mining, [30] and [31] proposed maximal itemset mining methods. In [32], the authors introduced closure operator and in [33], the authors introduced the concept of minimal generators. Several works (see [34]) proposed algorithms for mining closed itemsets, and some works proposed algorithms for mining minimal generators [35, 33]. CHARM-L [35] first mines all closed itemsets and then finds minimal generators based on the notion of differential lower shadow. In [36], the authors introduces the concept of succinct system of minimal generators, which minimally represents the minimal generators of all concepts. In [37], the authors formulated minimal and closed monotone DNF formulas as extensions of maximal and closed itemsets and the authors then designed a method to extract closed monotone DNF formulas. Blossom [38] is a framework for mining (frequent) boolean expressions. It integrates four categories: pure disjunctions (OR-clauses), pure conjunctions (AND-clauses), conjunction of disjunctions (CNF; conjunctive normal form), and disjunction of conjunctions (DNF; disjunctive normal form). Blossom is the first algorithm to mine minimal DNF generators.

The method has two steps: firstly it mines all minimal AND-clauses from the original dataset and secondly it mines all minimal OR-clauses from the new generated dataset by treating each minimal AND-clause as a new item with the corresponding new tidset. However, all itemset-summarization mining algorithms mentioned above suffer scalability problem as the size of dataset gets larger. For example, in one of our experiments for comparison with minDNF sampling algorithm (See Chapter 3), Blossom (written in C++) could not finish within 24 hours on a quad-core Intel i7 3.5GHz CPU machine, mining complete minimal DNF generators on Connect dataset (67557 transactions, 129 items) even with a minsup=80%.

In the context of frequent sequence mining, several algorithms [39, 40] have been proposed for mining closed sequences. In the context of graph mining, the authors in [41, 42, 43] propose closed graph mining algorithms.

1.3 Graph Theory

Here we review concepts related to graphs, that will be useful when we discuss the graph features and classification methods.

1.3.1 Preliminaries

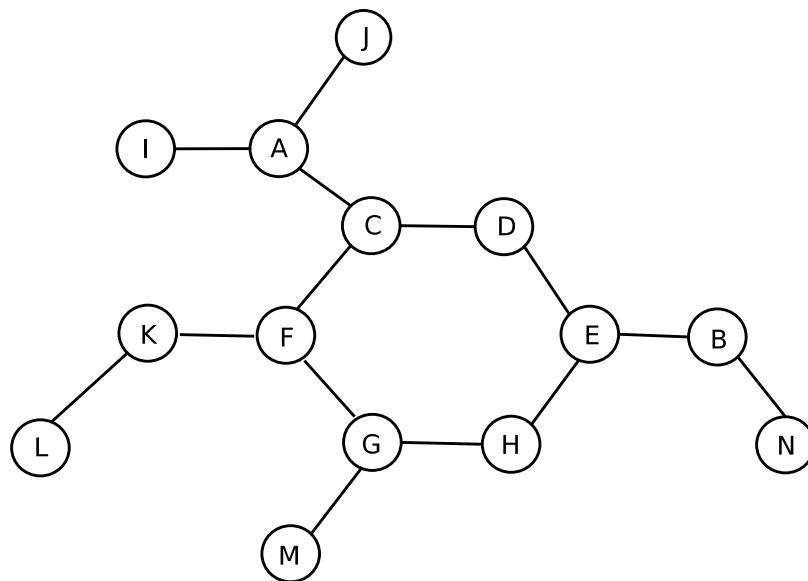


Figure 1.3: A graph example G_e

Graph: A graph $G = (V, E)$ is made of nodes V and edges E . We use n to denote the cardinality of V , i.e., the number of nodes. And we use m to denote the cardinality of E , i.e., the number of edges. Note that a graph can be *directed* or *undirected*. In a directed graph, each edge is directed, that is, each edge has an *origin* and a *destination* vertex. We use an ordered pair $(v_i, v_j) \in E$ to represent an edge with origin v_i and destination v_j . Note that in an undirected graph, there is no order between v_i and v_j . In other words, if $(v_i, v_j) \in E$, then $(v_j, v_i) \in E$ as well. In Figure 1.3 for example, there are $n = 14$ nodes and $m = 14$ edges in the undirected graph G_e .

Weighted Graphs: In certain applications it is useful to assign numerical values to the edges or vertices in a graph, which we call edge weights and vertex weights. In this thesis, we only consider edge weights in the graph. Usually we use a mapping function $\omega : E \rightarrow \mathcal{R}$ to denote that every edge $e \in E$ is assigned a weight $\omega(e) \in \mathcal{R}$. Depending on the specific applications, edge weights can be used to describe various properties such as distance, similarity, capacity etc.

Degrees: The degree of a vertex (node) $d(v)$, $v \in V$ in an undirected (and unweighted) graph $G = (V, E)$, is the number of edges (cardinality) that are incident to v . In a weighted graph, instead of taking the cardinality, all those concepts are generalized by summing over the edges weights. An undirected graph is called *regular graph* when all its vertex degrees are equal, and *k-regular* when that degree value is k . Take our running example Figure 1.3, for vertex A , $d(A) = 3$; for vertex D , $d(D) = 2$ in G_e . Note that G_e is not a regular graph.

Paths, Walks and Cycles: A *walk* of length l is defined as a sequence of vertices and edges. That is, $v_0, e_1, v_1, e_2, v_2, \dots, v_{l-1}, e_l, v_l$ such that $e_j = (v_{j-1}, v_j) \in E$ for $1 \leq j \leq l$. We say a walk is a *path*, if for any $i \neq j$ we have $e_i \neq e_j$. We define a path as a *simple path*, if for any $i \neq j$ we have $v_i \neq v_j$. We say that a path is a *cycle*, if $v_0 = v_l$. A cycle is called a *simple cycle*, if $v_i \neq v_j$ for $0 \leq i < j \leq l$. Take our running example G_e in Figure 1.3 again, there are two paths from vertex A to vertex B , i.e., $\{A, C, D, E, B\}$ and $\{A, C, F, G, H, E, B\}$. There is also a cycle

consists of vertices $\{C, D, E, H, G, F\}$.

Distances and Shortest Paths: For a path p in a graph $G = (V, E)$, the distance of the path p , denoted as $dist(p)$, is the edge weights sum on p . For two vertices $u, v \in V$ and if a path p starts from u and ends in v , we say p is the *shortest path* among all other possible paths from u to v , if its $dist(p(u, v))$ is the smallest. If the edge weights are non-negative, we could use Dijkstra's algorithm [44] to compute the single-source shortest paths problem (SSSP) in time $O(m + n \log n)$. If the edge weights contain negative values, we could use Bellman-Ford algorithm [44] to solve the problem, if there is no negative cycle in the graph. If all edge weights in the graph are unit values, simply a BFS could solve the problem in $O(m + n)$. In Figure 1.3, the shortest path length from vertex A to vertex B is 4 in G_e .

Subgraphs: Suppose that there are two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. If we have $V_1 \subseteq V_2$ and $E_1 \subseteq E_2$, We say G_1 is a subgraph of G_2 . We say G_1 is a *vertex-induced* subgraph of G_2 , if and only if E_1 contains all edges $e \in E_2$ that connect vertices in V_1 . We use $G_2(V_1)$ to denote the vertex-induced subgraph. Similarly, we say G_1 is an *edge-induced* subgraph of G_2 , if and only if V_1 is the set of all vertices in V_2 that are edge end points of E_1 . We use $G_2(E_1)$ to denote the edge-induced subgraph.

1.3.2 Algebraic Graph Theory

Isomorphism: Suppose we have two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. We say G_1 and G_2 are *isomorphic* if there exists a bijection mapping $\phi : V_1 \rightarrow V_2$:

$$\forall u, v \in V : (u, v) \in E_1 \iff (\phi(u), \phi(v)) \in E_2 \quad (1.3)$$

We denote it by $G_1 \simeq G_2$. Such a bijection is called *isomorphism*. Often we consider two graphs are the same if they are *isomorphic*.

Adjacency Matrix: An undirected or directed graph can be represented as an adjacency matrix $A(G)$ as follows:

$$[A]_{i,j} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases} \quad (1.4)$$

If G is undirected graph, then the adjacency matrix $A(G)$ is symmetric, i.e., $A^T(G) = A(G)$, where A^T denotes the transpose of A . For a weighted graph, the entries $[A]_{i,j}$ can be some numerical values rather than 0, 1.

Laplacian Matrix: The *Laplacian matrix* $L(G)$ is an $n \times n$ matrix. More formally, $L(G) = D(G) - A(G)$, where $D(G)$ is an $n \times n$ diagonal matrix, defined by setting its i -th diagonal entry equal to the corresponding degree of i -th vertex v_i . $A(G)$ is the adjacency matrix for G . The *normalized Laplacian matrix* is an $n \times n$ matrix define by $NL(G) = D(G)^{-1/2}L(G)D(G)^{-1/2}$.

1.4 Challenges

Although various algorithms were proposed for frequent pattern mining (see Section 1.2.3), efficiency still remains as a big challenge. In this thesis, we tackle the challenging problem of mining the simplest Boolean patterns from categorical datasets. One of the difficulties is that the pattern search space for the Boolean patterns is not available immediately. Hence, instead of complete enumeration, which is typically infeasible for this class of patterns, we develop effective sampling methods to extract a representative subset of the minimal Boolean patterns (in disjunctive normal form – DNF). Another difficulty comes from the sampling quality. Since we do not know the size of the potential search space, it is a challenge to design appropriate local MCMC transition probability to bound the amount of non-uniformity in the sampling. The third challenge is to find several useful properties to help prune the sampling space for performance boosting.

In the context of graph classification, the main challenge for classifying a set of graphs is how to convert the discrete graph objects into numeric features or similarities for effective classification. Previously graph kernel methods have attracted a lot of attention due to their ability to represent the graph data as a $N \times N$ symmetric, positive semi-definite *kernel matrix* $\mathbf{K} = \{\kappa(G_i, G_j)\}_{i,j=1}^N$ that

records the pair-wise similarities between graphs in \mathcal{D} . Conceptually, the kernel function $\kappa(G_i, G_j)$ represents an inner-product between the vectors corresponding to the two graphs G_i and G_j in some N -dimensional *feature space*; see [45] for more details on kernel methods. Once the kernel matrix has been constructed, it is possible to classify the graphs with a Support Vector Machine (SVM) [46], using the supplied kernel matrix \mathbf{K} . There has been a lot of research activity in trying to develop more effective and efficient graph kernel functions κ . These methods can broadly be classified into methods based on random walks [47, 48], shortest paths [49], cycles [50] subtrees [51, 52, 53], and subgraphs [54, 55, 56]. Despite the research above, it is fair to say that efficient and effective graph classification still remains a challenge, especially for large graphs. Thus, our main concern is on how to design both scalable and effective graph classification algorithm and how to find interesting graph patterns for classification.

1.5 Contributions

Our main contributions are two folds. For itemset mining and summarization, we present the first approach [57] to sample the simplest Boolean patterns, namely the minimal DNF expressions. We propose a novel weighting scheme to compute the transition probability matrix for the Markov chain Monte Carlo sampling algorithm, which bounds the amount of nonuniformity in the sampling. Our work in sampling Boolean patterns makes a number of novel contributions:

- We propose the first approach to generate a near-uniform sample of the minimal Boolean expressions. Our method, based on Markov Chain Monte Carlo sampling, yields a succinct subset of the simplest frequent Boolean patterns.
- We propose a novel theoretical characterization of the minimal DNF expressions, which allows us to prune the MCMC search space effectively. When combined with other optimization techniques, our method is also practically effective. For instance, we are able to sample interesting “support-less” patterns, i.e., where minimum frequency threshold is set to one.
- We perform an extensive set of experiments to demonstrate the effectiveness

of our method. In particular, we classify a variety of datasets from the UCI Machine Learning Repository [58], and show that minimal DNF patterns make very effective features for classification; much more so than purely conjunctive features. We also study the sample quality of our approach, as well as its scalability.

For graph classification, we propose an alternative approach [59] to construct a feature-vector for graph classification. Our novel contributions are as follows:

- Instead of relying on “patterns” like path, cycles, subtrees and subgraphs, we compute several global topological and label attributes from each graph $G_i \in \mathcal{D}$. The values for these attributes yield a numeric feature-vector $F_i = (f_{i1}, \dots, f_{ip})$. The set of feature vectors F_i and the corresponding class labels y_i are then used to construct an SVM classifier.
- We show that our approach is both effective and scalable compared to state-of-the-art graph kernel methods. We conduct an extensive set of experiments over several real graphs, representing chemical compounds, proteins, and cell-graph datasets.
- We demonstrate that our approach yields better or competitive accuracy in a fraction of the time taken by other kernels. Our method is particularly effective in classifying large unlabeled graphs, since it is able to effectively capture the structural differences among the classes.

CHAPTER 2

RELATED WORK

The two most prominent research areas in Frequent Pattern Mining (FPM) are *Itemset Mining* and *Graph Mining*. The obtained interesting patterns can be used to capture regions of high contrast among classes for construction of effective classifiers. However, mining such patterns for effective classification is challenging, particularly when the datasets are dense or structures are complex (e.g., graphs). In this chapter we shall briefly review several existing representative frequent pattern mining algorithms that are closely related to our research.

2.1 Frequent Pattern Summarization

The notion of frequent itemset mining was proposed by [3], which uses Apriori level-wise property for mining: if an itemset is frequent, then all its subsets should be frequent; if an itemset is not frequent, then all its supersets should not be frequent. From then on, frequent itemset mining became an active and fundamental research area, and many algorithms [8, 9, 10, 11, 12, 13] have been proposed to solve the problem or similar problems more efficiently and effectively.

The notion of minimal AND-clauses (called minimal itemset generators) was proposed in [33]. Minimal AND-clauses have attracted a lot of interest since it gives a succinct lossless representation of the set of the frequent itemsets. An important property of a minimal AND-clauses is that any subset of it must also be minimal. Methods that can mine minimal AND expressions include [36] (that focuses on finding the succinct minimal generators), CHARM-L [35] and Blossom [38]. CHARM-L [35] firstly mines all closed itemsets and then finds minimal generators based on the notion of differential lower shadow. [36] introduces the concept of succinct system of minimal generators, which minimally represents the minimal generators of all concepts. The algorithm of mining minimal monotone DNFs was proposed in

Portions of this chapter previously appeared as: G. Li, M. Semerci, B. Yener and M. Zaki, “Effective Graph Classification based on Topological and Label Attributes,” *Statistical Analysis and Data Mining*, vol. 5, no. 1, pp. 265–283, August 2012.

[37]. It formulates minimal and closed monotone DNF formulas as extensions of maximal and closed itemsets and the authors then proposed an algorithm to extract all closed monotone DNF formulas. Blossom [38] proposed a complete framework to extract the minimal DNF and pure AND-clauses, as well as the minimal CNF (conjunctive normal form – AND of OR-clauses) and pure OR-clauses. Blossom uses a two-step process to mine the minimal DNFs. It first mines all minimal AND-clauses, treats them as new items, as then extracts minimal OR-clauses over these composite AND-items. Disjunctive association rules have also been considered in [60]; they first mine all frequent AND-clauses, and then greedily select good OR combinations. The notion of disjunctive emerging patterns (EPs) for classification was proposed in [61]. Disjunctive EPs are Boolean expressions in CNF form, such that their support is high for the positive class and low for the negative class. However, they consider restricted CNF expressions that must contain a clause for each attribute. As we shall see in Chapter 3, we mine general DNF expressions, without any constraints.

2.2 Frequent Pattern Sampling

It is known that complete FPM is infeasible for all but very high support values for large datasets, since the search space for FP grows exponentially. Thus, the research focus in recent years has shifted to sampling based approaches.

Sampling is an old topic and a lot of research has been made in the past. The main purpose of sampling in data mining community is to select a subset of patterns from the original transaction-set, which could be used for estimating the statistics of the transaction-set, or for clustering, classification, regression and so forth. The main benefit of sampling is to improve the efficiency and effectiveness through a small number of representatives that fits in the memory for estimation. In this section, we will discuss several sampling methods. Note that the divisions are not rigid and there may exist overlaps between these categories.

2.2.1 Itemset Sampling

In the research area of itemset mining, there are several works focused on sampling directly from the input transaction-set.

The first work comes from [62]. The main idea by Toivonen in his paper [62] is to first randomly draw a subset of transactions that may probably generate frequent itemsets or association rules that hold in the whole database. Then the algorithm uses the results (i.e., mined frequent itemsets and negative border from the chosen transactions using a relatively lower support threshold) to verify the rest of the database and compute the exact support of the frequent itemsets. Note that a negative border is defined as the set of itemsets which are not frequent themselves, but all their immediate subsets are frequent. The sampling algorithm in [62] avoids several passes over the database to be analyzed and thus avoids the potential I/O bottleneck problem for large datasets.

In [63], Chen et al. used a two-phase sampling method to discover association rules in large databases. In the first phase, the proposed method quickly computes an estimate of each single item from an initial sample set of transactions. In the second phase, the authors used the estimated support to prune out “unuseful” transactions and choose “useful” transactions. The chosen transactions could form a final smaller sample set that reflects similar statistical characteristics as the original transaction set [63]. In [64], the authors showed that sampling can with high confidence accurately represent the data patterns from the database. The authors showed that, for small databases, a fixed portion of sampling (10% – 25%) of the original databases by a heuristic could obtain 95% of the frequent itemsets [64]. In [65], Chakaravarthy et al. presented some theoretical analysis for sampling association rules. The authors showed in [65] that both the frequent itemset mining and association rules can be solved independently of the number of transactions as well as the number of items with any required accuracy. In [66], the authors proposed to use a randomized approximation method for counting the number of frequent itemsets.

More recently, in [67], the authors presented a direct sampling approach for mining itemsets. Their sampling methods are non-process-simulating, and they claimed that the proposed methods yield nearly optimal time complexity as well as easy-controlled distribution of the produced patterns. More specifically, their four sampling algorithms [67] produce k patterns with distribution proportional to

frequency, area (i.e., frequency times size), squared frequency and discriminatory (i.e., frequency in positive data portion times frequency in negative data portion), respectively.

- a) Frequency-based sampling: the algorithm first starts to sample a record from the transactions with a probability proportional to the size of the corresponding power set. Then the sampler selects randomly an entry from the power set of the selected record. The selected entries by the above two steps follow the desired distribution.
- b) Area-based sampling: the first step is identical to frequency-based sampling, and in second step, instead of randomly selecting an entry from the power set, the algorithm draws a subset of the selected record with probability proportional to its size.
- c) Squared-frequency-based sampling: the first step is to draw two random transactions T_1, T_2 , with probability proportional to the size of power set of $2^{|T_1 \cap T_2|}$. Secondly, a set is drawn uniformly from the power set of $T_1 \cap T_2$.
- d) Discriminatory-based sampling: the first step is to sample T_+, T_- pair with probability proportional to the weight $(2^{T_+ \setminus T_-} - 1)2^{T_+ \cap T_-}$ ($+, -$ are class labels). Then, return the sample (R, R') with R uniformly drawn from the power set of $T_+ \setminus T_-$ (excluding ϕ) and R' uniformly drawn from the power set $T_+ \cap T_-$.

The proposed algorithms have time complexity $O(\|\mathcal{D}\| + kn)$ for frequency and area based sampling, and $O(\|\mathcal{D}\|^2 + kn)$ for squared frequency and discriminatory sampling, where $\|\mathcal{D}\|$ denotes the size of the given dataset (i.e., the sum of all transaction sizes), n denotes the number of items and k denotes the desired number of interesting patterns [67]. That is, each interesting pattern is generated with time linear in the number of items, after a linear or quadratic preprocessing phase. Although the time is almost optimal, there is no guarantee of finding all patterns satisfying interestingness threshold.

In the context of itemset summarization sampling, [68] gave the first randomized algorithm for generating frequent maximal sets via hypergraph traversal. In [69], a Metropolis-Hastings algorithm for sampling closed itemsets was given. The random walk is performed based on closure operators.

2.2.2 Graph Sampling

There exist many works focusing on sampling a subset of nodes and edges from a single graph [70, 71, 72, 73, 74, 75, 76]. The problem arises, for example, if people want to analyze online social networks, such as Twitter and Facebook, with millions of nodes and edges. However, for such real world huge graphs, it is often impossible to put the whole graph into the main memory or even so the cost of enumeration to compute basic graph properties is prohibitive. So people tend to use sampling methods to tackle the problems. The sampling methods range from node sampling [70] to edge sampling [70, 71], non-myopic sampling [70] to myopic sampling [72, 73, 74, 75, 76]. Note that the divisions are not rigid and there may exist overlaps between these categories.

2.2.3 Input Space versus Output Space Sampling

Compared to input space sampling that has a long history, output space sampling for local patterns is a relatively new research area and has received much attention in recent years. Generally speaking, input space sampling aims at sampling a small set of transactions from the input dataset, in order to find interesting prospective frequent patterns. For example, the algorithms [62, 63, 65] belong to this approach. The notion of output space [77], is the set of candidate frequent patterns. Sampling in output space means sampling interesting frequent patterns directly from the entire set of candidates. Note that the output space sampling is a much harder problem, since the number and the qualified patterns in output space is not available immediately [77].

As [77] pointed out, output space sampling has three important advantages: (1) high scalability, (2) the algorithm is independent of interestingness criteria and the objective function, and (3) easy to parallelize. Generally, most sampling work for output space enforce a markov random walk on the partial order graph of the

interesting patterns and the walker starts to collect qualified samples when it converges to the stationary distribution. Thus, most of output sampling algorithms are myopic since in each walk the sampler only sees the local neighborhood.

In [78, 79], the authors proposed a randomized sampling method to generate a small representative set of frequent maximal graph patterns. In more detail, firstly, their algorithm randomly generates a set of frequent maximal subgraphs, and secondly a smaller representative set from the maximal set is chosen. The method captures graph representative set by considering the pattern distances in the search space. However, their method did not provide any sampling guarantee. One of the first methods to uniformly sample maximal graph patterns via MCMC was presented in [80]. Note that, in a general random walk sampling strategy, each time the sampler chooses uniformly from one of neighbors from the current node. To be more specific, suppose the current node is u , then the transition probability is:

$$P(u, v) = \begin{cases} \frac{1}{d_u} & \text{if } v \in \text{adj}(u) \\ 0 & \text{otherwise} \end{cases} \quad (2.1)$$

where d_u is the degree of u , and $\text{adj}(u)$ denotes the neighborhood of u . Random walk sampling is simple, since the sampler collects samples during walking. But in [80], the authors proved that the stationary distribution of ergodic random walk for a vertex with transition probability defined in Equation 3.6 is directly proportional to the sum of edge weights incident to that vertex. In other words, the random walk sampling favors nodes with higher weights. If we treat each edge with weight 1, then random walk sampling favors nodes with higher degrees. To mitigate the effect of bias towards nodes with higher degree, [80] proposed to use the following edge weight definition:

$$w(u, v) = \begin{cases} 1 & \text{if } u \text{ and } v \text{ are both non-maximal} \\ \frac{c}{d_v} & \text{if } v \text{ is maximal and } u \text{ is non-maximal} \\ \frac{c}{d_u} & \text{if } u \text{ is maximal and } v \text{ is non-maximal} \\ 0 & \text{if } u \text{ and } v \text{ are both non-maximals} \end{cases} \quad (2.2)$$

and the transition matrix P is then defined as:

$$P(u, v) = \begin{cases} \frac{w(u, v)}{\sum_{x \in \text{adj}(u)} w(u, x)} & \text{if } v \in \text{adj}(u) \\ 0 & \text{otherwise} \end{cases} \quad (2.3)$$

In [80], the authors then proved that the sampled maximal graphs are generated with equal probability. In [77], the authors introduced a generic sampling framework to sample the output space of frequent subgraphs, which uses Metropolis-Hastings (MH) algorithm that is based on proposal distribution and sample rejection. To be more specific, the transition matrix P is defined as:

$$P(u, v) = \begin{cases} \frac{1}{\max(d_u, d_v)} & \text{if } v \in \text{adj}(u) \\ 1 - \sum_{x \in \text{adj}(u)} P(u, x) & \text{if } u = v \\ 0 & \text{otherwise} \end{cases} \quad (2.4)$$

The transition matrix is an adaptation of the classical MH algorithm [77]. The proposal distribution is defined by a general random walk, i.e., each neighbor is chosen with equal probability ($P(u, v) = 1/d_u$, $v \in \text{adj}(u)$), and the acceptance rate is defined as:

$$\text{accept-rate}(u, v) = \min \left\{ \frac{\pi(u)d_u}{\pi(v)d_v}, 1 \right\} = \min \left\{ \frac{d_u}{d_v}, 1 \right\} \quad (2.5)$$

Then, the authors used a similar idea for Support Proportional Sampling (i.e., sample a pattern in proportion to the support value) and Discriminatory Subgraph Sampling (used as features for graph classification) by selecting appropriate proposal distributions in [77].

Sampling Bias Correction: Note that in [80], the authors proved that the stationary distribution of ergodic random walk for a vertex with transition probability defined in Equation 3.6 is directly proportional to the sum of edge weights incident to that vertex. In order to correct the degree bias caused by random walk, several methods mentioned above use the modified transition matrix in MCMC to mitigate the sampling bias. There is another idea which still uses the completely

random walk strategy, but then use statistical methods to reweight the bias. One typical work, called Respondent Driven Sampling (RDS), comes from [81]. RDS forms asymptotically unbiased estimators by re-weighting of estimators. The authors [81] used Hansen-Hurwitz estimator [82]. For example, we may want to estimate the distribution of nodes' property Q . Consider a random walk that visits nodes $G = \{g_1, g_2, \dots, g_n\}$ consecutively, and the nodes in G do not have to be mutually different. These nodes can be grouped into m sets, S_1, S_2, \dots, S_m . The nodes in the same group have the same property, namely, $S_i = \{g : Q(g) = Q_i\}$. A simple case is that the nodes could be grouped by their degrees. The Hansen-Hurwitz property [82] states that, the estimator $\hat{E}(Q) = \frac{1}{n} \sum_{g \in G} \frac{Q(g)}{\pi(g)}$ is a consistent and unbiased estimator of the sum $E(Q) = \sum_{g \in V} Q(g)$, where π is the stationary distribution of the walk and V is the set of nodes in the graph. The two special cases for the estimator are: when $Q = I_{S_i}$, the indicator of a node g in the set S_i , i.e., $I_{S_i}(g) = 1$ if $g \in S_i$ and 0 otherwise. Then $\hat{E}(I_{S_i})$ estimates how many nodes will be in S_i . When $Q = 1$, the estimator $\hat{E}(1)$ estimates the total number of nodes in the graph. Therefore, the degree distribution is estimated as:

$$\hat{p}_i = \frac{\hat{E}(I_{S_i})}{\hat{E}(1)} = \frac{\sum_{g \in S_i} \frac{1}{\text{Deg}(g)}}{\sum_{g \in V} \frac{1}{\text{Deg}(g)}} \quad (2.6)$$

where $\text{Deg}(g)$ denotes the degree of node g in the graph. In [82], the authors claimed that \hat{p}_i is consistent with the true p_i , as the steps of walks increases. Note that RDS estimator can be considered as an importance sampling estimator, weighted by the stationary distribution π [82].

2.2.4 Single versus Parallel Threads

There are also several works using multiple independent parallel walks to improve convergence rate [83]. In [81], the authors propose to start multiple walks from the same node and in [84], the authors propose to start from different nodes to better utilize multiple chains. By increasing the number of parallel random walks, the number of the potential walk length is reduced and time is shortened to obtain a prespecified number of samples. Another advantage of using K multiple parallel walks is that we can reduce the possibility that using only one walk it is easy to get

trapped in a certain region. Also, the implementation is trivial: we can do implementation on multiple machines or on one machine using multithreads. However, as [81] pointed out, the shortcoming of increasing the number of random walks is that it can result in increasing the redundant number of samples close to the root node.

2.3 Pattern-based (Itemset) Classification

Several research work use the mined patterns to build classification models. The basic idea in the literature focuses mainly on building a rule-based classifier by mining high support and discriminative rules [85, 86, 87, 88]. HARMONY [85] is an association-rule based algorithm that directly finds one of the highest confidence classification rules for each training transaction. The final classification model is built from the union of these rules on the entire database. Krimp [86] employs Minimum Description Length (MDL) to select a small number of itemsets which serve as an approximation of the original database. Then a MDL-based classifier is built based on the selected small set of instances.

iCAEP [87] is based on emerging patterns (EPs) which are used to capture differences between classes in the database. iCAEP is an information based classifier which classifies an instance t by accumulating EPs that exist in t mined from the training data. Some variants include disjunctive emerging patterns (DEPs) [61, 89] mining for classification, which are usually useful in sparse data. Traditionally, simple contrast patterns consist of only conjunctions, whereas disjunctive means adding disjunctions to the contrast patterns, for example, $(a_1 \vee a_4) \wedge (b_1 \vee b_3)$. Note that each DEP should contain one or more items from the domain of *every* attribute.

2.4 Graph Classification

In the past decade a lot of algorithms for classifying graph objects were proposed.

2.4.1 Graph Kernels

Graph kernels compute the similarity based on the common patterns that the pair-wise graphs share. Specifically the kernels are designed to use random walks [47, 48, 90, 91], shortest paths [49], cyclic patterns [50], subtrees [51, 52, 92, 53] and subgraphs [54, 55, 56]. Another class of graph kernels, e.g., the diffusion kernel [93], deals with the similarity between nodes of a single graph. However, our focus in this paper is on kernels between different graphs, which we discuss in more detail below.

Random Walk Kernels: The similarity of two graphs $G_i, G_j \in \mathcal{D}$ can be quantified by counting labeled walks that are common to both of them. The random walk kernel [47], one of the first graph kernels, is based on this idea. The kernels in [48, 90] are also based on random walks over labeled graphs. Computing the pair-wise kernel values has worst case $O(n^6)$ complexity, where n denotes the number of nodes in G_i and G_j . A more efficient version of the random walk kernel was proposed in [91], reducing the complexity to $O(n^3)$ per pair of graphs. One potential problem with these kernels is that artificially high kernel values may be obtained by repeatedly visiting same nodes and edges multiple times [94]. We refer to [95] for a recent overview of random walk based graph kernels.

Shortest Path Kernels: The shortest-path graph kernel [49] first computes the shortest-path graph $S = (V_S, E_S)$ for each graph $G = (V, E) \in \mathcal{D}$. Here $V_S = V$, and a weighted edge (v_a, v_b) exists in E_S if v_a and v_b are connected by a path in G , with the edge weight representing the shortest path length between v_a and v_b (infinity if they are not reachable). Given the shortest-path graphs S_i and S_j for two input graph G_i and G_j the kernel is defined as the sum over all pairs of edges from S_i and S_j , using any suitable positive definite kernel on the edges. The all-pairs shortest-path graphs can be computed in $O(n^3)$ time, and the kernel can then be computed in $O(n^4)$ time, since S_i and S_j each have $O(n^2)$ edges. Other variants of the shortest path kernel include equal length shortest paths, k shortest paths, k shortest disjoint paths, and so on [49].

Cyclic Pattern Kernels: The cyclic pattern kernel [50] is based on counting the number of common cycles that occur in both graphs. Since there is no known polynomial time algorithm to find all the cycles in a graph, sampling and time-bounded enumeration of cycles are used to measure the similarity of the graphs.

Subtree Kernels: Subtree kernels are based on common subtrees in the graphs [51]. The main idea is to consider pairs of nodes from G_i and G_j and see if they share common tree-like neighborhoods, i.e., to count the pairs of identical subtrees of height h rooted at vertex $v_a \in G_i$ and $v_b \in G_j$. The kernel is defined as the sum over all pairs of vertices of a suitably defined vertex pair kernel. The complexity of this approach is $O(n^2h4^d)$, where d denotes the maximum degree. Another subtree kernel was proposed in [52], based on a path representation of the trees obtained via a depth-first search on the input graphs. The kernel function is computed on these paths (e.g., the ratio of the longest common path).

The recently proposed Weisfeiler-Lehman Kernel [53], is a fast subtree kernel that scales up to large, labeled graphs. It uses the Weisfeiler-Lehman isomorphism test, which uses iterative multiset-label determination, label compression, and re-labeling steps. The isomorphism test terminates after a pre-specified number of iterations h . If the sets of labels for nodes are not identical, then two graphs are considered as non-isomorphic, otherwise, they are isomorphic. The WL graph kernel counts the matching multiset labels for the two graphs G_i and G_j in each iteration of the WL isomorphism test. The WL kernel has $O(mh)$ complexity, where m is the number of edges in the graphs.

Graphlet and Subgraph Kernels: Similar graphs should have similar subgraphs. Graphlet kernels measure the similarity of two graphs by the dot product of count vectors of all possible connected subgraphs of order k (i.e., the graphlets, also called as k -minors) [54, 55]. For any k (usually set to 3, 4, or 5), there are $2^{\binom{k}{2}}$ possible graphlets of size k , but many of them are isomorphic. Usually, to avoid the dependence on the size, the count vector is normalized into a probability vector, and the graphlet kernel is re-defined as the dot product of the normalized count vectors for two graphs. Exhaustive enumeration of all graphlets has complexity $O(n^k)$.

For a graph with bounded degree d , the connected graphlets can be enumerated in $O(nd^{k-1})$ [54].

2.4.2 Frequent Subgraph Mining for Classification

Frequent subgraph mining can also be used to define a kernel between two graphs [56]. Let $\mathcal{F} = \{s_1, \dots, s_p\}$ denote the set of p frequent and discriminative subgraph patterns mined from \mathcal{D} . Each graph $G_i \in \mathcal{D}$ is then represented as a binary feature vector $\{0, 1\}^p$ where feature j is set to 1 if and only if s_j is isomorphic to a subgraph in G_i . The kernel between G_i and G_j can be defined over their binary feature vectors. CORK [56] implements this approach; it uses gSpan [29] to mine the subgraphs, and selects near-optimal features (subgraphs) from that set, that are most discriminative for classification. That is, CORK [56] measures the quality of a set of subgraphs to discriminate target classes using a specific set of features.

CHAPTER 3

SAMPLING MINIMAL FREQUENT BOOLEAN (DNF) PATTERNS

In this chapter we tackle the challenging problem of mining the simplest Boolean patterns from categorical datasets. Instead of complete enumeration, which is typically infeasible for this class of patterns, we develop effective sampling methods to extract a representative subset of the minimal Boolean patterns (in disjunctive normal form – DNF). We make both theoretical and practical contributions, which allow us to prune the search space based on provable properties. Our approach can provide a near-uniform sample of the minimal DNF patterns. We also show that the mined minimal DNF patterns are very effective when used as features for classification.

3.1 Introduction

Frequent pattern mining, long a mainstay of data mining, is moving away from complete enumeration methods to approaches that can effectively sample the pattern space for the most interesting patterns. This shifting trend is in keeping with the growing complexity of the data as well as the types of patterns sought. Whereas much research in the past has focused on itemset mining, i.e., conjunctive patterns, our focus is on the entire class of Boolean patterns in the disjunctive normal form (DNF), i.e., disjunctions over conjunctive patterns. Such Boolean patterns can help discover interesting relationships among attributes (e.g., gene expression mining [38]).

Complete enumeration of all frequent Boolean patterns is prohibitive in most real-world datasets, and thus the main issue is how to effectively sample a representative subset, which can in turn be used as features to build classification models. Furthermore, we focus on the problem of mining only the most simple Boolean

This chapter previously appeared as: G. Li and M. Zaki, “Sampling Minimal Frequent Boolean (DNF) Patterns,” In *Proceedings of the 18th SIGKDD International Conference on Knowledge Discovery and Data Mining*, Beijing, China, 2012, pp. 87–95.

patterns that completely characterize a subset of the data, i.e., the minimal DNF expressions. Our work makes number of novel contributions:

- a) We propose the first approach to generate a near-uniform sample of the minimal Boolean expressions. Our method, based on Markov Chain Monte Carlo (MCMC) sampling, yields a succinct subset of the simplest frequent Boolean patterns.
- b) We propose a novel theoretical characterization of the minimal DNF expressions, which allows us to prune the pattern search space effectively. When combined with other optimization techniques, our approach is also practically effective. For instance, we are able to sample interesting “support-less” patterns, i.e., where the minimum frequency threshold is set to one. The pruning techniques can be applied by any method (even a complete one) for mining Boolean expressions.
- c) We perform an extensive set of experiments to demonstrate the effectiveness of our method. In particular, we classify a variety of datasets from the UCI Machine Learning Repository [58], and show that minimal DNF patterns make very effective features for classification; more so than purely conjunctive features. We also study the sample quality of our approach, as well as its scalability.

3.1.1 Preliminaries

Boolean Expressions : Let AND, OR, and NOT denote the logical operators. We denote a negated item as \bar{z} (NOT z). We also call \bar{z} the complement of z . We use the symbols \wedge and \vee to denote AND and OR, respectively. For example, $A \vee B$ and $A \wedge B$ denote logical expressions A OR B, A AND B, respectively. For conciseness we also use $|$ in place of \vee and we usually omit the \wedge operator. For example, $A|BC|D$ denotes the Boolean expression A OR (B AND C) OR D.

A *literal* could be either an item z or its complement \bar{z} . An AND-clause contains only the AND operator over all its literals, e.g., BCD . Likewise, an OR-clause contains only the OR operator over all its literals, e.g., $C|E|F$. We assume

that a clause does not contain both a literal and its complement – e.g., $A \wedge \bar{A}$ leads to contradiction, and $\bar{A} \vee A$, to a tautology.

We adopt the disjunctive normal form (DNF) to represent Boolean expressions. A Boolean expression Z is said to be in DNF if it consists of OR of AND-clauses, with the NOT operator (if any) directly preceding only literals, written as:

$$Z = \bigvee_{i=1}^k Z_i = \bigvee_{i=1}^k (z_{i1} \wedge z_{i2} \wedge \cdots \wedge z_{im_i}) \quad (3.1)$$

Here each z_{ik} is a literal and each $Z_i = (z_{i1} \wedge \dots \wedge z_{im_i})$ is an AND-clause. The size or length of a Boolean expression Z is the number of literals in Z , denoted $|Z| = \sum_{i=1}^k m_i$.

Tidset and Support : Given a tuple $(t, t.X) \in \mathcal{D}$, and a literal l , the *truth value* of l in t is 1 if $l \in t.X$, and 0 otherwise. Likewise, the truth value of \bar{l} is 1 if $l \notin t.X$, and 0 otherwise. We say t *satisfies* a Boolean expression Z , if after replacing every literal in the Boolean expression with its truth value, the Boolean expression evaluates to true. The set of all satisfying transactions is called the *tidset* of Z , and is denoted as

$$T(Z) = \{t \in \mathcal{T} \mid t \text{ satisfies } Z\} \quad (3.2)$$

The number of satisfying transactions is called the *support* of Z in \mathcal{D} , denoted $sup(Z) = |T(Z)|$.

Minimal Boolean Expressions : Given DNF expressions $X = \bigvee_{i=1}^m X_i$ and $Y = \bigvee_{j=1}^n Y_j$, where X_i and Y_j are AND-clauses, we say that X is a *subset* of Y , denoted $X \subseteq Y$, iff there exists an injective (or into) mapping $\phi : X \rightarrow Y$, that maps each clause X_i to $\phi(X_i) = Y_{j_i}$, such that $X_i \subseteq Y_{j_i}$. If $X \subset Y$ and $|X| = |Y| - 1$, we say that X is a *parent* of Y , and Y is a *child* of X .

Definition 1 *A DNF expression Z is said to be minimal or minDNF (with respect to support) if there does not exist any expression $Y \subset Z$, such that $T(Y) = T(Z)$.*

A minimal AND-clause is called *minAND* for short. A minimal Boolean expression is also called a minimal generator.

A minDNF expression Z is thus the simplest DNF expression with tidset $T(Z)$. Given a user-specified minimum support threshold σ_{\min} , we say that a DNF expression Z is frequent if $\text{sup}(Z) \geq \sigma_{\min}$. However, note that the support of a minDNF expression is not monotonic, since the addition of an item to a clause causes the support to drop, whereas, the addition of an item as a new clause causes the support to increase. For example, $\text{sup}(A) = 3$, since $T(A) = 124$, but $\text{sup}(AB) = 1$ (since $T(AB) = 1$) and $\text{sup}(A|E) = 4$ (since $T(A|E) = 1234$). Thus, the support of Z 's children can be higher or lower. Further, any infrequent clause can be made frequent by adding additional clauses. For example, if $\sigma_{\min} = 2$, then C is infrequent, but $C|F$ is frequent. To prevent such ‘‘trivial’’ clauses, we also impose a minimum support threshold σ_{\min}^c on the clauses. For any DNF expression $Z = \bigvee_{i=1}^m Z_i$, we say that Z is *frequent* if $\text{sup}(Z) \geq \sigma_{\min}$, and $\text{sup}(Z_i) \geq \sigma_{\min}^c$ for all $i = 1, \dots, m$. Since a clauses' support cannot exceed the support of the whole DNF expression, we have the condition that $\sigma_{\min}^c \leq \sigma_{\min}$ (usually, we just set them equal).

Given σ_{\min} and σ_{\min}^c , the *complete minDNF mining task* is to enumerate all frequent minDNF expressions. However, given the huge search space, it is typically not feasible to mine the complete set of minDNF expressions. Instead, we will focus on sampling a representative subset.

Markov Chain Monte Carlo (MCMC) Methods : A *Markov chain* is a mathematical model for stochastic systems with discrete or continuous states controlled by transition probabilities. A Markov chain satisfies the property that the current state depends only on the previous (the most recent) state, i.e., $P(X_{t+1} = s_{t+1} | X_t = s_t, X_{t-1} = s_{t-1}, \dots, X_0 = s_0) = P(X_{t+1} = s_{t+1} | X_t = s_t)$ for all $t \in \mathcal{N}$ and $s_t \in \mathcal{S}$, where \mathcal{N} and \mathcal{S} denote the time space and state space, respectively. Let $p^t(i, j)$ denote the t -step transition probability, i.e., $p^t(i, j) = P(X_{n+t} = s_{n+t} | X_n = s_n)$. If for all states $s_i, s_j \in \mathcal{S}, i \neq j$, there $\exists t' \in \mathcal{N}$, s.t. $P^t(i, j) > 0$ is satisfied for all $t > t'$, we call the Markov chain *aperiodic*. Given two states s_i and s_j , we say s_j is *reachable* from s_i , if $\exists t$, s.t. $P^t(i, j) > 0$, and we denote it as $s_i \rightarrow s_j$. If all state

pairs are mutually reachable, we call the Markov chain *irreducible*. A Markov chain that is aperiodic and irreducible is called *ergodic*. An ergodic Markov chain has a stationary distribution $\pi = (\pi_i | s_i \in \mathcal{S})$, that satisfies three properties: (1) $\pi_i > 0$. (2) $\sum_{s_i \in \mathcal{S}} \pi_i = 1$. (3) $\pi P = \pi$, i.e., $\sum_{s_i \in \mathcal{S}} \pi_i P(i, j) = \pi_j$. The stationary distribution for an ergodic Markov chain is unique. A Markov chain is *time reversible* iff it has a stationary distribution π that satisfies the balance condition $\forall s_i, s_j \in \mathcal{S}$, $\pi_i P(i, j) = \pi_j P(j, i)$.

In the context of minDNF mining, each Markov state can be taken to be a Boolean expression, with transitions allowed, for example, only between parent and child expressions. Starting from the empty expression, we can then use Monte Carlo methods to perform random walks in the expression space to sample the minimal expressions. The main challenges include efficiency, and guaranteeing sampling quality. We address these questions below.

3.2 Minimal Boolean Expressions

In this section we prove some properties of minDNF expressions, which will allow us to design effective pruning strategies while sampling.

Lemma 1 *Any subset of a minimal AND-clause generator must also be a minimal generator.*

PROOF: Let X be a minAND expression, and let $Y \subset X$. Assume that Y is not minimal. Then there exists a minAND expression $Z \subset Y$, such that $t(Z) = t(Y)$. However, in this case, $t((X \setminus Y) \cup Z) = t(X)$, which contradicts the fact that X is minimal. Thus, Y must be a minAND expression. ■

Theorem 1 *A DNF expression $Z = \bigvee_{i=1}^n Z_i$ is minDNF iff it satisfies the following two properties:*

- a) *For any Z_i , ($i = 1, \dots, n$), we have $T(Z_i) \not\subseteq \bigcup_{j \neq i} T(Z_j)$. In other words, for any tidset of a clause, it is not a subset of the unions of tidsets over the other clauses.*

b) If we delete any item z_{ja} from a clause Z_j to yield a new clause $Z'_j = Z_j \setminus z_{ja}$, then for the resulting DNF expression $Z' = (\bigvee_{i \neq j} Z_i) \vee Z'_j$, we have $T(Z') \neq T(Z)$.

PROOF: If (a) is violated, we can simply delete Z_i without changing support, which would contradict the fact that Z is minimal. Likewise, if (b) is violated, it would contradict Z 's minimality. For the reverse direction, suppose a DNF expression $Z = \bigvee_{i=1}^n Z_i$ satisfies properties (a) and (b). We have to show that Z is a minDNF. Assume that Z is not minimal. Then there exists a minDNF $Y = \bigvee_{j=1}^m Y_j$, such that $Y \subset Z$ and $T(Y) = T(Z)$, which implies that there exists an injective mapping ϕ that maps each $Y_j \in Y$ to $\phi(Y_j) = Z_i \in Z$, such that $Y_j \subseteq Z_i$ and $T(Y_j) \supseteq T(Z_i)$. There are two cases to consider: (1) If ϕ is a bijection, then $m = n$, and there exist a clause $Y_j \in Y$, such that $Y_j \subseteq \phi(Y_j) = Z_i \in Z$. However, in this case property (b) of Z is violated, since we can delete some item from $(Z_i \setminus Y_j)$, and the resulting expression will still have the same support at Z . (2) If ϕ is not a bijection, then $m < n$, and there exists a clause $Z_k \in Z$, such that $\phi^{-1}(Z_k) \notin Y$. However, in this case property (a) of Z is violated, since $T(Y) = T(Z)$ implies that $T(Z_k) \subseteq \bigcup_{i \neq k} T(Z_i)$. Therefore, Z must be a minimal DNF expression. ■

Lemma 2 *A minDNF consists of OR of minAND expressions, i.e., if $Z = \bigvee_{i=1}^n Z_i$ is minDNF, then each Z_i must be a minimal AND-clause.*

PROOF: Assume some Z_i is not a minimal AND-clause. Then there exists a literal $l \in Z_i$, such that $T(Z_i \setminus l) = T(Z)$. In this case we can delete l from Z_i without affecting $T(Z)$, which violates property (b) in Theorem 1. ■

Lemma 3 *If $Z = \bigvee_{i=1}^n Z_i$ is minDNF, for any $Z_i, Z_j \in Z$, we have $Z_i \not\subseteq Z_j$. In other words, no clause is a subset of another clause.*

PROOF: Suppose $Z_i \subseteq Z_j$. Thus $T(Z_i) \supseteq T(Z_j)$ and property (a) in Theorem 1 is violated. ■

Please note that Theorem 1 is a sufficient condition for Lemmas 2 and 3 but not a necessary condition. As such a DNF expression that satisfies Lemmas 2 and 3, need not be a minimal generator. We use this observation to reduce the random walk state space.

Corollary 1 *Any clause-wise subset (obtained by deleting an entire clause) of a minDNF expression Z is also minDNF.*

PROOF: The proof is similar to Lemma 1. Suppose a clause-wise subset $Z_s \subset Z$ is not minDNF. Then we can replace Z_s with its equivalent minDNF, say Z'_s , in Z , without affecting the tidset of Z . This would contradict the minimality of Z . ■

For example, for the example in Table 1.1, $B|DF|E$ is a minimal DNF generator, with tidset $T(B|DF|E) = 12345$. Thus all clause-wise subsets, namely B , DF , E , $B|DF$, $B|E$, $DF|E$ are minDNF expressions.

Corollary 2 *Disallowing the empty pattern \emptyset as a valid minDNF, then a single item is always a minimal AND-clause and thus a minDNF as well.*

PROOF: It is easy to see this by Definition 1. ■

In our running example in Table 1.1, items A , B , C , D , E and F are all minDNFs.

3.3 minDNF Sampling Algorithm

The state space for the Markov chain for the minDNF mining consists of DNF expressions linked by immediate subset-superset or parent-child relationships. The DNF partial order is generated via the following four operations: a) `Add_As-Clause` (AAC): add a new clause comprising a single item into the DNF. b) `Add_To-Clause` (ATC): add an item to an existing clause. c) `Delete_The-Clause` (DTC): Delete a single item clause from the DNF. d) `Delete_From-Clause` (DFC): Delete an item from a clause (with at least two items) in the DNF. The added or deleted item can be either an item or its complement.

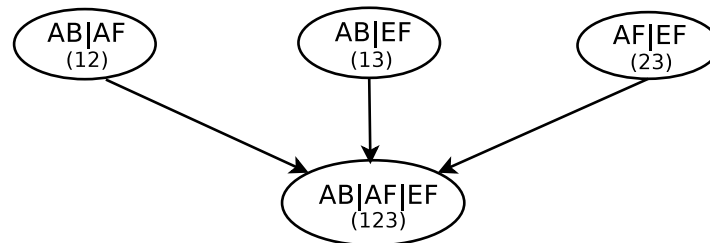
Lemma 4 *The partial order graph of minimal DNF generators is disconnected.*

PROOF: We can prove it by counter-example. Consider the example in Table 1.1. We ignore negated items for now. $AB|AF|EF$ is a minDNF with $T(AB|AF|EF) = 123$. However, all its parents are not minimal DNFs. Take its parent $B|AF|EF$ as an example, $T(B) = 135$, $T(AF) = 2$, $T(EF) = 3$, thus property (a) in Theorem 1 is violated. Similarly, all its children are not minimal DNF generators. For instance, $AB|ACF|EF$ is not a minimal DNF generator since property (b) in Theorem 1 is violated. ■

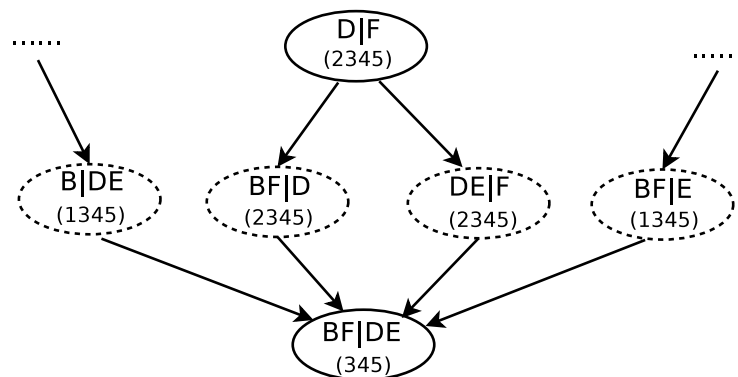
So, if we only keep minDNF expressions in the partial order graph, $AB|AF|EF$ would become an isolated point, and would never be reached! We provide two sampling solutions below.

3.3.1 Sampling in Clause-wise State Space

The first, rather naive, solution is to sample in the clause-wise state space. In particular, rather than adding or deleting an item by AAC, ATC, DTC or DFC each time, we add or delete a clause instead. From Lemma 2, every minDNF consists of only minimal AND-clauses. Also by Corollary 1, any clause-wise subset is also a minDNF. Thus, the partial order graph is connected and all parents of the nodes in the graph will be minDNFs as shown in Figure 3.1(a). The solid ovals represent nodes which are minDNFs. However, this naive idea requires that we first mine the complete set of minimal AND-clauses from the dataset. As we shall see in the experiments, for reasonable support values mining all possible minimal AND-clauses is very expensive or intractable.



(a) Clause-wise



(b) Item-wise

Figure 3.1: Clause- and Item-wise State Space

3.3.2 Sampling in Item-wise State Space

Given that the item-wise state space is disconnected, in order to guarantee all minDNFs are reachable, we also need to keep non-minimal DNFs in the graph. However, the goal is to reduce the number of non-minimal DNF in the graph to as few as possible while retaining all possible minDNFs. The following two lemmas help in this direction.

Lemma 5 *Let Z be a DNF that violates Lemma 2. No extension of Z (by AAC or ATC operations) can result in a minDNF.*

PROOF: At least one of the clauses in Z is not a minimal AND-clause. Any future extension by adding a literal to Z cannot be a minDNF, since all its minimal AND-clause subsets must also be minimal by Lemma 1. ■

This lemma states that any DNF that violates Lemma 2 should be removed from the graph, which can greatly reduce the size of the search space. Furthermore, we also remove any DNF node that violates Lemma 3. Note that this pruning still keeps the graph connected since it is always possible to find other paths. As an example, for the data in Table 1.1, $DE|E$ should be removed since one of its parents $DE|EF$, which is a minimal generator, can be reached by another path, namely from $DE|F$. As mentioned earlier, Definition 1 is a sufficient condition for Lemma 2 and Lemma 3, but not a necessary condition. There still exist DNFs that satisfy Lemma 2 and Lemma 3 but are not minimal generators.

Transition Probability Matrix: It is well known that a regular random walk on the partial order graph favors the nodes with higher degree [80]. In fact, if the edges are weighted, and the graph is undirected (i.e., symmetric weights: $w(u, v) = w(v, u)$ for all connected node pairs), then nodes are sampled proportional to the total weight of a node $s(u) = \sum_v w(u, v)$. We state without proof from [80]: In an ergodic random walk with an associated weighted connected (undirected) graph, the stationary distribution of a vertex is directly proportional to the sum of the edge weight incident to that vertex.

Consider a random walk on the partial order graph where there are both minimal and non-minimal DNF generators, with the transition probability matrix

P given as:

$$P(u, v) = \begin{cases} \frac{w(u, v)}{\sum_{x \in N_u} w(u, x)} & \text{if } v \in N_u \\ 0 & \text{otherwise} \end{cases} \quad (3.3)$$

Here N_u denotes the *neighbors* of u , i.e., the set of nodes adjacent to u . Further, let $N_u^m = \{v \in N_u | v \text{ is a minDNF}\}$ and $N_u^n = \{v \in N_u | v \text{ is not a minDNF}\}$ denote set of u 's neighbors that are minDNF and not minDNF, respectively. Also, let $d_u^m = |N_u^m|$ and $d_u^n = |N_u^n|$ be the minDNF degree and non-minDNF degree of expression u . Clearly the degree of u is given as $d_u = |N_u| = d_u^m + d_u^n$. We define the weight $w(u, v)$ on each edge (u, v) as follows:

$$w(u, v) = \begin{cases} \frac{(1-\alpha)c}{\max\{d_u^m, d_v^m\}} & \text{if } u \text{ and } v \text{ are minDNFs} \\ \frac{\alpha c}{d_v^n} & \text{if } v \text{ is minDNF but } u \text{ is not} \\ \frac{\alpha c}{d_u^m} & \text{if } u \text{ is minDNF but } v \text{ is not} \\ 1 & \text{if } u \text{ and } v \text{ are not minDNFs} \end{cases} \quad (3.4)$$

Here $0 < \alpha < 1$ is a weighting term, and $c > 0$ is a scaling constant. We shall see that α controls the degree of non-uniformity in the sampling, and along with c also affects the convergence rate of the sampling method (it has no impact on the correctness). From the definition, one can verify that the edge weights in the graph are symmetric and the transition probability matrix is stochastic. Moreover, the weights favor transitions to minDNF nodes. We prove that the defined random walk converges to a stationary distribution.

Theorem 2 *An ergodic random walk on the weighted graph where the transition matrix P is defined via Eq. (3.3), using the weight function in Eq. (3.4) is reversible. Further, the random walk converges to a stationary distribution.*

PROOF: Essentially, we can show that P satisfies the detailed balance condition: $\forall s_i, s_j \in \mathcal{S}, \pi_i P(i, j) = \pi_j P(j, i)$. Furthermore, the walk is finite, irreducible, and can be made aperiodic with minor tweaking [80]. Details omitted due to lack of space. ■

Definition 2 *Let π denote the stationary distribution for a Markov chain, where*

$\pi(u)$ denotes the probability of visiting node u . The non-uniformity of a random walk is defined as the ratio of the maximum to the minimum probability of visiting a minDNF node.

Theorem 3 *The non-uniformity of minDNF sampling defined by Eq. (3.4) is bounded by the ratio $1/\alpha$.*

PROOF: The stationary distribution for any node v is given as $\pi(v) = \frac{s(v)}{W}$, where $s(v) = \sum_{y \in N_v} w(v, y)$ is the total weight for node v , and $W = \sum_v s(v)$ is the sum of the weights over all nodes in the Markov chain [80]. W is a constant for a given graph, thus $\pi(u) \propto s(u)$. Let u be a minDNF expression, with minDNF degree d_u^m and non-minDNF degree d_u^n . The total weight for u is given as $s(u) = \sum_{v \in N_u} w(u, v) = d_u^m \cdot \frac{\alpha c}{d_u^n} + (1 - \alpha)c \sum_{y \in N_u^m} \frac{1}{\max(d_u^m, d_y^m)}$. Note that $\sum_{y \in N_u^m} \frac{1}{\max(d_u^m, d_y^m)} \leq d_u^m \cdot \frac{1}{d_u^m} = 1$. Equality is achieved only if $d_u^m \geq d_y^m$ for all $y \in N_u^m$, in which case $s(u) = \alpha c + (1 - \alpha)c = c$. On the other hand, in the worst case we may assume that $d_y^m \gg d_u^m$ so that the second term vanishes in the limit, in which case we have $s(u) = \alpha c$. Thus the worst-case non-uniformity in sampling is $\frac{c}{\alpha c} = \frac{1}{\alpha}$. ■

minDNF Sampling Algorithm

Input: $\mathcal{D}, \sigma_{\min}, \sigma_{\min}^c, k$

Output: k minimal DNF generators

1. $B =$ select a frequent item randomly
2. IF **is_minimal**(B)
3. Output B , insert B in \mathcal{B}
4. IF $|\mathcal{B}| == k$ THEN return //k minDNFs sampled
5. $\mathcal{F} =$ **Compute-Local-Neighborhood**(B)
6. $\mathbf{P} =$ **Local-Transition-Matrix**(B, \mathcal{F}) //Eq. (3.3), (3.4)
7. Select a DNF B_{next} from \mathcal{F} proportional to \mathbf{P}
8. Set $B = B_{next}$, and go to Line 2

Compute-Local-Neighborhood(B)

9. For each Boolean expression f in neighborhood of B
10. IF $sup(f)$ satisfies σ_{\min}^c and σ_{\min}
11. If Lemma 2 and Lemma 3 are satisfied
12. Insert f in \mathcal{F}
13. If conditions (a), (b) in Theorem 1 satisfied
14. Set $f.min = \text{True}$;

Figure 3.2: minDNF Sampling Algorithm

The pseudo-code for the minDNF sampling algorithm is outlined in Figure 3.2. The method always starts by picking a random frequent item (or its negation; we omit that here). Given the current node B , we first check if it minimal, and if so add it to the sampled set of patterns \mathcal{B} . If k steps have been performed, we stop (line 4). Otherwise, in line 5, determine all the immediate parents and children of the current node B that satisfy support constraints, as given in the function **Compute-Local-Neighborhood** in lines 9-14. To get all possible parents and children of the current node B , the four operations AAC, ATC, DTC, DFC are used in line 9. In Line 10 we test the support and prune out those patterns that do not satisfied the conditions. In line 11, we use Lemmas 2 and Lemma 3 to determine whether f is qualified to be remain in the partial order graph. We further test property (a) and (b) in Theorem 1 for f to determine its minimality. Returning back to line 6, we compute the transition probability \mathbf{P} according to Equation 3.3, 3.4. Then we select a DNF expression B_{next} proportional to P to continue the walk in lines 7-8. Note that ideally, we should have a burn-in period for the random walk before patterns are output. We can start counting patterns after a sufficient number of steps (for line 4 check). We omit these details here, for clarity.

3.3.3 Optimizations

Transition Probability Matrix: Whereas Eq. (3.4) gives a good guarantee on the sampling quality, it can be expensive to compute, since we have to determine the minDNF and non-minDNF degree for a node, as well as for all of its neighbors N_u . Instead, we propose another weighting scheme that leads to much faster sampling, without sacrificing the sampling quality too much:

$$w(u, v) = \begin{cases} 1 & \text{if } u \text{ and } v \text{ are minDNFs} \\ \frac{c}{d_v} & \text{if } v \text{ is minDNF but } u \text{ is not} \\ \frac{c}{d_u} & \text{if } u \text{ is minDNF but } v \text{ is not} \\ 0.5 & \text{if } u \text{ and } v \text{ are not minDNFs} \end{cases} \quad (3.5)$$

We note that if the nodes u and v are both either minDNFs or non-minDNFs then we do not need to compute their degrees. Only if one of the nodes is a minDNF, we have to compute its degree (d_u or d_v), but we do not have to determine its minDNF or non-minDNF degrees. The theorem below, shows that sampling quality is still good:

Theorem 4 *The sampling non-uniformity of weighting scheme in Eq. (3.5) is bounded by $1 + d^m/c$, where d^m is maximum minDNF degree of a node.*

PROOF: Let u be a minDNF node, with minDNF degree d_u^m and non-minDNF degree d_u^n . The total weight for u is given as $s(u) = \sum_{v \in N_u} w(u, v) = d_u^m \cdot \frac{c}{d_u} + d_u^n \leq c + d_u^m$. The last step holds since $d_u^n \leq d_u$. The least weight at any node occurs when $d_u^m = 0$, and the most weight is when $d_u^m = \max_u \{d_u^m\} = d^m$. Thus, the non-uniformity is given as $\frac{c+d^m}{c} = 1 + d^m/c$. ■

In practice, this weighting scheme works well. One reason for this is that the expected minDNF degree of a node is much better than the worst-case d^m given above.

Random Walks with Jumps and Restarts: Even after pruning, the partial order graph of sampling minimal DNF generators is large. The walker may be thus get trapped in local regions of the graph which consists of non-minimal DNFs. If this happens, our algorithm will not output minimal DNFs even after a long run, although the samples are guaranteed to be uniform. To avoid getting stuck in local parts, we use the following two strategies: i) Random Walks with Random Jumps (RWRJ): In case the algorithm outputs no minimal DNF generators even after r consecutive steps, we abort the current path, and randomly jump to any earlier minimal DNF generator in the history as its new start. Any such node is then deleted, so that it will not again be chosen as a jump point. ii) Random Walks with Restart (RWR): At each step in the random walk, we enable a certain probability r that the walker jumps back to the root node (empty itemset). We confirm empirically that RWRJ is the better strategy.

AND- and OR-Clause Cache: To further improve execution time, we pre-compute the frequent AND-clauses and OR-Clauses of length 2, and store them

in a hash table, so that they can be used to quickly test for the valid ATC and AAC operations. For example, when we apply ATC operations on a clause Z_i in DNF Z , we first get all frequent candidate items I_{ij} to be added for each literal $z_{ij} \in Z_i$. Then the candidate items to be added by ATC for the clause Z_i are $\{ \bigcap_j I_{ij} | z_{ij} \in Z_i \}$. We can do so quickly by looking up the candidate items in the hash table. This step avoids searching the whole \mathcal{Z} space when we apply ATC operations and thus improves the efficiency.

Fast Minimality Determination: Since we perform random walks on a partial order graph that consists of both minimal and non-minimal DNFs, Lemma 6 and Lemma 7 mentioned below can help to quickly determine the minimality of a DNF when performing random walks without explicitly testing properties (a), (b) in Theorem 1, which can save a lot of computational overhead.

Lemma 6 *Let $Z = \bigvee_{i=1}^m Z_i$, with $m \geq 2$, be a general DNF expression that violates property (a) in Theorem 1. Let $T(Z_k) \subseteq \bigcup_{j \neq k} T(Z_j)$. By adding an item to clause Z_k in Z , the resulting DNF cannot be minDNF.*

PROOF: The tidset of a clause is anti-monotonic. By adding an item x to clause Z_k , resulting in clause Z'_k , i.e., $Z'_k = Z_k \wedge x$, we still have $T(Z'_k) \subseteq \bigcup_{j \neq k} T(Z_j)$. Hence property (a) is violated. ■

However, note that adding an item to other clauses rather than Z_1 in Z can result in a minDNF node.

Lemma 7 *Let $Z = \bigvee_{i=1}^m Z_i$ be a general DNF expression, with $m \geq 2$, and let clause $Z_k \in Z$ violate property (b) in Theorem 1. Adding an item to clause Z_k cannot result in a minimal DNF expression.*

PROOF: Since Z_k violates property (b) in Theorem 1, there exists item $x \in Z_k$, such that $T(Z_k) \cup (\bigcup_{j \neq k} T(Z_j)) = T(Z'_k) \cup (\bigcup_{j \neq k} T(Z_j))$, where $Z_k = Z'_k \wedge x$. Let $Z''_k = Z_k \wedge y = Z'_k \wedge x \wedge y$ and consider the DNF expression $Z'' = Z''_k \vee (\bigvee_{j \neq k} Z_j)$. Assume that Z'' is minDNF, which implies that $(Z'_k \wedge x \wedge y)$ is minAND by Lemma 1. This in turn implies that the difference set $D_y = T(Z'_k \wedge y) - T(Z'_k \wedge x \wedge y)$ is non-empty. We consider three cases: (1) If $D_y \subseteq \bigcup_{j \neq k} T(Z_j)$, then $T(Z'_k \wedge x \wedge y) \cup (\bigcup_{j \neq k} T(Z_j)) =$

$T(Z'_k \wedge y) \cup (\bigcup_{j \neq k} T(Z_j))$, which contradicts the assumption that Z'' is a minDNF. (2) If $D_y \cap \bigcup_{j \neq k} T(Z_j) = \emptyset$, then we have $D_y \subseteq T(Z'_k) \subseteq T(Z'_k) \cup (\bigcup_{j \neq k} T(Z_j)) = T(Z_k) \cup (\bigcup_{j \neq k} T(Z_j))$, which implies that $D_y \subseteq T(Z_k) = T(Z'_k \wedge x)$. However, by definition, $D_y \subseteq T(Z'_k \wedge y)$. Hence $D_y \subseteq T(Z'_k \wedge x) \cap T(Z'_k \wedge y) = T(Z'_k \wedge x \wedge y)$. But this implies that $D_y = \emptyset$, which contradicts the assumption that $Z'_k \wedge x \wedge y$ is minAND. (3) If $D_y \not\subseteq \bigcup_{j \neq k} T(Z_j)$ and $D \cap \bigcup_{j \neq k} T(Z_j) \neq \emptyset$ then we can divide D_y into two parts $D_y = D'_y \cup D''_y$, such that $D'_y \subseteq \bigcup_{j \neq k} T(Z_j)$, $D''_y \cap \bigcup_{j \neq k} T(Z_j) = \emptyset$, and $D'_y \neq \emptyset, D''_y \neq \emptyset$. Similar to step (2), $D''_y \subseteq T(Z'_k \wedge x \wedge y)$, which implies that $D_y = D'_y \cup D''_y = T(Z'_k \wedge y) - T(Z'_k \wedge x \wedge y) = D'_y$. However, this implies that $D''_y = \emptyset$, which is a contradiction. From the three cases above, we conclude that Z'' is not minDNF. ■

Once again, note that adding an item to $Z_j, j \neq k$, may possibly generate a minimal DNF. Consider an example from Table 1.1 again, with DNF $D|BF$. It violates property (b) in Theorem 1 since $T(D|BF) = T(D|F)$ and $D|F$ is a minimal DNF generator. However, one extension of this DNF is $BF|DE$, which is a minimal DNF generator. Figure 3.1(b) shows this example. The solid ovals in the figure are minimal DNF generators, and dashed ovals represent non-minimal DNFs.

3.3.4 Sampling Minimal AND-clauses

Lemma 8 *The partial order graph on minimal AND-clause is connected.*

PROOF: According to Lemma 1, any subset of a minimal AND-clause generator is also the minimal AND-clause. Thus, in the partial order graph a node is connected to to all its immediate subsets/parents. ■

For example, if ABC is a minAND then ABC has three parents, AB , AC and BC , which are all minANDs. Given this connected state-space, a simple symmetric transition probability matrix suffices to sample minAND expressions:

$$P(u, v) = \begin{cases} \frac{1}{\max(d_u, d_v)} & \text{if } v \in N_u \\ 1 - \sum_{x \in N_u} P(u, x) & \text{if } u = v \end{cases} \quad (3.6)$$

We can use this matrix in the algorithm in Figure 3.2 to mine all minAND clauses.

Table 3.1: Classification Performance: Accuracy \pm Standard Error: cls denotes #classes

| dataset | description | | | SVM | | minDNF Sampling | | | minAND Sampling | | |
|---------------|-------------|-------|-------|---------------------------------|---------------------------------|--------------------------------|---------------------------------|--------------------------------|-----------------|--------------------------------|---------------------------------|
| | cls | trans | items | SVM-orig | SVMd | minDNF | minDNF $\sigma_{\min} = 5\%$ | minDNF* | minAND | minAND $k = 500$ | minAND $\sigma_{\min} = 5\%$ |
| adultsample | 3 | 977 | 113 | 59.5 \pm 11.7 | 36.6 \pm 0.7 | 79.8 \pm 0.9 | 81.8\pm1.3 | 57.1 \pm 1.6 | 48.6 \pm 1.4 | 45.1 \pm 0.5 | 75.2 \pm 0.9 |
| anneal | 5 | 898 | 73 | 33.6 \pm 5.0 | 43.7 \pm 3.5 | 97.6\pm0.4 | 97.2 \pm 0.4 | 92.0 \pm 0.6 | 80.0 \pm 0.6 | 91.7 \pm 0.7 | 82.5 \pm 0.6 |
| audiology | 24 | 226 | 154 | 33.2 \pm 2.8 | 33.2 \pm 2.8 | 79.7\pm3.0 | 75.1 \pm 2.3 | 48.3 \pm 3.3 | 43.8 \pm 2.0 | 33.4 \pm 1.4 | 37.5 \pm 1.5 |
| balancescale | 3 | 625 | 20 | 87.8\pm1.0 | 87.8\pm1.0 | 71.7 \pm 1.4 | 74.1 \pm 1.3 | 72.0 \pm 1.4 | 46.8 \pm 3.9 | 74.0 \pm 1.2 | 15.7 \pm 7.7 |
| breastwisc1 | 2 | 683 | 30 | 86.7 \pm 0.8 | 60.8 \pm 1.6 | 95.4 \pm 0.5 | 95.5\pm0.6 | 95.0 \pm 0.6 | 75.6 \pm 1.1 | 93.2 \pm 0.8 | 77.2 \pm 0.5 |
| breastwisc2 | 2 | 699 | 90 | 88.0 \pm 1.1 | 88.0 \pm 1.1 | 94.4\pm0.6 | 91.9 \pm 0.9 | 89.4 \pm 1.0 | 61.5 \pm 1.0 | 72.2 \pm 1.3 | 86.6 \pm 0.9 |
| breastwisc3 | 2 | 683 | 89 | 84.2 \pm 1.5 | 84.0 \pm 1.5 | 93.8\pm0.6 | 93.8\pm0.9 | 87.1 \pm 1.1 | 45.8 \pm 1.0 | 93.6 \pm 0.8 | 52.1 \pm 0.3 |
| breastcancer | 2 | 286 | 41 | 71.3 \pm 1.3 | 72.0\pm1.0 | 67.8 \pm 3.0 | 68.5 \pm 2.3 | 64.7 \pm 2.1 | 54.8 \pm 2.3 | 60.3 \pm 2.1 | 68.5 \pm 2.4 |
| brownselected | 3 | 186 | 182 | 89.8 \pm 1.5 | 39.2 \pm 2.1 | 99.5\pm0.4 | 99.0 \pm 0.6 | 88.3 \pm 2.1 | 50.4 \pm 1.8 | 64.4 \pm 2.5 | 67.9 \pm 2.9 |
| bupa | 2 | 345 | 7 | 50.1 \pm 3.5 | 54.8 \pm 3.2 | 63.2\pm2.7 | 63.2\pm2.7 | 63.2\pm2.7 | 54.8 \pm 3.2 | 54.8 \pm 3.2 | 54.8 \pm 3.2 |
| car | 4 | 1728 | 21 | 64.0 \pm 0.8 | 64.0 \pm 0.8 | 81.0\pm0.4 | 77.2 \pm 0.6 | 76.7 \pm 0.8 | 71.2 \pm 0.4 | 79.1 \pm 0.8 | 87.2 \pm 0.7 |
| crx | 2 | 690 | 53 | 74.8 \pm 2.5 | 85.4\pm1.1 | 83.6 \pm 1.5 | 83.7 \pm 1.5 | 74.1 \pm 1.5 | 62.1 \pm 1.5 | 76.4 \pm 1.3 | 71.9 \pm 1.2 |
| glass | 6 | 214 | 22 | 48.1 \pm 4.7 | 23.3 \pm 3.9 | 75.7 \pm 1.0 | 77.2\pm1.1 | 75.2 \pm 1.4 | 62.5 \pm 1.6 | 50.5 \pm 0.7 | 70.0 \pm 1.6 |
| hayesroth | 3 | 132 | 15 | 46.1 \pm 4.6 | 46.1 \pm 4.6 | 73.5 \pm 3.5 | 76.8\pm2.8 | 72.3 \pm 4.0 | 68.4 \pm 3.2 | 78.7 \pm 3.3 | 58.9 \pm 4.0 |
| heartdisease | 2 | 303 | 29 | 70.3 \pm 5.5 | 65.7 \pm 2.8 | 78.5 \pm 2.2 | 78.9\pm2.2 | 76.0 \pm 2.7 | 67.2 \pm 3.5 | 62.0 \pm 1.7 | 71.6 \pm 3.1 |
| ionosphere | 2 | 351 | 142 | 83.8 \pm 0.7 | 84.6 \pm 1.6 | 89.5\pm1.5 | 88.8 \pm 1.3 | 86.3 \pm 1.6 | 49.3 \pm 0.8 | 80.8 \pm 1.2 | 46.5 \pm 0.9 |
| iris | 3 | 150 | 12 | 93.3 \pm 2.4 | 68.7 \pm 2.0 | 94.9 \pm 1.3 | 95.5\pm1.4 | 94.9 \pm 1.8 | 95.2 \pm 1.4 | 95.4 \pm 1.3 | 94.7 \pm 1.8 |
| lenses | 3 | 24 | 9 | 83.0 \pm 7.7 | 83.0 \pm 7.7 | 80.3 \pm 6.8 | 72.5 \pm 7.9 | 61.6 \pm 11.1 | 78.2 \pm 6.2 | 87.0\pm8.3 | 86.2 \pm 8.2 |
| lungcancer | 3 | 32 | 157 | 52.9\pm10.3 | 52.9\pm10.3 | 52.1 \pm 7.5 | 44.6 \pm 7.5 | 39.3 \pm 8.2 | 33.2 \pm 6.1 | 31.7 \pm 4.3 | 34.4 \pm 6.8 |
| lymphography | 4 | 148 | 50 | 82.4\pm4.1 | 81.1 \pm 3.0 | 80.0 \pm 3.6 | 79.1 \pm 2.4 | 70.9 \pm 3.3 | 63.6 \pm 3.1 | 75.9 \pm 2.9 | 68.0 \pm 2.6 |
| monks1 | 2 | 556 | 17 | 67.6 \pm 2.6 | 67.6 \pm 2.6 | 84.3 \pm 1.4 | 83.7 \pm 1.4 | 77.5 \pm 1.6 | 65.4 \pm 1.8 | 92.7\pm0.9 | 66.6 \pm 1.6 |
| monks2 | 2 | 601 | 17 | 64.2 \pm 1.1 | 64.2 \pm 1.1 | 66.3 \pm 1.1 | 71.9 \pm 1.7 | 62.6 \pm 1.4 | 53.8 \pm 1.6 | 74.7\pm1.6 | 68.2 \pm 1.7 |
| monks3 | 2 | 554 | 17 | 74.4 \pm 0.6 | 74.4 \pm 0.6 | 93.4 \pm 1.1 | 96.4\pm0.9 | 89.2 \pm 1.2 | 69.7 \pm 1.4 | 93.6 \pm 1.1 | 84.5 \pm 1.0 |
| postoperative | 3 | 90 | 20 | 66.7 \pm 1.8 | 67.8\pm2.1 | 57.4 \pm 4.3 | 55.1 \pm 3.8 | 56.3 \pm 3.1 | 58.4 \pm 4.7 | 58.2 \pm 4.2 | 61.6 \pm 3.4 |
| primarytumor | 21 | 339 | 37 | 28.6 \pm 0.6 | 28.6 \pm 0.6 | 40.2\pm2.0 | 38.6 \pm 2.1 | 32.6 \pm 2.1 | 30.0 \pm 1.7 | 36.2 \pm 2.5 | 33.9 \pm 2.3 |
| promoters | 2 | 106 | 228 | 72.5 \pm 6.5 | 72.5 \pm 6.5 | 74.2\pm3.0 | 65.2 \pm 3.8 | 62.5 \pm 3.9 | 57.6 \pm 3.2 | 65.5 \pm 4.3 | 66.1 \pm 4.6 |
| shuttle | 2 | 253 | 16 | 96.5 \pm 0.7 | 96.8 \pm 1.0 | 97.1\pm0.9 | 94.4 \pm 1.2 | 90.5 \pm 1.8 | 81.2 \pm 2.3 | 96.4 \pm 1.1 | 95.0 \pm 1.5 |
| tictactoe | 2 | 958 | 27 | 67.8 \pm 1.0 | 67.7 \pm 0.9 | 78.7\pm1.2 | 76.9 \pm 1.3 | 70.2 \pm 1.3 | 47.4 \pm 1.0 | 68.8 \pm 1.6 | 76.1 \pm 1.4 |
| titanic | 2 | 2201 | 8 | 76.8 \pm 1.1 | 76.8 \pm 1.1 | 79.0 \pm 0.9 | 77.9 \pm 1.0 | 79.0 \pm 0.9 | 78.9 \pm 1.0 | 79.1\pm0.9 | 79.0 \pm 0.9 |
| voting | 2 | 435 | 32 | 91.0 \pm 0.7 | 91.3 \pm 0.8 | 94.6\pm1.1 | 93.9 \pm 0.8 | 91.8 \pm 1.0 | 71.9 \pm 1.1 | 83.1 \pm 1.4 | 78.3 \pm 1.1 |
| wdbc | 2 | 569 | 64 | 88.9 \pm 4.3 | 75.9 \pm 3.8 | 95.3 \pm 0.8 | 95.5\pm0.8 | 92.9 \pm 1.0 | 69.6 \pm 1.0 | 84.8 \pm 0.9 | 85.2 \pm 0.8 |
| wine | 3 | 178 | 37 | 85.9 \pm 4.8 | 96.7 \pm 2.0 | 97.9\pm0.5 | 97.6 \pm 1.1 | 94.4 \pm 1.3 | 76.3 \pm 2.6 | 95.9 \pm 1.3 | 74.7 \pm 1.9 |
| yeastRPR | 3 | 186 | 182 | 89.8 \pm 1.5 | 39.2 \pm 2.1 | 99.3 \pm 0.7 | 99.7\pm0.2 | 87.6 \pm 2.0 | 59.7 \pm 2.2 | 73.7 \pm 1.8 | 43.6 \pm 2.0 |
| zoo | 7 | 101 | 36 | 95.1 \pm 1.5 | 95.1 \pm 1.5 | 96.3\pm1.6 | 96.2 \pm 1.5 | 89.5 \pm 2.1 | 82.5 \pm 3.5 | 95.5 \pm 1.9 | 83.2 \pm 3.7 |

3.4 Experiments

We evaluate the benefits of mining minDNF expressions by using them as features for categorical data classification task. We also evaluate the sampling quality. We compare our results with Blossom [38], which is the only current algorithm that can mine minDNF patterns (although it is a complete method). For pure-AND clauses, we also compare with CHARM-L [35], a state-of-the-art method for mining minimal AND-clauses (i.e., minimal generators.) All experiments are performed on a quad-core Intel i7 3.5GHz CPU, with 16GB memory, and 2TB disk, running Linux (Ubuntu 11.10). The minDNF sampling code was implemented in C++.

3.4.1 Classification Performance

We experimented with a wide range of datasets from the UCI repository [58] as shown in Table 3.1. First, each of the 34 datasets was converted into a categorical one using entropy-based discretization [96], as implemented in the Orange data mining suite [97]. The total number of transactions and items in each dataset, and the number of classes (ranging from 2 to 24) are shown in the table. Next we ran a linear SVM [46, 98, 99], on the original (non-discretized) dataset, as well as on the discretized dataset, using 5-fold cross-validation. For a given run of minDNF sampling, we converted each of the mined minDNF patterns into a binary attribute, that takes on the value 1 if a transaction satisfies the minDNF formula, and 0 otherwise. This binary-valued dataset is then classified using linear SVM with 5-fold cross-validation, again using the Orange library (which in turn uses LIBSVM [100]). Since the sampling is randomized, we repeat the sampling 10 times, and report the averages.

For minDNF and minAND sampling the default parameters are as follows: We use the random walks with random jump approach, with $j = 3$. We use $\alpha = 0.9$ and set c to the average transaction length. The minimum support for the DNF and clause were both set to 1, i.e., $\sigma_{\min} = \sigma_{\min}^c = 1$. Finally, we sampled $k = 100$ minDNF patterns. Thus, for minDNF and minAND the number of “items” or features is at most k for all datasets (it can be less after removing duplicates). Note also that we do not perform any feature selection (it may be possible to further

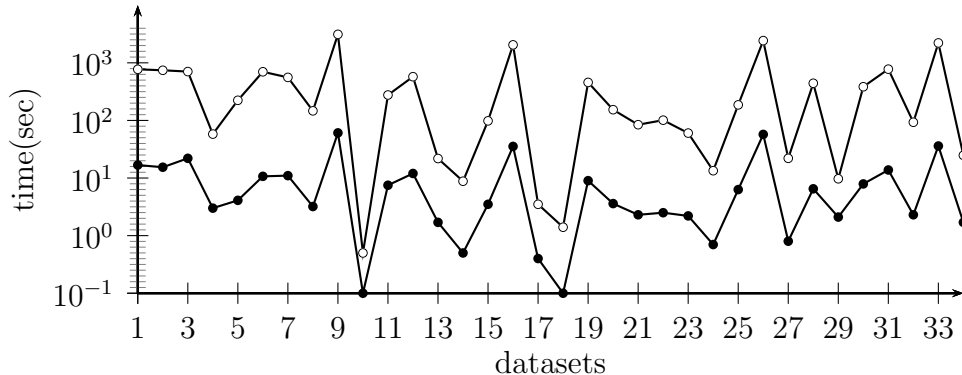


Figure 3.3: Time (in sec) for minDNF (white dots) and minDNF* (black dots)

improve the performance if this is done).

Table 3.1 shows the 5-fold cross-validation classification accuracy and standard-error for each algorithm, over each of the datasets, averaged over 10 runs. The best results are shown in bold. In the table, results for minDNF sampling using the weight matrix in Eq. (3.4) are denoted as minDNF, whereas results using the faster weight computation in Eq. (3.5) are denoted as minDNF*. SVM-orig and SVMd denote the performance of SVM on the original and discretized dataset, respectively. Finally, non-default parameters are indicated in the table (third row), when we compare minAND with $k = 500$, and minDNF/minAND with $\sigma_{\min} = 5\%$.

minDNF Time: The total time for minDNF and minDNF* is shown in Figure 3.3 and actual numbers are recorded in Table 3.2. The datasets are numbered in the order they appear in Table 3.1. The running time is affected by the number of items, since the more the items, the more the number of neighbors, which affects the weight/transition probability computation time. However, note that these results are with support one, and substantial speedup is possible for higher support values. We can observe that minDNF* is typically over an order of magnitude faster than minDNF, though this comes at some penalty in classification performance, as we describe next.

minDNF versus SVMd: Our minDNF sampling algorithm yields a near-uniform sample and we can see from the classification accuracies in columns 7 & 8 that the sampled minDNF patterns make excellent features. In 24 out of the 34 datasets, they

Table 3.2: Time (in sec) for minDNF and minDNF*

| dataset | minDNF | minDNF* |
|---------------|--------|---------|
| adultsample | 772.0 | 16.8 |
| anneal | 742.2 | 15.4 |
| audiology | 704.9 | 22.0 |
| balancescale | 57.9 | 3.0 |
| breastwisc1 | 223.7 | 4.1 |
| breastwisc2 | 699.0 | 10.7 |
| breastwisc3 | 558.6 | 11.0 |
| breastcancer | 146.7 | 3.2 |
| brownselected | 3123.3 | 60.8 |
| bupa | 0.5 | 0.1 |
| car | 276.3 | 7.5 |
| crx | 574.1 | 12.0 |
| glass | 21.8 | 1.7 |
| hayesroth | 8.8 | 0.5 |
| heartdisease | 98.3 | 3.5 |
| ionosphere | 2049.8 | 35.4 |
| iris | 3.5 | 0.4 |
| lenses | 1.4 | 0.1 |
| lungcancer | 457.2 | 9.0 |
| lymphography | 153.5 | 3.6 |
| monks1 | 84.3 | 2.3 |
| monks2 | 101.0 | 2.5 |
| monks3 | 60.3 | 2.2 |
| postoperative | 13.4 | 0.7 |
| primarytumor | 186.2 | 6.3 |
| promoters | 2428.9 | 57.2 |
| shuttle | 22.0 | 0.8 |
| tictactoe | 441.7 | 6.5 |
| titanic | 9.7 | 2.1 |
| voting | 384.2 | 7.9 |
| wdbc | 776.8 | 13.8 |
| wine | 93.0 | 2.3 |
| yeastRPR | 2215.1 | 36.0 |
| zoo | 24.8 | 1.7 |

yield the best accuracy among all methods, including SVM-orig (on the original) and SVMd (on the discretized datasets). On three datasets, the differences between minDNF and best method is not significant. On two datasets SVMs substantially outperform minDNF (namely, balancescale and postoperative, where the difference is more than 10%).

minDNF versus minAND: Comparing the Boolean expression features comprising minDNF patterns versus minAND patterns (both with $k = 100$), we find that minDNF substantially outperforms minAND (see columns 7 and 10). Although initially unexpected, it is perhaps not that surprising, given the fact that minDNFs can be considered as disjunctive rules, and are much more informative than simple conjunctive rules. Over all the 34 datasets, minDNF sampling yields on average 2.69 clauses per DNF expression, with a standard deviation of 1. For fairness, we also compared minDNF (with $k = 100$) to minAND with $k = 500$ features (see column 11). The larger number of features improves minAND in most cases. However, minDNF (with $k = 100$) is still substantially better than minAND ($k = 500$); only on three datasets (lenses, monks1, and monks2) does minAND outperform minDNF. These results indicate that overall minDNF patterns are more effective than minAND patterns.

Table 3.3 shows the averaged number of clauses and averaged total number of items of 100 sampled minDNF patterns in 10 runs in classification experiments (the first and the second column). The third column shows the averaged number of items of 500 sampled minAND patterns in 100 runs. We find that in 32 out of the 34 datasets, the averaged number of clauses in sampled minDNFs is less than or equal to 4.0 (see first column). Two datasets (balancescale and car) have the averaged clause length 6.9 and 4.4. We also find that in 29 out of the 34 datasets (see second column), the averaged number of items in sampled minDNFs is less than or equal to 10.0. Five datasets (balancescale, car, monks1, monks2 and monks3) have the averaged clauses length 22.0, 15.7, 10.1, 10.1 and 10.2. This demonstrates that with approximately equal or even less pattern complexity of minDNF, we can achieve better classification accuracy. minDNF patterns are overall more effective than minAND patterns.

minDNF* Sampling: Since minDNF sampling using Eq. (3.4) does take more time, we also compared with minDNF* that uses the faster weight computation in Eq. (3.5). We can see that minDNF* sampling suffers in performance compared to minDNF. However, minDNF* still outperforms SVMd on 22, SVM-orig on 21,

Table 3.3: Averaged Clauses Length & Items Length of minDNF and minAND

| dataset | minDNF clause len. | minDNF item len. | minAND item len., $k = 500$ |
|---------------|-----------------------|---------------------|--------------------------------|
| adultsample | 3.1±1.4 | 5.0±2.7 | 4.9±1.5 |
| anneal | 4.0±1.9 | 7.4±4.1 | 3.9±1.0 |
| audiology | 2.1±0.8 | 3.1±1.3 | 19.6±5.9 |
| balancescale | 6.9±2.2 | 22.0±8.4 | 3.2±0.8 |
| breastwisc1 | 3.0±1.1 | 6.9±3.1 | 4.2±1.0 |
| breastwisc2 | 3.2±1.2 | 6.3±2.6 | 3.3±1.0 |
| breastwisc3 | 3.5±1.3 | 6.8±2.8 | 3.3±1.0 |
| breastcancer | 3.1±1.3 | 6.2±3.3 | 3.8±1.0 |
| brownselected | 2.2±0.7 | 4.0±1.9 | 12.3±4.3 |
| bupa | 1.1±0.3 | 1.2±0.3 | 1.0±0.0 |
| car | 4.4±1.5 | 15.7±6.8 | 4.5±1.0 |
| crx | 2.9±1.1 | 5.8±2.9 | 5.8±1.4 |
| glass | 2.0±0.9 | 3.4±2.0 | 3.1±0.9 |
| hayesroth | 3.6±1.1 | 7.9±2.9 | 2.6±0.7 |
| heartdisease | 2.1±1.0 | 4.0±2.8 | 4.9±1.2 |
| ionosphere | 2.6±1.0 | 5.2±2.6 | 5.4±1.3 |
| iris | 2.2±0.9 | 3.6±1.9 | 2.2±0.7 |
| lenses | 2.5±0.8 | 5.5±2.4 | 2.8±0.9 |
| lungcancer | 2.2±0.7 | 3.6±1.5 | 6.0±1.2 |
| lymphography | 2.3±0.9 | 4.2±2.2 | 4.9±1.1 |
| monks1 | 3.4±1.1 | 10.1±3.8 | 4.3±1.1 |
| monks2 | 3.7±1.0 | 10.1±3.9 | 4.3±1.1 |
| monks3 | 3.5±0.9 | 10.2±3.4 | 4.3±1.1 |
| postoperative | 2.1±0.9 | 3.8±2.0 | 3.4±1.0 |
| primarytumor | 2.2±0.8 | 4.5±2.4 | 6.7±1.8 |
| promoters | 2.4±0.9 | 5.2±2.4 | 4.0±0.7 |
| shuttle | 3.3±1.0 | 9.2±3.7 | 4.1±1.1 |
| tictactoe | 3.6±1.2 | 9.8±3.8 | 4.9±1.1 |
| titanic | 1.7±0.8 | 3.0±2.1 | 2.1±0.7 |
| voting | 2.6±0.9 | 6.1±2.7 | 6.2±1.3 |
| wdbc | 2.3±1.2 | 4.5±3.1 | 5.1±1.1 |
| wine | 2.4±1.0 | 5.0±2.7 | 4.2±1.1 |
| yeastRPR | 2.2±0.8 | 4.2±2.1 | 13.4±3.6 |
| zoo | 2.0±0.7 | 3.4±1.8 | 4.2±1.0 |

minAND patterns (with $k = 100$) on 31, and minAND (with $k = 500$) on 18 out of the 34 datasets.

Effect of Support: To see the effect of minimum support on classification accuracy, we ran minDNF and minAND sampling with the minimum support set to 5% of the number of transactions (see columns 8 & 12). Overall, we find that adding the frequency constraint is not that beneficial to for minDNF sampling, since mining frequent minDNFs improves the accuracy only slightly in 13 datasets, when com-

pared to the minDNFs with support one. On the other hand, frequent minANDs are better than support-one minANDs on 26 datasets. Mining frequent expressions obviously lowers running times.

Negated Items: We also experimented with minDNF expressions containing negated items. However, in this case the accuracy was slightly better for the negated items in only 5 out of the 34 datasets, which unfortunately came at the expense of a significant increase in the runtime (sometimes by orders of magnitude). We conclude that negated items do not confer significant advantages in terms of classification (at least for the UCI datasets tested).

Our results above clearly demonstrate the value of mining/sampling minDNF patterns, especially in support-less mode (i.e., $\sigma_{\min} = 1$). Besides those, we also study the effect of α in classification performance as well as alternative ways of constructing features by minDNF patterns.

α effect: Our default parameter is $\alpha = 0.9$. Here we also experimented $\alpha = 0.25$ (see column 3 in Table 3.4) and $\alpha = 0.5$ (see column 4 in Table 3.4). Note that column 2 contains the results from our default parameter settings. In 23 out of the 34 datasets, $\alpha = 0.9$ gave the best performance. This is because the random walk favors edges whose two ends are of different types, i.e., one end is a minDNF pattern and the other end is not a minDNF pattern (see Equation 3.4). Thus, by setting $\alpha = 0.9$, and independency among the sampled minDNFs is increased and the redundancy is reduced, compared to both $\alpha = 0.25$ and $\alpha = 0.5$, which conversely could be used to build a more effective classifier.

Alternative Ways of Constructing minDNF Features: For converting each of the mined minDNF patterns into a binary feature, instead of taking on the value 1 if the whole transaction satisfies the minDNF formula and 0 otherwise, we converted each minDNF pattern by taking on the value 1 if (1) XOR of all the clauses in the minDNF is 1 (see column 5 in Table 3.4) and (2) the majority of clauses in the minDNF is satisfied by the transaction (see column 6 in Table 3.4). The experimental results show that our original way of converting into binary features

is still the best among all the three approaches.

3.4.2 Sampling Evaluation

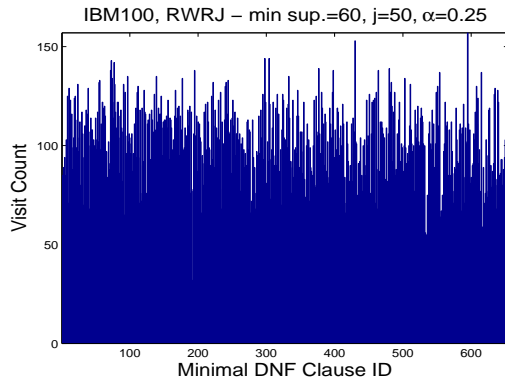
Having shown the effectiveness of minDNF sampling for classification, we now study the sampling quality provided by minDNF and the sensitivity to various parameters. We use both small (first three) and large (last three) datasets shown in Table 3.5 for these experiments. The last four are taken from the FIMI repository [34]. The Gene dataset is from [38], and IBM100 is a synthetic dataset generated using the IBM itemset generator [3].

Random Walk Type: We first show the effect of the type of random walk, i.e., random walk with random jumps (RWRJ, $j = 50$) versus random walks with restart (RWR, $r = 0.02$), as shown in Figures 3.4 and 3.5. With $\sigma_{\min} = 60$, on the IBM100 dataset, there are 654 distinct minDNF patterns (found using Blossom). We ran minDNF sampling for $k = 654 \times 100$ iterations. We use $\alpha = 0.25$ and c is set to the avg. transaction length.

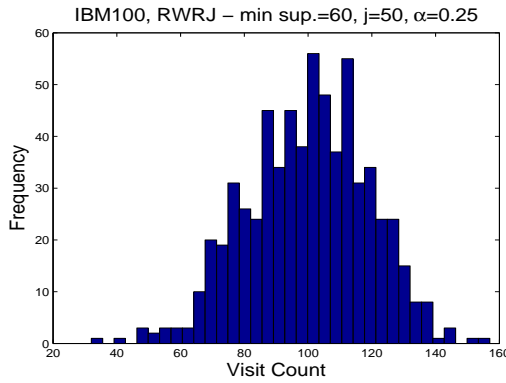
If the dataset has f minDNF patterns and we perform the uniform sampling for $k = f \times t$ steps, the number of times m a specific pattern will be sampled is described by the binomial distribution, $\mathcal{B}(m|k, p)$, where $p = \frac{1}{f}$. The expected number of times a minDNF pattern is visited is given as $kp = f \cdot t \cdot \frac{1}{f} = t$, and the standard deviation is $\sqrt{kp(1-p)} = \sqrt{\frac{t(f-1)}{f}}$. For the IBM100 dataset, we have $f = 654$ and $t = 100$, and thus in the ideal case we expect to see each pattern 100 times, with standard deviation of $\sqrt{100 \cdot 653/654} = 9.99$. Figures 3.4 and 3.5 plot the number of times each pattern is visited (a), and the count histogram (b). The sampling statistics, namely the maximum, minimum, median, and standard deviation of visits counts for RWRJ and RWR are shown in Table 3.6. It is very clear that RWRJ is much superior to the RWR strategy; its median is closer to the ideal case, and the standard deviation is smaller. Whereas the RWRJ strategy jumps to a node in its history, RWR always restarts from the empty pattern. As such RWR is biased towards sampling patterns close to the origin, and this is reflected in the sampling quality.

Table 3.4: Classification Performance (Augmented Experiments): Accuracy \pm Standard Error

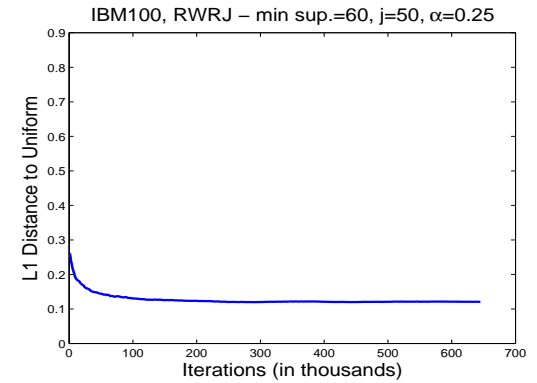
| dataset | minDNF Sampling | | | | |
|---------------|--------------------------------|--------------------------------|--------------------------------|--------------------------------|--------------------------------|
| | minDNF | minDNF $\alpha = 0.25$ | minDNF $\alpha = 0.5$ | minDNF Majority | minDNF XOR |
| adultsample | 79.8 \pm 0.9 | 68.9 \pm 1.3 | 75.5 \pm 1.1 | 77.1 \pm 1.1 | 81.6\pm1.0 |
| anneal | 97.6\pm0.4 | 92.5 \pm 0.8 | 93.0 \pm 0.8 | 87.7 \pm 0.9 | 96.1 \pm 0.5 |
| audiology | 79.7\pm3.0 | 43.3 \pm 2.3 | 55.4 \pm 2.3 | 71.0 \pm 2.4 | 72.6 \pm 2.7 |
| balancescale | 71.7 \pm 1.4 | 73.5\pm1.3 | 72.4 \pm 1.3 | 31.9 \pm 5.6 | 70.5 \pm 1.4 |
| breastwisc1 | 95.4 \pm 0.5 | 95.6 \pm 0.8 | 95.6\pm0.6 | 91.3 \pm 0.7 | 95.0 \pm 0.7 |
| breastwisc2 | 94.4\pm0.6 | 92.0 \pm 0.8 | 92.1 \pm 0.8 | 74.6 \pm 0.9 | 93.1 \pm 0.8 |
| breastwisc3 | 93.8\pm0.6 | 88.9 \pm 1.0 | 91.6 \pm 0.9 | 70.0 \pm 0.7 | 92.8 \pm 0.8 |
| breastcancer | 67.8 \pm 3.0 | 67.6 \pm 2.2 | 69.8\pm2.2 | 68.3 \pm 2.2 | 67.3 \pm 3.1 |
| brownselected | 99.5\pm0.4 | 91.0 \pm 2.2 | 94.5 \pm 1.5 | 98.9 \pm 0.5 | 98.4 \pm 0.7 |
| bupa | 63.2 \pm 2.7 | 63.2 \pm 2.7 | 63.2 \pm 2.7 | 63.2 \pm 2.7 | 63.2 \pm 2.7 |
| car | 81.0\pm0.4 | 77.7 \pm 0.7 | 76.5 \pm 0.8 | 63.9 \pm 0.8 | 76.6 \pm 0.7 |
| crx | 83.6\pm1.5 | 78.3 \pm 1.3 | 80.2 \pm 1.4 | 81.9 \pm 1.6 | 83.6 \pm 1.6 |
| glass | 75.7 \pm 1.0 | 75.0 \pm 1.3 | 76.0 \pm 1.5 | 76.8\pm1.1 | 76.7 \pm 1.2 |
| hayesroth | 73.5 \pm 3.5 | 75.4 \pm 2.7 | 75.7\pm2.4 | 54.8 \pm 4.1 | 72.2 \pm 2.7 |
| heartdisease | 78.5\pm2.2 | 76.9 \pm 2.0 | 77.7 \pm 2.1 | 78.3 \pm 2.2 | 77.5 \pm 2.2 |
| ionosphere | 89.5\pm1.5 | 86.9 \pm 1.4 | 89.5\pm1.5 | 89.5 \pm 1.6 | 89.4 \pm 1.1 |
| iris | 94.9 \pm 1.3 | 94.8 \pm 1.6 | 95.3 \pm 1.5 | 95.2 \pm 1.3 | 95.5\pm1.5 |
| lenses | 80.3\pm6.8 | 67.4 \pm 8.5 | 66.1 \pm 8.7 | 76.1 \pm 7.8 | 60.4 \pm 7.6 |
| lungcancer | 52.1\pm7.5 | 40.8 \pm 6.1 | 40.7 \pm 7.5 | 44.9 \pm 6.6 | 43.0 \pm 7.1 |
| lymphography | 80.0\pm3.6 | 74.5 \pm 3.1 | 77.1 \pm 3.2 | 79.6 \pm 3.2 | 76.9 \pm 3.3 |
| monks1 | 84.3\pm1.4 | 82.1 \pm 1.6 | 77.3 \pm 1.8 | 66.5 \pm 1.8 | 82.5 \pm 1.4 |
| monks2 | 66.3\pm1.1 | 65.0 \pm 1.3 | 65.6 \pm 1.8 | 59.1 \pm 1.7 | 66.1 \pm 1.6 |
| monks3 | 93.4 \pm 1.1 | 92.9 \pm 1.0 | 94.2\pm0.9 | 76.0 \pm 1.4 | 91.0 \pm 1.2 |
| postoperative | 57.4 \pm 4.3 | 59.4\pm4.3 | 59.2 \pm 4.0 | 56.2 \pm 5.0 | 56.0 \pm 5.1 |
| primarytumor | 40.2\pm2.0 | 35.5 \pm 2.1 | 36.4 \pm 2.2 | 40.1 \pm 2.0 | 37.3 \pm 2.2 |
| promoters | 74.2\pm3.0 | 63.6 \pm 4.1 | 63.1 \pm 3.8 | 61.9 \pm 4.5 | 64.7 \pm 4.3 |
| shuttle | 97.1\pm0.9 | 92.3 \pm 1.6 | 92.0 \pm 1.6 | 83.2 \pm 2.4 | 90.7 \pm 1.9 |
| tictactoe | 78.7\pm1.2 | 76.0 \pm 1.3 | 76.2 \pm 1.3 | 62.7 \pm 1.6 | 73.2 \pm 1.2 |
| titanic | 79.0\pm0.9 | 78.4 \pm 0.9 | 79.0\pm0.9 | 78.9 \pm 1.0 | 79.0\pm0.9 |
| voting | 94.6\pm1.1 | 92.4 \pm 1.1 | 93.2 \pm 1.0 | 93.2 \pm 0.9 | 92.0 \pm 1.1 |
| wdbc | 95.3 \pm 0.8 | 92.8 \pm 0.8 | 94.3 \pm 0.8 | 95.6\pm0.9 | 94.5 \pm 0.9 |
| wine | 97.9\pm0.5 | 92.8 \pm 1.5 | 96.5 \pm 1.3 | 96.6 \pm 1.1 | 96.5 \pm 1.1 |
| yeastRPR | 99.3\pm0.7 | 88.6 \pm 2.1 | 94.0 \pm 1.2 | 99.0 \pm 0.6 | 98.2 \pm 0.7 |
| zoo | 96.3\pm1.6 | 90.2 \pm 2.8 | 92.9 \pm 2.2 | 96.2 \pm 1.6 | 96.3\pm1.6 |



(a) Visit Counts

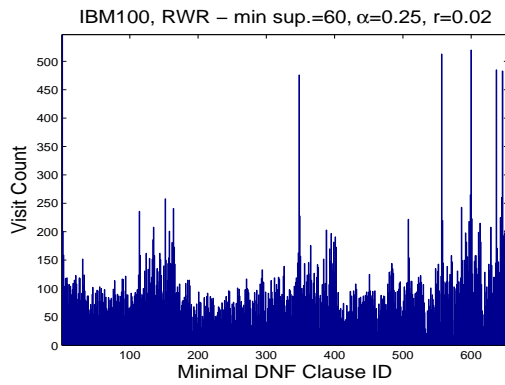


(b) Count Histogram

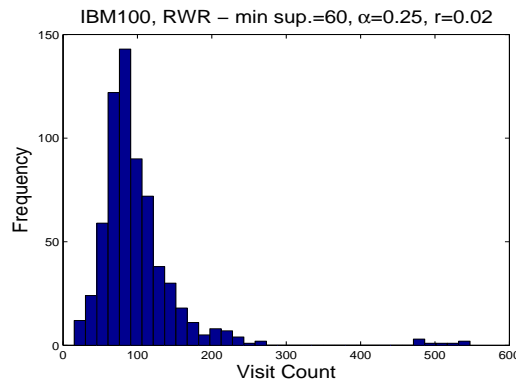


(c) Variation Distance

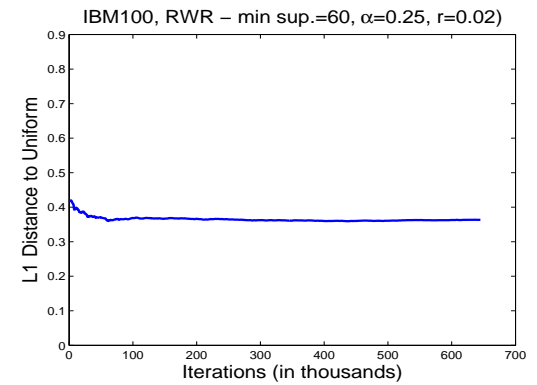
Figure 3.4: Sampling Quality (RWRJ): IBM100



(a) Visit Counts



(b) Count Histogram



(c) Variation Distance

Figure 3.5: Sampling Quality (RWR): IBM100

Table 3.5: Datasets

| Dataset | trans | items | avg. trans len |
|---------|--------|-------|----------------|
| IBM100 | 100 | 20 | 9.3 |
| Gene | 74 | 824 | 86.1 |
| Chess | 3196 | 75 | 37 |
| Connect | 67557 | 129 | 43 |
| Retail | 88162 | 16470 | 10.3 |
| Kosarak | 990002 | 41270 | 8.1 |

Table 3.6: Sampling Statistics: IBM100

| | Maximum | Minimum | Median | Std |
|------|---------|---------|--------|------|
| RWRJ | 157 | 32 | 101 | 19.0 |
| RWR | 547 | 15 | 89 | 60.5 |

Convergence Rate: One important issue in using MCMC sampling is to determine when the initial distribution converges to the stationary distribution and how fast the convergence rate is. It is well known that the mixing time is closely related to the spectral gap, $\gamma = |\lambda_1 - \lambda_2| = |1 - \lambda_2|$, which is defined as the absolute difference between the largest $\lambda_1 = 1$ and the second largest eigenvalue λ_2 of the transition matrix P [101]. The larger the spectral gap, the faster the walk converges. Unfortunately, we cannot compute the entire transition matrix P ; the whole point of sampling is to avoid enumerating all the minDNF patterns. An alternative strategy to measure the convergence rate is to compute the total variation distance, defined as $vd(P^t(s, \cdot), \pi) = 0.5 \sum_{q \in \mathcal{S}} |P^t(s, q) - \pi(q)|$, where s is the initial state, P^t is the transition matrix at time t , and π is the desired stationary distribution.

Figures 3.4(c) and 3.5(c) plot the variation distance for the RWRJ and RWR sampling methods. Here we compute the variation distance empirically. To be more specific, we estimate $P^t(s, q)$ at time t via the count histogram, converted into a probability distribution of visitations, to each pattern q from the initial empty pattern $s = \emptyset$. We compute the variation distance after every 1000 steps. We can see that the distance converges to slightly around 0.12 for RWRJ and to 0.36 for RWR, indicating that RWRJ is the better strategy. We also ran experiments on the Gene and Chess datasets, and obtained similar results (figures omitted due to space constraints).

Table 3.7: Effect of Parameters: IBM100

| α, c, j | npats | mean | std | max | min | med | time |
|-----------------|-------|------|------|------|-----|------|--------|
| $j = 3$ | 636.2 | 15.7 | 10.0 | 60.2 | 1 | 14 | 98.8s |
| $j = 5$ | 635.2 | 15.7 | 8.9 | 48.0 | 1 | 15.8 | 100.2s |
| $j = 10$ | 636 | 15.7 | 8.1 | 45.2 | 1 | 16.4 | 100.2s |
| $j = 50$ | 629 | 15.9 | 7.6 | 38.2 | 1 | 16 | 100.2s |
| $j = 100$ | 637.4 | 15.7 | 7.6 | 39.6 | 1 | 16.2 | 101.6s |
| $\alpha = 0.1$ | 654 | 15.3 | 5.4 | 34 | 1.6 | 15 | 74.8s |
| $\alpha = 0.25$ | 654 | 15.3 | 5.3 | 34.2 | 2.6 | 15 | 80s |
| $\alpha = 0.5$ | 654 | 15.3 | 5.6 | 35.4 | 2.2 | 15 | 89.4s |
| $\alpha = 0.75$ | 651.8 | 15.3 | 7.0 | 39.4 | 1 | 16 | 96.8s |
| $\alpha = 0.9$ | 634.8 | 15.8 | 7.6 | 37.8 | 1 | 16.4 | 101.7s |
| $c = 5$ | 633.2 | 15.8 | 7.3 | 35.8 | 1 | 16.4 | 125.6s |
| $c = avg(9.3)$ | 632.8 | 15.8 | 7.5 | 45.4 | 1 | 16 | 100.0s |
| $c = max(17)$ | 639 | 15.7 | 7.9 | 47.2 | 1 | 16.2 | 88.8s |
| $c = 50$ | 633.2 | 15.8 | 8.8 | 50.6 | 1 | 15.8 | 79.1s |

Effect of α, c, j : Table 3.7 shows the effect of these three parameters on the sampling quality on IBM100 with $\sigma_{\min} = 60$. We set $k = 10000$ iterations. We run each experiment 5 times, and report the average number of distinct minDNFs sampled (npats), mean, standard deviation (std), maximum, minimum, and median (med) of the visit counts, and the average total time. Ideal sampling should yield a mean visit count of $k/f = 15.3$, and a standard deviation of $\sqrt{k/f(1 - 1/f)} = 3.9$, since IBM100 has $f = 654$ minDNF patterns for $\sigma_{\min} = 60$. First, we look at the effect of j , fixing $c = avg$ and $\alpha = 0.9$. Larger j results in a smaller standard deviation, and ideally j should not be constrained. However, for many of the classification datasets the random walk could get trapped in a local region, and therefore, we set $j = 3$ in our earlier experiments. Next, we look at the effect of α , setting $j = 50$ and $c = avg$. We find that larger α takes more time, with a slight increase in std, most likely due to the constraint on j . Lastly, we fix $j = 50$, $\alpha = 0.9$ and vary c . Larger c values take lesser time, but also result in higher deviation. The average c value (9.3 for IBM100) offers an acceptable choice.

Scalability: Table 3.8 shows the time to sample the small (with $k = 1000$) and large datasets (with $k = 100$) for various support thresholds using minDNF and minDNF*. Blossom was unfortunately not able to mine the complete set of patterns

Table 3.8: Running Time: minDNF and minDNF*

| Dataset | 1% | 5% | 10% | 20% |
|---------|-----------|----------|----------|----------|
| IBM100 | 1m50s | 45.5s | 40.2s | 20.9s |
| * | 17.5s | 1.6s | 14.5s | 9.9s |
| Gene | 3h45m12s | 46m13s | 31m1s | 3m45s |
| * | 19m11s | 6m52s | 5m23s | 3m36s |
| Chess | 11h26m11s | 6h41m34s | 5h23m29s | 4h28m33s |
| * | 1h6m8s | 59m22s | 42m15s | 28m33s |
| Connect | 15h52m47s | 8h23m18s | 7h59m22s | 6h31m39s |
| * | 4h44m2s | 2h19m22s | 1h58m53s | 1h34m48s |
| Retail | 50m4s | 1m6s | 35.0s | 3.1s |
| * | 59m25s | 21.5s | 12.8s | 4.7s |
| Kosarak | 27h58m43s | 2h24m39s | 9m55s | 3m41s |
| * | 8h31m4s | 20m3s | 2m14s | 1m40s |

for any of these datasets for the support levels shown even after 24hours for the smaller datasets, and 48hrs for the large ones. We note that whereas minDNF provides better theoretical guarantee, minDNF* is significantly faster (by as much as an order of magnitude). We also compared minAND sampling time with Blossom-MA and CHARM-L, both of which can mine minimal AND-clauses. For example, for the Gene dataset with 10% support, minAND took 0.7s to sample 1000 patterns, whereas CHARM-L took 40m54s and Blossom-MA took 2h58m56s. For lower support values, neither of these methods could finish within 24hours, whereas for 1% support minAND finished in 1.3s. These results confirm that complete mining is practically infeasible, whereas sampling provides a viable alternative.

3.5 Conclusions

In this chapter we presented the first approach to sample the simplest Boolean patterns, namely the minimal DNF expressions. We propose a novel weighting scheme to compute the transition probability matrix for the Markov chain Monte Carlo sampling algorithm, which bounds the amount of non-uniformity in the sampling. Since the method can be slow in practice, we also suggest a faster alternative, that yields effective sampling quality as well. We perform an extensive set of experiments to test various design parameters, and justify our choices. Finally, somewhat

surprisingly, we found that the minimal DNF patterns make very effective features for classification. Via an extensive set of experiments on UCI datasets, we show that our method outperforms simple AND-clause based features, as well as the SVM method, typically by a wide margin, though it does suffer in the runtime comparison. However, the faster weight computation approach yields significantly faster running times, with some loss in the classification accuracy. The minDNF features still remain the effective across the different classifiers.

CHAPTER 4

GRAPH CLASSIFICATION USING GLOBAL TOPOLOGICAL PATTERNS

In this chapter we tackle the challenging problem of graph classification over a set of graphs. Graph classification is an important data mining task, and various graph kernel methods have been proposed in the past decade. These methods have proven to be effective, but they tend to have high computational overhead. In this chapter, we propose an alternative approach to graph classification that is based on feature-vectors constructed from different global topological attributes, as well as global label features. The main idea here is that the graphs from the same class should have similar topological and label attributes. Our method is simple and easy to implement, and via a detailed comparison on real benchmark datasets, we show that our topological and label feature-based approach delivers better or competitive classification accuracy, and is also substantially faster than other graph kernels. It is the most effective method for large unlabeled graphs.

4.1 Introduction

With the proliferation of graph data, there has been a lot of interest in recent years to develop effective methods for classifying graph objects [102]. Applications range from chem-informatics [52, 92] (e.g., compounds that are active or inactive for some target) and bioinformatics [90, 103] (e.g., classifying proteins into different families, classifying tissue samples), to telecommunication networks (e.g., classifying customers based on their calling behavior) and social networks (e.g., classifying users based on their feeds on Twitter, Facebook, etc.).

The graph classification problem can be stated as follows: There is a dataset of graphs $G_i \in \mathcal{D}$, with $i = 1, \dots, N$. Each graph $G_i = (V_i, E_i)$ is given as a collection of vertices $V_i = \{v_{i1}, \dots, v_{in}\}$ and edges $E_i = \{(v_a, v_b) | v_a, v_b \in V_i\}$. The graph G_i

This chapter previously appeared as: G. Li, M. Semerci, B. Yener and M. Zaki, "Effective Graph Classification based on Topological and Label Attributes," *Statistical Analysis and Data Mining*, vol. 5, no. 1, pp. 265–283, August 2012.

may have labels on the nodes and edges, drawn from some common set of labels Σ for the entire dataset \mathcal{D} . Finally, each graph G_i has a corresponding class $y_i \in C$, where C is the set of k categorical class labels, given as $C = \{1, \dots, k\}$. The goal of graph classification is to learn a model $f : \mathcal{D} \rightarrow C$ that predicts the class label for any graph. Typically the model is learned from a *training set* of graphs with known class labels. The model is then evaluated on a *testing set* of graphs. The accuracy of the classification model can be tested by comparing the predicted output $\hat{y}_i = f(G_i)$ with the true class label y_i (provided it is known).

The main challenge in classifying graphs is how to convert the discrete graph objects into numeric features or similarities for effective classification. Graph kernel methods have attracted a lot of attention due to their ability to represent the graph data as a $N \times N$ symmetric, positive semi-definite *kernel matrix* $\mathbf{K} = \{\kappa(G_i, G_j)\}_{i,j=1}^N$ that records the pair-wise similarities between graphs in \mathcal{D} . Conceptually, the kernel function $\kappa(G_i, G_j)$ represents an inner-product between the vectors corresponding to the two graphs G_i and G_j in some N -dimensional *feature space*; see [45] for more details on kernel methods. Once the kernel matrix has been constructed, it is possible to classify the graphs with a Support Vector Machine (SVM) [46, 98, 99], using the supplied kernel matrix \mathbf{K} . There has been a lot of research activity in trying to develop more effective and efficient graph kernel functions κ . These methods can broadly be classified into methods based on random walks [47, 48], shortest paths [49], cycles [50] subtrees [51, 52, 53, 104], and subgraphs [54, 55, 56]. Despite the research above, it is fair to say that efficient and effective graph classification still remains a challenge, especially for large graphs.

In this chapter we propose an alternative approach to constructing a feature-vector for graph classification. Instead of relying on “patterns” like path, cycles, subtrees and subgraphs, we compute several global topological and label attributes from each graph $G_i \in \mathcal{D}$. The values for these attributes yield a numeric feature-vector $F_i = (f_{i1}, \dots, f_{ip})$. The set of feature vectors F_i and the corresponding class labels y_i are then used to construct an SVM classifier. We show that our approach is both effective and scalable compared to state-of-the-art graph kernel methods. We conduct an extensive set of experiments over several real graphs, representing

chemical compounds, proteins, and cell-graph datasets. We demonstrate that our approach yields better or competitive accuracy in a fraction of the time taken by other kernels. Our method is particularly effective in classifying large unlabeled graphs, since it is able to effectively capture the structural differences among the classes.

4.2 Graph Attributes for Classification

As we have seen above, while many sophisticated graph kernels have been proposed, efficiency and scalability remain as challenges, for large graph datasets. Our basic idea is to compute several topological and label attributes for each graph in the dataset, and to use the derived feature-vector attributes for classification. Like most of the graph kernel work, we use a Support Vector Machine (SVM) as the classifier of choice. The graph attributes we use are listed below.

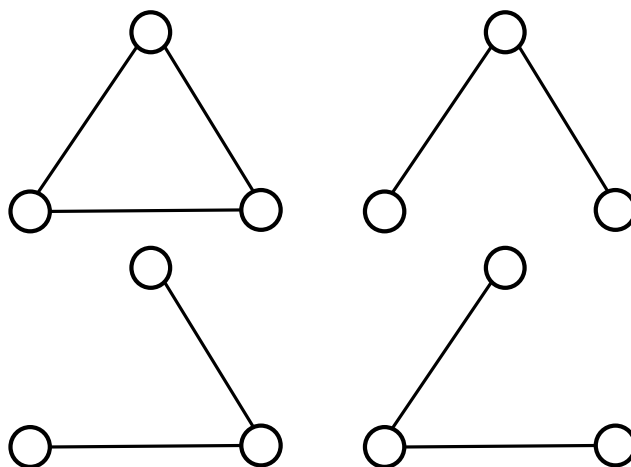


Figure 4.1: A triangle with its three triples

- f-1. **Average degree:** The degree of a node is defined as the number of its neighboring edges. Average degree is the average value of the degree of all nodes in the graph, i.e., $\bar{d}(G) = \sum_i^n d(u_i)/n$, where $d(u_i)$ denotes the degree of node u_i .
- f-2. **Average clustering coefficient:** For a node u , the clustering coefficient $c(u)$ represents the likelihood that any two neighbors of u are connected. More formally, the clustering coefficient of a node u is defined as: $c(u) = \frac{\lambda(u)}{\tau(u)}$, where

$\lambda(u)$ is the number of triangles (complete graph with three nodes) of a node u and $\tau(u) = \frac{d(u)^2 - d(u)}{2}$, the number of triples a node u has. Figure 4.1 shows a triangle and its three triples. Alternatively, the clustering coefficient for node u can be defined as the ratio of the number of actual edges between the neighbors of u to the number of possible edges between them. The clustering coefficient $C(G)$ of a graph is the average of $c(u)$ taken over all the nodes in the graph, i.e., $C(G) = \frac{1}{n} \sum_{i=1}^n c(u_i)$. Here we use $C(G)$ as one of our global graph features. Generally, average clustering coefficient is a very popular metric in network analysis, but in some specific graph datasets, such as chemical compounds, there do not exist many triangles in any graph instance, which results in the clustering coefficient taking on value close to 0.

- f-3. **Average effective eccentricity:** The eccentricity of a node u is defined as $e(u) = \max\{d(u, v) : v \in V\}$, where the distance $d(u, v)$ is the length of the shortest path from u to v . For *effective eccentricity* we take the maximum length of the shortest path from u , so that u can reach at least 90 percent of nodes in the graph. Effectiveness is a more robust measure if we take noise into consideration. The average effective eccentricity is the average of effective eccentricities of all nodes in the graph.
- f-4. **Maximum effective eccentricity (effective diameter):** Maximum effective eccentricity is defined as the maximum value of effective eccentricity over all nodes in the graph. Note that the maximum eccentricity is the graph diameter, i.e., $\text{diam}(G) = \max\{e(u) | u \in V\} = \max\{d(u, v) | u, v \in V\}$. Maximum effective eccentricity is thus the same as effective diameter.
- f-5. **Minimum effective eccentricity (effective radius):** Minimum effective eccentricity is defined as the minimum value of effective eccentricity over all nodes in the graph. Note that minimum eccentricity is called the graph radius, i.e., $\text{rad}(G) = \min\{e(u) | u \in V\}$, thus minimum effective eccentricity is the effective radius.
- f-6. **Average path length (closeness centrality):** The closeness centrality of a node u is defined as the reciprocal of the averaged total path length between

node u and every other node that is reachable from node u , where $u \in V$, i.e., $close(u) = \frac{n-1}{\sum_{v \in V, v \neq u} d(u,v)}$. We take the average of closeness centrality of all nodes as a global metric for a graph.

- f-7. **Percentage of central points:** We define a point u to be a central point if it has an eccentricity equal to the effective radius of the graph, i.e., it satisfies: $\{u \in V : effective-rad(G) = e(u)\}$. The ratio of the number of central points to the total number of points in the graph is selected as a feature.

Table 4.1: Global graph feature vector for the example (F: Feature, V: Value)

| | | | | | | | | | | |
|---|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| F | f_1 | f_2 | f_3 | f_4 | f_5 | f_6 | f_7 | f_8 | f_9 | f_{10} |
| V | 2.10 | 0.00 | 5.75 | 8 | 4 | 0.29 | 0.15 | 1.00 | 0.00 | 0.45 |
| F | f_{11} | f_{12} | f_{13} | f_{14} | f_{15} | f_{16} | f_{17} | f_{18} | f_{19} | f_{20} |
| V | 20 | 21 | 2.56 | 2.15 | 0.0 | 42.00 | 20 | 1.09 | 1.11 | 0.48 |

- f-8. **Giant connected ratio:** A giant component is a subgraph that is connected and has the maximum number of nodes. We take the ratio of the number of nodes of the giant connected component to the total number of nodes in the entire graph as a global metric for a graph. Note that if the entire graph is connected, the ratio is 1, thus in those datasets that are comprised of connected graphs, this attribute will not be a good graph descriptor. However, not all graphs in our experimental study are connected, thus this ratio may be a meaningful attribute to use.
- f-9. **Percentage of isolated points:** We define a isolated point in a graph to be a node with degree zero. The ratio of isolated points to the total number of nodes in the entire graph is considered as a feature. For graphs that are connected, this feature will not be meaningful, but there are datasets we used in our study that do have isolated points.
- f-10. **Percentage of end points:** A node which has a degree of one is defined as an end point. The ratio of the number of end points to the total number of nodes in the entire graph is selected as a feature.

- f-11. **Number of nodes:** Total number of nodes in the graph.
- f-12. **Number of edges:** Total number of edges in the graph.
- f-13. **Spectral radius:** The spectral radius is defined as the largest magnitude eigenvalue of the adjacency matrix of the graph. More formally, let $|\lambda_1| > |\lambda_2| > \dots > |\lambda_s|$ be the distinct eigenvalues of the adjacency matrix A of the graph, sorted by their magnitude. The spectral radius of the graph, $\rho(G)$, is defined as: $\rho(G) = |\lambda_1|$.
- f-14. **Second largest eigenvalue:** The value of the second largest eigenvalue of the adjacency matrix A , i.e., $|\lambda_2|$.
- f-15. **Trace:** Sum of the eigenvalues of the adjacency matrix, i.e., $\sum_i \lambda_i$. This is in fact equivalent to the trace of the adjacency matrix A , i.e., $Tr(A) = \sum_{i=1}^n a_{ii}$. This feature is useful only if the graph has several loops, i.e., an edge joining a vertex to itself. For a simple graph, which is loop-free, the trace equals 0 because the elements on the main diagonal of A are all zeros.
- f-16. **Energy:** Squared sum of the eigenvalues of the adjacency matrix A . More formally, the energy of a graph G is: $E(G) = \sum_i \lambda_i^2$.
- f-17. **Number of eigenvalues:** Number of distinct eigenvalues, $s \leq n$, of the adjacency matrix A of the graph. The adjacency matrix A of an undirected graph has n eigenvalues, however, they are not necessarily distinct.
- f-18. **Label Entropy:** We employ label entropy to measure the uncertainty of labels. Suppose a graph G has q different labels: l_1, \dots, l_q , then the label entropy is given as: $H(G) = -\sum_{i=1}^q p(l_i) \log p(l_i)$.
- f-19. **Neighborhood Impurity:** We define the impurity degree of a node u as:

$$ImpurityDeg(u) = |L(v) : v \in N(u), L(u) \neq L(v)| \quad (4.1)$$

where $L(u)$ is the label, and $N(u)$ is the neighborhood of (the nodes adjacent to) node u . If all nodes in the neighborhood of u have the same node label, the

impurity degree is 0. For the whole graph, we are only interested in the nodes that have impurity degree larger than 0, i.e., the nodes which have at least one neighbor whose label is different. The neighborhood impurity of a graph G is defined as the average impurity degree over nodes with positive impurity.

- f-20. **Link Impurity:** An edge (u, v) is defined to be impure if $L(u) \neq L(v)$. The link impurity of a graph G is defined as: $\frac{|(u,v) \in E: L(u) \neq L(v)|}{m}$, where m is the number of total edges in graph G .

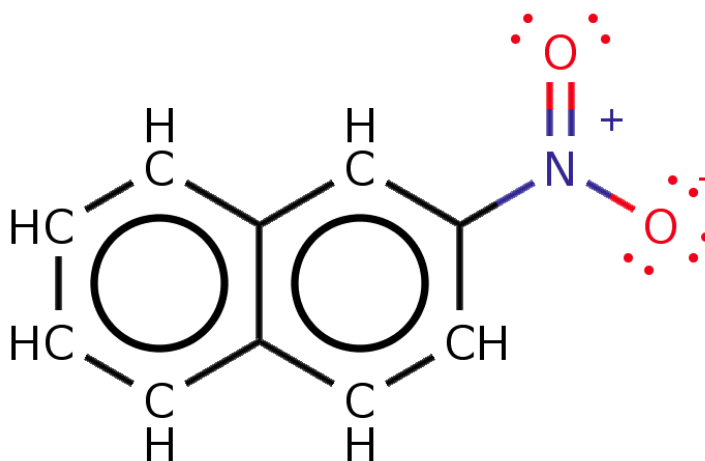


Figure 4.2: A chemical compound from PTC dataset (with implicit hydrogens)

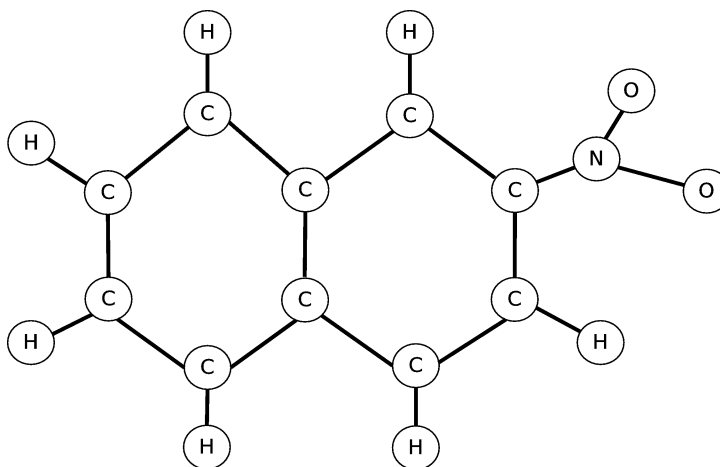


Figure 4.3: The graph representation for Figure 4.2 (labels on node/atoms: O:Oxygen, H:Hydrogens, N:Nitrogen, C:Carbon. Labels on edges/bonds: A:Aromatic, S:Single, D:Double)

Example: Figure 4.2 illustrates an example from PTC chemical compound dataset (see Section 4.3.2 for description). Figure 4.3 is a graph representation for the molecule in Figure 4.2. Nodes/edges are assigned a label based on their types, properties, etc. Table 4.1 gives the graph feature vector based on the 20 global graph attributes that are listed in the order given above. For instance, there are 20 nodes in the graph, with 9 nodes having degree 1, and 11 nodes having degree 3. The average degree (f_1) is therefore $\bar{d} = \frac{9 \times 1 + 11 \times 3}{20} = \frac{42}{20} = 2.1$. Since there are no triangles in the graph the clustering coefficient (f_2) is 0. As another example, the graph has $m = 21$ edges, but there are 11 impure links ($L(u) \neq L(v)$), thus the link impurity is given as $f_{20} = 10/21 = 0.48$. The other features can be computed based on their definition.■

Graph Classification: In computing the feature values, if a certain graph in the dataset is disconnected and contains several components, we compute the average value for a given graph metric over all the components. Each graph G_i in the database is finally represented by its corresponding feature vector F_i over the 20 topological and label attributes. However, using the raw or unnormalized feature values does not perform well. This is mainly because the original feature have different range of values (see Table 4.1 for example), which would give more importance to features with larger values than those with smaller values. Instead we normalize the feature values via range and z-normalization.

In range normalization each value x of a feature f_i is transformed into $r(x) = \frac{x - \min}{\max - \min}$, where min and max denote the minimum and maximum value for f_i . In z-normalization x is replaced by $z\text{-score}(x) = \frac{x - \mu}{\sigma}$, where μ and σ are the mean and standard deviation for f_i . By normalizing the values all features are considered on equal footing, which helps improve the classification accuracy. Once each graph G_i in the dataset \mathcal{D} has been transformed into its corresponding normalized feature-vector of length 20, $F_i = (f_{i1}, \dots, f_{i20})$, we use a SVM classifier over the new feature-vector dataset, using the Gaussian or radial basis (RBF) kernel (we tried a linear kernel too, but RBF gave better results).

Computational Complexity: The various graph attributes range from the simple to the complex, with higher computational times for the more complex features. In the analysis below, we use n to denote the number of nodes $|V|$ (also called the graph order), and m to denote the number of edges $|E|$ (also called the graph size).

For each graph in the database, the number of nodes (f_{11}) and edges (f_{12}) are already known, so their cost is $O(1)$. If they are not known before-hand, we can compute them in one pass over the entire graph in time $O(n+m)$. The degree based attributes like the percentage of isolated or end nodes (f_9, f_{10}), and average degree (f_1) can be computed in time linear in the graph order and size, i.e., in $O(n+m)$ time. The giant connected component (f_8) can also be found via breadth-/depth-first search in $O(n+m)$ time.

The clustering coefficient (f_2) can be computed in time $O(nd_{\max}^2)$, where d_{\max} is the maximum degree for the graph. A better approximation is to use the average degree $d = \frac{2m}{n}$. To compute the clustering coefficient for each node takes on average $O(d^2) = \frac{2m^2}{n^2}$. The time to compute the average clustering coefficient over all nodes is then $O(\frac{m^2}{n})$.

The eccentricity based attributes (f_3, f_4, f_5, f_7) and the average path length (f_6) can be easily computed from the all-pairs shortest path matrix. The all-pairs matrix can be computed in time $O(n^2 + nm)$ via n calls of single-source shortest paths, each of which can be computed in breadth-/depth-first search in time $O(n+m)$, since we assume that each edge has weight one. From the shortest path matrix, the attributes can be computed in $O(n^2)$ time.

The spectral attributes (f_{13} to f_{17}) depend on the eigen-decomposition of G , which can be computed in $O(n^3)$ time in the worst case. However, typically real-world graphs are sparse, which can be exploited to reduce the complexity to $O(n^2)$ [105]. Also note that when the input graphs are very large, we compute only the top $k \geq 2$ eigenvalues. For sparse graphs, the top k eigenvalues can be computed in $O(mkt + nk^2t + k^3t)$ time (e.g., using the Implicitly Restarted Lanczos Method [106]), where t is the number of iterations until convergence. For sparse graphs, with $m = O(n)$, if $k \ll n$, the time reduces to $O(nt)$. The trace (f_{15}) and energy (f_{16}) are computed only over these k eigenvalues. The number of eigenvalues

(f_{17}) is not very informative in this case.

Finally, label entropy (f_{18}) can be computed in $O(n)$, neighborhood impurity (f_{19}) can be computed in $O(nd_{\max})$ and link impurity (f_{20}) can be computed in $O(n + m)$ time.

4.3 Experiments

4.3.1 Experimental Setup

We compare our graph feature based classification approach with state-of-the-art graph kernel classifiers. More specifically, we compare with the following graph kernel methods: fast geometric Random-walk (RW) kernel [91], Shortest-path (SP) kernel [49], Graphlet (GK) kernel [54], Ramon-Gärtner (RG) subtree kernel [51], and Weisfeiler-Lehman (WL) subtree kernel [53]. We relied on a Matlab implementation of all of these kernels¹. As suggested in [53], we used the tuned parameter settings for each of the kernels as follows. For RW, the decay weight is chosen in the range $\lambda \in \{10^{-6}, \dots, 10^{-2}\}$. For RG we set $\lambda_r = \lambda_s = 1$. For the WL kernel, we choose $h = \{1, \dots, 10\}$, which means that 10 different kernel matrices are computed. For SP, we use equal length shortest paths, and for GK we use connected 3-minors. We also compare with the subgraph-feature based approach of CORK [56]², which is implemented in C++. For CORK, we use 10% minimum support to mine the frequent discriminative subgraphs. Recall that CORK uses the efficient gSpan [29] frequent graph mining algorithm, and then does feature selection. We also compared with a direct approach that uses all mined frequent subgraphs as binary features for classification; we used the Gaston [107] subgraph mining method for this.

In the discussion below, our graph feature approach is denoted as GF. GF was written in Python, with NumPy [108] and Networkx [109] modules for linear algebra and graph support. Note that both NumPy and Matlab use low-level C implementations for most matrix operations, therefore, the timing results are comparable³, though there might be slight differences. We present results for three variants of the GF approach: GF(no) denotes the method using a raw or unnormalized feature

¹obtained from Prof. K. Borgwardt and N. Shervashide

²obtained from Marisa Thoma

³For performance comparison of NumPy and Matlab, see [110] for instance.

vector, whereas GF(r) and GF(z) use range and z-score normalized feature vectors, respectively.

We use the libsvm [111] (Support Vector Machine library) for all of the kernels and our method. The graph kernels use the kernel matrix computed via the particular graph kernel, whereas we use the default Gaussian or radial basis (RBF) kernel in libsvm: $\kappa(G_i, G_j) = \exp\{-\gamma\|F_i - F_j\|^2\}$, where $\gamma = \frac{1}{p}$, where p equals the number of features, which is 20 for all datasets, except those that do not have any labels on the nodes and edges. The latter include the cell-graph datasets, and the non-label versions of the other datasets; for these the number of features is $n' = 17$, since we do not use the label-based features (f_{18} to f_{20}). We also use a RBF kernel for CORK, since that gave us the best results.

For each method, we perform 10 runs of 10-fold cross-validation, and we tune the C parameter, for C-SVM, using only the training folds. We will report the graph kernel matrix computation (for other methods), or for the graph feature computation (for GF). We will plot SVM training times for each method for some selected datasets as well. All the experiments were performed on MAC OS X 10.5 with two 2.66GHz Dual Core Intel Xeon processors, with 4GB 667MHz DDR2 memory.

For performance assessment, we report the average accuracy and standard deviation over the 10-fold cross-validation run 10 times. We also assess whether the accuracy of our method is significantly better or worse than the accuracy of the best previous method. For this we use the paired t -test for the difference between the accuracies in each fold. We report the value of the t -statistic, given as: $t = \frac{\sqrt{N}\mu}{\sigma}$, where μ and σ^2 denote the mean and variance of the difference in accuracy between the two methods, and $N = 100$ is the total sample size (10 runs times 10 folds). We fail to reject the null hypothesis, that there is no significant difference, if $t \in (-t_{\alpha/2, N-1}, t_{\alpha/2, N-1})$, where α is the significance level for a two-tailed t -test with $N - 1$ degrees of freedom. We use $\alpha = 0.05$ for which the interval is $(-1.98, 1.98)$. If the value of t -statistic is outside this interval, the two methods are statistically significantly different in performance.

Table 4.2: Benchmark Datasets

(a) Chemical Compound Datasets

| dataset | size (N) | classes | positive | negative | avg. $ V $ | avg. $ E $ | Max. $ V $ | Max. $ E $ | avg. deg |
|---------|--------------|---------|----------|----------|------------|------------|------------|------------|----------|
| MUTAG | 188 | 2 | 125 | 63 | 17.7 | 38.9 | 28 | 33 | 2.19 |
| NCI1 | 4110 | 2 | 2057 | 2053 | 29.9 | 32.3 | 111 | 119 | 2.16 |
| NCI109 | 4127 | 2 | 2079 | 2048 | 29.7 | 32.1 | 111 | 119 | 2.16 |
| PTC(MM) | 336 | 2 | 129 | 207 | 25.0 | 25.4 | 109 | 108 | 1.98 |
| PTC(FM) | 349 | 2 | 143 | 206 | 25.2 | 25.6 | 109 | 108 | 1.99 |
| PTC(MR) | 344 | 2 | 152 | 192 | 25.6 | 26.0 | 109 | 108 | 1.99 |
| PTC(FR) | 351 | 2 | 121 | 230 | 26.1 | 26.5 | 109 | 108 | 1.99 |

(b) Protein Datasets

| class | size (N) | classes | positive | negative | avg. $ V $ | avg. $ E $ | Max. $ V $ | Max. $ E $ | avg. deg |
|-------|--------------|---------|----------|----------|------------|------------|------------|------------|----------|
| D & D | 1178 | 2 | 691 | 487 | 284.3 | 715.7 | 5748 | 14267 | 4.98 |
| CATH1 | 712 | 2 | 384 | 328 | 205.7 | 819.8 | 568 | 2356 | 7.79 |
| CATH2 | 190 | 2 | 109 | 81 | 308.0 | 1254.8 | 568 | 2220 | 8.14 |

(c) Cell-Graph Datasets: E: Breast; O: Bone; A: Brain

| tissue | class | size (N) | avg. $ V $ | avg. $ E $ | Max. $ V $ | Max. $ E $ | avg. deg |
|--------|-------------------|--------------|------------|------------|------------|------------|----------|
| Breast | Invasive (EI) | 202 | 966.9 | 12503.6 | 1956 | 51454 | 22.07 |
| | Non-invasive (EN) | 93 | 889.7 | 13459.4 | 1940 | 48750 | 25.47 |
| | Benign (EB) | 151 | 829.4 | 15677.7 | 1885 | 49165 | 34.72 |
| Bone | Ost(OO) | 49 | 532.2 | 2324.7 | 2855 | 18790 | 5.42 |
| | Frac(OF) | 39 | 497.7 | 1599.2 | 1913 | 13564 | 4.25 |
| | Normal(ON) | 20 | 174.2 | 1174.0 | 612 | 7309 | 8.14 |
| Brain | Glioma(AG) | 329 | 4550.2 | 43400.5 | 7311 | 98572 | 18.04 |
| | Inflammation(AI) | 107 | 4244.1 | 39457.7 | 7113 | 90029 | 17.06 |
| | Benign(AB) | 210 | 789.0 | 3988.9 | 1755 | 9309 | 9.65 |

4.3.2 Datasets

We used three different types of graph datasets: chemical compounds, proteins, and cell graphs. See Table 4.2 for statistics on the different graphs. The table shows the total size of each dataset, including the number of classes, the number of points in each class, the average and maximum number of vertices and edges for the graphs, and the average degree. Note that the chemical compounds and protein datasets have two classes, whose sizes are shown under the positive and negative column labels. The cell-graphs datasets have three classes, and their sizes are shown under the “size(N)” column.

Chemical Compounds: The chemical compound datasets include MUTAG [112], NCI1 and NCI109 [113], and PTC [114], which have been employed as benchmark datasets in previous graph kernel papers. MUTAG is a dataset of mutagenic aromatic and heteroaromatic nitro compounds assayed for mutagenicity on bacterium *Salmonella typhimurium*. We used two balanced subsets of the NCI (National Cancer Institute) datasets. The class labels are based on an anti-cancer screen, as active or inactive. The PTC (The Predictive Toxicology Challenge) datasets record the carcinogenicity of several hundred chemical compounds for Male Rats (MR), Female Rats (FR), Male Mice (MM) and Female Mice (FM). As one can in Table 4.2(a), these graphs are very small (20-30 nodes, and 25-40 edges) and sparse, with average degree around 2.

Proteins: The D&D dataset [115], which has also been used by previous studies, consists of 1178 proteins, with 691 enzymes and 487 non-enzymes. In addition, we created two new datasets from CATH [116], a manually curated database of protein domain structures. CATH1 consists of proteins in the same class (Mixed Alpha-Beta), but having different architectures (Alpha-Beta Barrel vs. 2-layer Sandwich). CATH2 has proteins in the same class (Mixed Alpha-Beta), architecture (Alpha-Beta Barrel), and topology (TIM Barrel), but in different homology classes (Aldolase vs. Glycosidases). CATH2 is harder to classify, since proteins in the same topology class are structurally similar. The protein graphs are 10 times larger in size than chemical compounds, with 200-300 nodes and 700-1250 edges (Table 4.2(b)), and

average degree is 5-8. We use another variant of the CATH datasets, without the node labels (which correspond to the amino acids). We denote these as CATH1 (w/o L) and CATH2 (w/o L).

Cell-graphs: We also performed experiments on cancer Cell-Graph datasets [103, 117, 118]. These graphs were constructed from the histo-pathological samples from three different types of tissues: breast, bone and brain. Within each type of tissue we consider three classes: healthy (Normal/Benign), cancerous (Invasive, Osteosarcoma:Ost, Glioma), and damaged (Non-invasive, Fracture:frac, Inflammation). We perform binary classification for each pair of classes within a tissue type. Usually, it is much easier to distinguish between normal and cancerous classes, but harder to classify cancerous versus damaged classes, for each tissue type. For example, for the breast tissue, classifying EB vs. EN and EB vs. EI is easier than classifying EI vs. EB. In contrast to the chemical compounds and proteins, the cell graphs are even larger and denser (see Table 4.2(c)). Average graph size is 5-10 times larger than proteins, with 200-4500 nodes and 100-43000 edges. Average degree is 4-8 for bone, 10-18 for brain, and 22-35 for breast tissue. The cell-graphs are unlabeled (i.e., no labels on nodes or edges).

4.3.3 Graph Kernel Comparison

4.3.3.1 Chemical Compound Datasets

Table 4.3 shows the accuracy comparison for our GF approach versus other graph kernels on the chemical compound datasets. Each cell records the average classification accuracy, as well as the standard deviation, over the 10 fold cross-validation over 10 different runs. Table 4.4 reports the wall clock running times for each method on the different datasets. A ‘-’ in any cell means that the computation of the kernel matrix did not finish in one day, and thus the run was aborted. We can observe from the results that graph features with unnormalized/raw values, denoted GF(no), do not perform well in terms of accuracy. Furthermore, for these graphs, the z-normalized GF(z) approach delivers the best results, except for PTC(FM) and PTC(FR).

On the MUTAG dataset, GF(z), is the best overall method. Looking at the

Table 4.3: Accuracy (\pm Standard Deviation): Chemical Compound Datasets (bold t -statistic means statistically significant)

| | MUTAG | NCI1 | NCI109 | PTC(MM) | PTC(FM) | PTC(MR) | PTC(FR) |
|----------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|
| GF(no.) | 86.75 \pm 6.38 | 67.18 \pm 2.99 | 66.30 \pm 1.67 | 60.99 \pm 4.05 | 55.90 \pm 6.40 | 51.15 \pm 5.87 | 58.12 \pm 6.52 |
| GF(r) | 87.11 \pm 8.59 | 69.64 \pm 1.69 | 69.44 \pm 2.26 | 62.78 \pm 6.83 | 63.90 \pm 3.97 | 57.52 \pm 6.63 | 66.71 \pm 6.47 |
| GF(z) | 91.37 \pm 4.77 | 75.62 \pm 2.05 | 74.22 \pm 1.66 | 63.38 \pm 5.43 | 59.87 \pm 6.13 | 63.94 \pm 7.05 | 62.96 \pm 6.65 |
| RW | 84.01 \pm 6.61 | – | – | 60.58 \pm 8.92 | 58.98 \pm 9.72 | 51.40 \pm 5.77 | 64.63 \pm 8.74 |
| SP | 88.13 \pm 7.15 | 73.82 \pm 1.61 | 72.89 \pm 2.17 | 57.52 \pm 9.98 | 52.41 \pm 9.79 | 58.46 \pm 6.08 | 63.67 \pm 5.27 |
| GK | 83.93 \pm 6.48 | 69.18 \pm 2.62 | 69.82 \pm 1.89 | 58.04 \pm 8.22 | 55.86 \pm 8.95 | 55.19 \pm 5.66 | 59.41 \pm 5.36 |
| RG | 86.23 \pm 4.41 | – | – | 64.30 \pm 7.89 | 58.45 \pm 7.01 | 57.61 \pm 8.32 | 63.52 \pm 6.40 |
| WL | 86.89 \pm 6.33 | 84.11 \pm 1.61 | 83.50 \pm 2.34 | 67.23 \pm 5.87 | 64.37 \pm 6.57 | 58.10 \pm 7.18 | 65.22 \pm 5.34 |
| CORK | 86.19 \pm 7.82 | 78.12 \pm 1.61 | 77.76 \pm 1.48 | 61.85 \pm 8.04 | 57.90 \pm 6.53 | 60.75 \pm 7.31 | 65.51 \pm 9.82 |
| t -statistic | 4.02 | -28.61 | -29.27 | -3.91 | -0.88 | 3.26 | 1.40 |

Table 4.4: Running Times on Chemical Compound Datasets (h:hours, m:minutes, s:seconds)

| | MUTAG | NCI1 | NCI109 | PTC(MM) | PTC(FM) | PTC(MR) | PTC(FR) |
|------|--------------|---------------|---------------|--------------|--------------|--------------|--------------|
| GF | 0.78s | 36.48s | 36.77s | 2.30s | 2.56s | 2.66s | 2.50s |
| RW | 5m3s | – | – | 2h3m42s | 2h16m11s | 2h12m7s | 2h17m28s |
| SP | 4.61s | 16m56s | 21m2s | 35.82s | 35.79s | 36.14s | 37.72s |
| GK | 1.42s | 3m21s | 3m25s | 4.88s | 5.05s | 5.04s | 5.22s |
| RG | 42m54s | – | – | 2h11m1s | 2h16m54s | 2h14m17s | 2h20m6s |
| WL | 5.88s | 15m30s | 16m1s | 14.86s | 16.22s | 15.51s | 16.10s |
| CORK | 1m1s | 33m19s | 35m44s | 19.92s | 23.90s | 23.03s | 27.04s |

Table 4.5: Accuracy (\pm Standard Deviation): Chemical Compound Datasets Without Labels

| | MUTAG w/o L | NCI1 w/o L | NCI109 w/o L | PTC(MM) w/o L | PTC(FM) w/o L | PTC(MR) w/o L | PTC(FR) w/o L |
|---------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|
| GF(no.) | 82.43 \pm 7.51 | 65.72 \pm 2.37 | 64.14 \pm 1.83 | 55.03 \pm 9.95 | 51.85 \pm 6.46 | 51.99 \pm 7.89 | 59.24 \pm 6.87 |
| GF(r) | 87.28 \pm 6.46 | 67.25 \pm 2.25 | 66.63 \pm 1.50 | 62.44 \pm 8.37 | 63.29 \pm 7.27 | 56.45 \pm 8.44 | 66.70 \pm 6.02 |
| GF(z) | 87.78 \pm 6.35 | 71.36 \pm 2.21 | 71.53 \pm 1.33 | 63.39 \pm 10.36 | 61.29 \pm 4.87 | 63.42 \pm 8.55 | 62.15 \pm 7.68 |
| GK | 87.39 \pm 6.96 | 62.75 \pm 2.16 | 62.03 \pm 2.50 | 61.02 \pm 6.38 | 59.84 \pm 8.66 | 56.61 \pm 9.01 | 65.80 \pm 6.17 |
| WL | 86.75 \pm 5.82 | 78.69 \pm 3.33 | 77.03 \pm 5.47 | 66.99 \pm 7.62 | 64.73 \pm 9.63 | 57.01 \pm 7.83 | 62.10 \pm 7.40 |
| CORK | 86.78 \pm 7.84 | – | – | – | – | – | – |

t -statistic, it is significantly better than the next best method, SP kernel, at a significance level of $\alpha = 0.05$, since $t \notin (-1.98, 1.98)$. Considering the time, we can see that GF takes only a fraction of the time compared to other methods. It is 6 times faster than SP.

On the NCI1 and NCI109 datasets, the WL subtree kernel has the best accuracy. The NCI datasets are quite tree-like; we can see in Table 4.2 that for both the average and maximum number of edges, we have $|E| \approx |V|$. Given that the WL kernel is a subtree kernel, it is well suited for such tree-like datasets. However, in terms of time, GF is over 25 times faster.

The PTC datasets are also tree-like, and thus the WL kernel performs well. However, while WL is the best method for MM, GF(z) is the best method for MR. These differences are statistically significant. On FM, the WL kernel has a slight advantage, whereas on FR data, GF(r) is slightly better, though there is no significant difference between them. In terms of computational time, GF method is vastly superior, being 6 times faster than WL. GK is the second fastest method, but its accuracy is not very high. Compared to RW, SP, RG, and CORK, our GF approach is one to three orders of magnitude faster.

To study the effect of graph topology versus graph attributes/labels, we also compared the performance of the GF method versus the GK, WL and CORK methods on unlabeled versions of the chemical compound datasets, as shown in Table 4.5. Unfortunately, CORK could not run on the unlabeled datasets (except for MUTAG) since too many frequent topological subgraphs are mined once labels are omitted, and the (sub)graph isomorphism checks become expensive. The main observation from these experiments is that GF’s performance remains essentially the same between the labeled and unlabeled datasets, which indicates that the label information is not that helpful for GF; it relies primarily on the topological features. Also, WL does suffer when labels are omitted, especially for the NCI datasets. This suggests that for the NCI datasets labels are important to improve performance, and thus GF may also benefit if we can incorporate more effective label features (which is part of future work).

4.3.3.2 Protein Datasets

The protein graphs are much larger compared to the chemical compound datasets. The accuracy and timing results are shown in Tables 4.6 and 4.7. In terms of accuracy, among the GF variants, the unnormalized version is the worst, whereas GF(r) gives the best results, except on the unlabeled D&D (w/o L) and CATH2(w/o L) datasets.

Table 4.6: Accuracy (\pm Standard Deviation): (a) Protein Datasets, (b) Protein Datasets without Labels. (bold values indicate best performance, and bold t -statistic means statistically significant)

(a) Original Datasets

| | D&D | CATH1 | CATH2 |
|----------------|------------------------------------|------------------------------------|------------------------------------|
| GF(no) | 62.99 \pm 4.49 | 83.29 \pm 4.98 | 61.58 \pm 10.27 |
| GF(r) | 76.32 \pm 2.72 | 99.02 \pm 0.90 | 81.57 \pm 5.39 |
| GF(z) | 75.95 \pm 2.66 | 98.46 \pm 1.32 | 79.27 \pm 9.46 |
| RW | – | – | – |
| SP | – | 98.88 \pm 1.37 | 96.32 \pm 3.37 |
| GK | 75.13 \pm 2.71 | 98.32 \pm 0.84 | 94.74 \pm 4.71 |
| RG | – | – | – |
| WL | 78.29 \pm 3.05 | 98.59 \pm 1.09 | 94.21 \pm 4.97 |
| CORK | 71.22 \pm 4.56 | 94.24 \pm 2.77 | 97.89 \pm 2.58 |
| t -statistic | -3.75 | 0.07 | -26.75 |

(b) Datasets without Node/Edge Labels

| | D&D (w/o L) | CATH1(w/o L) | CATH2(w/o L) |
|----------------|------------------------------------|------------------------------------|------------------------------------|
| GF(no) | 62.48 \pm 3.35 | 82.86 \pm 5.43 | 61.05 \pm 8.87 |
| GF(r) | 76.06 \pm 3.31 | 98.60 \pm 1.54 | 77.89 \pm 7.74 |
| GF(z) | 77.51 \pm 5.08 | 97.90 \pm 1.57 | 81.05 \pm 3.49 |
| RW | – | – | – |
| SP | – | 97.89 \pm 1.70 | 76.32 \pm 8.57 |
| GK | 69.27 \pm 4.78 | 97.61 \pm 2.52 | 66.84 \pm 11.05 |
| RG | – | – | – |
| WL | 74.19 \pm 2.60 | 98.59 \pm 1.26 | 76.84 \pm 8.22 |
| CORK | – | – | – |
| t -statistic | 4.74 | 0.15 | 4.67 |

For the D&D enzyme dataset, the only kernel methods that finished within 24 hours, were GF, GK, WL and CORK. Here WL has the best accuracy, but GF is about 3 times faster. Even though CORK is 4 times faster, since it uses C++ and

Table 4.7: Running Times on Protein Datasets (h:hours, m:minutes, s:seconds)

| | D&D | CATH1 | CATH2 |
|------|---------------|--------------|--------------|
| GF | 52m35s | 4m36s | 2m15s |
| RW | – | – | – |
| SP | – | 1h42m12s | 42m26s |
| GK | 23h14m53s | 37m8s | 15m54s |
| RG | – | – | – |
| WL | 2h12m57s | 22m33s | 4m45s |
| CORK | 14m10s | 41m28s | 43m6s |

GF uses python, the timing comparison is not entirely fair. Also, it is worth noting that on the unlabeled versions of the protein datasets, CORK could not be run.

For the CATH datasets, we can see that GF is 2-5 times faster than WL, about 20 times faster than SP, 4-8 times faster than GK, and 10-20 times faster than CORK. RW, and RG were not able to finish in the allotted time (1 day). For CATH1, we find that GF(r) has a slight (though not significant) advantage over other approaches in terms of accuracy. CATH1 is an easier dataset to classify, since the proteins in the two classes differ at the architecture level, and thus are structurally different. All the methods do well on this dataset. On CATH2 data, CORK gave the best overall results, and GF was significantly worse. CATH2 is a much harder dataset to classify from a structural viewpoint. This is because the two classes have significant structural similarity. On the other hand, there are possibly significant differences in the protein sequences between the two classes. GF mainly relies on topological graph features, and the three label features (f_{18} to f_{20}) are not able to capture the subsequence similarity (since they are local and are not designed to look at subsequences). GF is thus not able to distinguish between the two classes as well as the other kernels that can effectively utilize label information. To verify this hypothesis, we removed the node (amino acid) labels from the CATH datasets, thus all methods have to rely only on topological information. We also include a comparison the unlabeled D&D dataset for completeness. CORK aborted on all the unlabeled proteins datasets, since the (sub)-graph isomorphism checks in this case become too expensive. We now find that GF(z) is significantly superior to other methods on CATH2(w/o L), and also on the D&D dataset. While the accuracy

of $\text{GF}(z)$ remains close to the labeled case, the other methods suffer a significant drop in accuracy. For example, WL has an accuracy of 94.21 on CATH2, but only 76.85 on CATH2(w/o L). This confirms two things: i) there is significant sequence similarity between sequences in the same class, exploiting which helps the other methods, and ii) even though the CATH2 classes are topologically similar, there are enough structural differences that GF is still able to exploit. Likewise GF can leverage the topological differences between the enzymes and non-enzymes for D&D.

4.3.3.3 Cell-Graph Datasets

Table 4.8 shows the accuracy comparison for GF versus other graph kernels on the cell-graph datasets. The corresponding timing results are shown in Table 4.9. One of the differences between cell-graph and the other datasets is that cell-graphs are much larger (e.g., 4550 nodes and 43400 edges for brain graphs). Furthermore, while the other datasets are labeled, the cell-graph data does not have any labels.

We show the results separately for each tissue type, and we show results for binary classification. Among the GF variants, $\text{GF}(z)$ is usually better than $\text{GF}(r)$, or is close to it. The unnormalized version is significantly worse. For GF, we use only the top $k = 2$ eigenvalues for Bone(O), and $k = 100$ for Brain(A). This is because computing all the eigenvalues for these large graphs is expensive.

For the cell-graph datasets, GF significantly outperforms all other methods in terms of accuracy, and also has an advantage in terms of time. For the largest graphs, from brain tissues, even GK and WL were not able to complete within a day of computation time. We also do not report the accuracies for CORK, since it aborted on the cell-graphs datasets. Due to the large graph size, and the lack of labels, the graph mining step in CORK fails to enumerate any discriminative subgraphs.

The accuracy of the competing methods depends on the two classes being compared. WL has comparable accuracy only on the easier to classify pairs, namely benign and cancerous breast graphs (EB vs. EI), normal and fractured bone graphs (OF vs. ON), and benign and inflamed brain graphs (AB vs. AI). On the other hand, on the other hard pairs, such as inflamed/noninvasive/damaged versus cancer

Table 4.8: Accuracy (\pm Standard Deviation) on Cell-Graph Datasets – E: Breast, O: Bone, and A: Brain. (bold t -statistic means statistically significant)

| | EB vs EI | EN vs EB | EN vs EI | OF vs ON | ON vs OO | OF vs OO |
|----------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|-------------------------------------|
| GF(no) | 57.78 \pm 7.25 | 61.92 \pm 5.21 | 64.48 \pm 4.96 | 65.67 \pm 20.76 | 71.19 \pm 14.06 | 58.83 \pm 16.59 |
| GF(r) | 87.70 \pm 5.30 | 86.35 \pm 7.02 | 82.76 \pm 5.35 | 98.33 \pm 5.00 | 94.29 \pm 7.00 | 53.47 \pm 22.08 |
| GF(z) | 88.05 \pm 4.27 | 84.58 \pm 6.96 | 83.84 \pm 4.41 | 97.67 \pm 6.67 | 92.86 \pm 7.14 | 63.75 \pm 14.09 |
| RW | – | – | – | 90.00 \pm 11.06 | 76.43 \pm 14.30 | 49.72 \pm 14.28 |
| SP | 76.27 \pm 5.16 | 77.61 \pm 6.29 | 73.22 \pm 6.14 | 94.67 \pm 8.19 | 78.33 \pm 14.57 | 60.67 \pm 12.19 |
| GK | 66.01 \pm 9.57 | 75.19 \pm 8.13 | 62.38 \pm 6.91 | 56.00 \pm 21.12 | 60.71 \pm 13.27 | 51.39 \pm 13.57 |
| RG | – | – | – | 72.33 \pm 12.52 | 67.04 \pm 15.54 | 48.87 \pm 14.07 |
| WL | 87.23 \pm 4.57 | 74.81 \pm 6.09 | 71.22 \pm 8.15 | 98.33 \pm 5.00 | 63.57 \pm 17.63 | 61.25 \pm 13.60 |
| t -statistic | 0.23 | 9.31 | 12.74 | 0 | 12.28 | 1.29 |

| | AG vs AI | AG vs AB | AB vs AI |
|----------------|------------------------------------|------------------------------------|------------------------------------|
| GF(no) | 75.36 \pm 5.06 | 61.04 \pm 3.98 | 66.20 \pm 8.36 |
| GF(r) | 88.28 \pm 3.40 | 98.70 \pm 1.45 | 99.06 \pm 2.81 |
| GF(z) | 87.91 \pm 2.86 | 99.26 \pm 0.91 | 98.74 \pm 2.87 |
| RW | – | – | – |
| SP | – | – | – |
| GK | – | – | 97.79 \pm 2.01 |
| RG | – | – | – |
| WL | – | – | 99.38 \pm 1.88 |
| t -statistic | – | – | -0.14 |

Table 4.9: Running Times on Cell-Graph Datasets (h:hours, m:minutes, s:seconds)

| | EB vs EI | EN vs EB | EN vs EI | OF vs ON | ON vs OO | OF vs OO |
|----|-----------------|---------------|-----------------|--------------|---------------|--------------|
| GF | 1h11m13s | 47m11s | 1h15m40s | 23.7s | 1m26s | 1m43s |
| RW | – | – | – | 11m16s | 18m14s | 40m8s |
| SP | 8h21m36s | 5h44m23s | 9h40m4s | 19m8s | 1h11m58s | 1h27m2s |
| GK | 14h12m4s | 9h51m37s | 11h46m28s | 5m33s | 9h47s | 10m47s |
| RG | – | – | – | 24.15s | 43.70s | 1m29s |
| WL | 1h55m42s | 44m29s | 1h23m35s | 55.30s | 2m56s | 2m32s |

| | AG vs AI | AG vs AB | AB vs AI |
|----|-----------------|------------------|-----------------|
| GF | 17h50m5s | 13h53m38s | 5h15m44s |
| RW | – | – | – |
| SP | – | – | – |
| GK | – | – | 9h11m12s |
| RG | – | – | – |
| WL | – | – | 7h24m29s |

(EN vs. EI, OF vs. OO, and AI vs. AG), our graph feature approach is significantly superior. For example, on EN vs. EI, GF(z) has an accuracy of 83.84, whereas SP has an accuracy of 73.22, WL has an accuracy of 71.22 and GK has an even lower value (62.38).

It is interesting to note that since cell-graph datasets do not have labels, all the kernels can only use structural information for computing the kernel matrix. The fact that GF has the best accuracies implies that the topological attributes we compute are well-suited to extract discriminating features among the graphs from different classes. In fact, these attributes are much better at capturing the topological differences than the corresponding kernels based on subtrees, graphlets, shortest paths, and random walks, especially on large, unlabeled graphs, such as the cell graphs.

4.3.3.4 SVM Training Times

Figure 4.4 plots SVM training times for the different methods on four selected datasets: MUTAG, PTC(MR), NCI1 and CATH1. They all use the same libsvm package. For MUTAG, the training times for GF and CORK are about 5 times faster than other graph kernel methods, which are all relatively close to each other. For PTC(MR), there are significant differences in the training times. GF still has

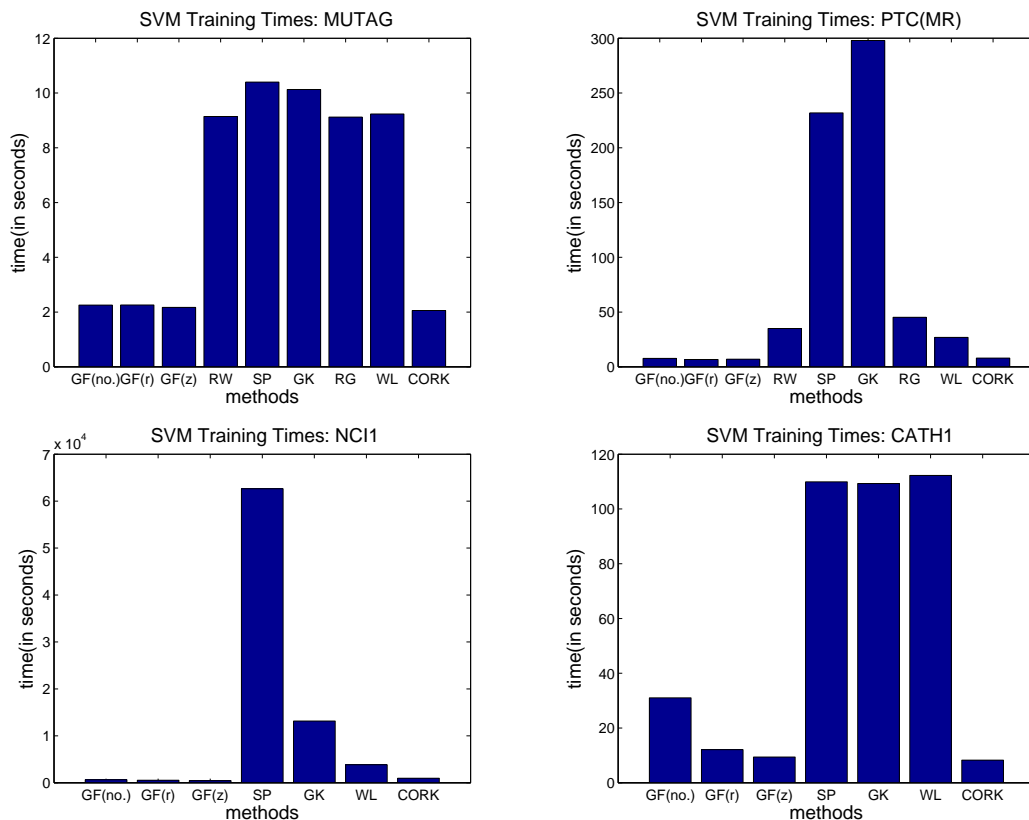


Figure 4.4: SVM Training Times

the least computational time, while the Graphlet kernel had the longest time. GF is over 4 times faster than the WL kernel. For NCI1, since the computation of RW and RG kernel matrix did not finish within a day, we excluded them from the plot. GF retains the fastest SVM training speed. Finally on the CATH1 dataset, GF(z) and CORK are the fastest, whereas the WL kernel is the slowest, being over 10 times slower than GF(z). Figure 4.4 demonstrates that GF methods with normalization give rise to easily optimized SVM classifiers compared to other graph kernel methods.

4.3.3.5 Scalability Study

To study the scalability of our GF approach, we selected some datasets from each of the three groups: NCI1, PTC(FR), CATH2, and OF-OO. Next, we replicate the instances 10, 50 and 100 times, and we compare with the original replication factor of 1. For instance, NCI1 dataset with replication factor 100 has $100 \times 4110 =$

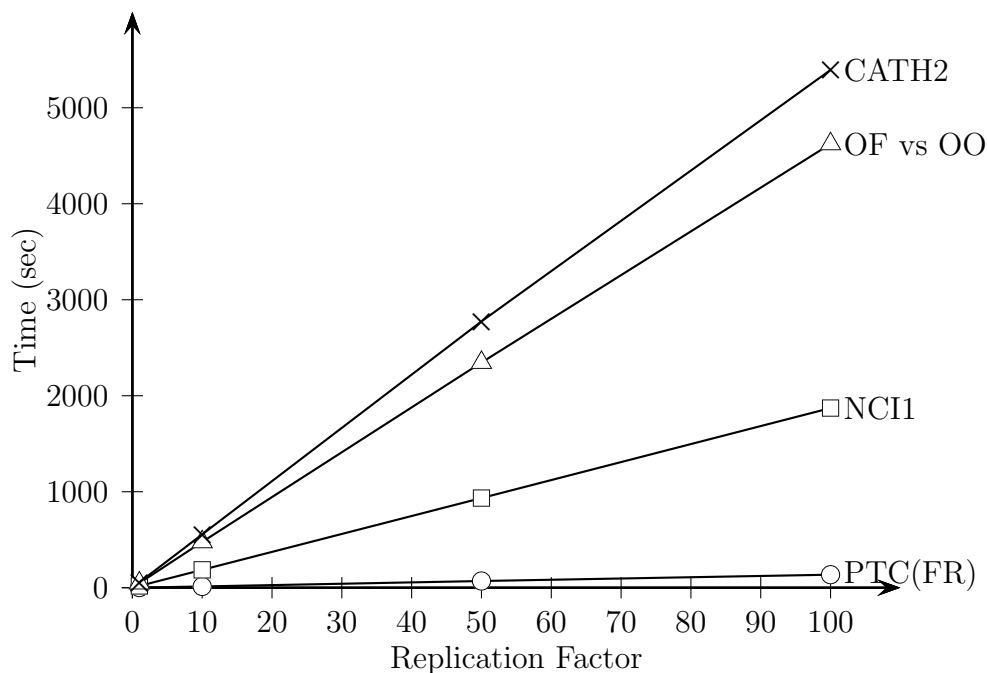


Figure 4.5: Scalability Study: GF(z) Time for various Replication Factors

411,000 instances. Figure 4.5 shows the time it takes to extract the GF topological features for the four selected datasets. As expected, the runtime of GF is linear in the number of graph instances.

4.3.4 Frequent Subgraph Features

We evaluated the effectiveness of using frequent subgraph features for graph classification. Recall that the CORK method first uses the gSpan algorithm [29] to mine frequent subgraphs, and then uses feature selection to create the final classifier. CORK’s performance on chemical compound datasets appears in Table 4.3. We also employed the Gaston [107] frequent subgraph miner to obtain all frequent subgraphs at 10% minimum support (same as that used for CORK/gSpan). Next, we convert each subgraph into a binary feature, noting its presence or absence in each training/testing example, without doing any feature selection. Table 4.12 shows the accuracy of this approach on the chemical compound datasets. We find that except for PTC(FR), this approach is not as good as the WL or GF methods. Interestingly, the Gaston approach is typically better than CORK, even though it does not do any feature selection. It also is the fastest among all approaches in terms of time;

its efficient C++ implementation finishes in under 2s for all the chemical compound datasets (though it is not fair to compare it with the Matlab/Python implementations). Unfortunately, Gaston was not able to mine all the frequent subgraphs for the protein and cell-graph datasets (at 10% minimum support threshold) within the 24 hour time threshold. For some higher support values (e.g., 30%) Gaston was able to run, but it output so many patterns that the SVM classification step failed (it exceeded the 4GB memory). Combined with the fact that even CORK cannot be run on the cell-graph and other unlabeled datasets, the whereas subgraph features can help for labeled graphs, they are not effective for unlabeled graphs, where GF is particularly strong.

4.3.5 Feature Importance Study

We carried out a detailed study to rank the different graph features, with the goal to identify which features carry the most information. We used the SVM-wrapper feature selection method [119], which ranks the features via recursive feature elimination using SVM. In Table 4.10 and 4.11 we record the ranks of the top- k effective features, for $k = 5$. The last row of the table notes the number of occurrences of each feature in the top five list. While it is clear that the most informative graph attributes are dataset dependent, some general conclusions can still be made. For instance, for GF with range normalization, the top-5 features (shown in bold in Table 4.10) based on the number of occurrences include: (1) average clustering coefficient (f_2), (2) number of nodes (f_{11}), (3) number of eigenvalues (f_{17}), (4) number of edges (f_{12}), and (5) energy (f_{16}). For GF with z-normalization, the top-5 occurrences based features (shown in bold in Table 4.11) are: (1) number of nodes (f_{11}), (2) average degree (f_1), (3) average clustering coefficient (f_2), (4) number of edges (f_{12}), and (5) number of eigenvalues (f_{17}). Thus good discriminating features depend on both the graph datasets and different normalization methods. We still observe some high level trends in the rankings from the tables. The spectral attributes (f_{13} , f_{14} , f_{16} , f_{17}) are generally quite effective. The shortest path based attributes (f_3 , f_4 , f_5 , f_6 , f_7) do not appear to be that effective compared to the spectral attributes, especially for range normalization. f_1 (average degree) and f_2

Table 4.10: Feature rankings – range normalization

| F | f_1 | f_2 | f_3 | f_4 | f_5 | f_6 | f_7 | f_8 | f_9 | f_{10} | f_{11} | f_{12} | f_{13} | f_{14} | f_{15} | f_{16} | f_{17} | f_{18} | f_{19} | f_{20} |
|---------------|----------|-----------|----------|----------|----------|----------|----------|----------|----------|----------|-----------|----------|----------|----------|----------|----------|-----------|----------|----------|----------|
| MUTAG | 4 | | | | 3 | | | | | | | 2 | 1 | | | 5 | | | | |
| NCI1 | | | | | | | | 5 | 2 | | | | 4 | | | | 1 | | | 3 |
| NCI109 | | | | | | | | | 2 | | | | 4 | 5 | | | 1 | | | 3 |
| PTC(MM) | 4 | 1 | | 3 | | 2 | | | | 5 | | | | | | | | | | |
| PTC(FM) | | 1 | | | | 3 | 5 | | | | | 4 | | 2 | | | | | | |
| PTC(MR) | | 1 | | | | | 3 | | | | | | 4 | 2 | | | | | 5 | |
| PTC(FR) | | 1 | | | | | | | | 5 | 2 | 3 | | | | 4 | | | | |
| D&D | | | | | | 2 | | | | 4 | | | | 3 | | | 5 | 1 | | |
| CATH1 | | 5 | | | | | | | | | 4 | 1 | | | | 2 | 3 | | | |
| CATH2 | | | | | | | | | | | | 4 | | | | 2 | 5 | 1 | | 3 |
| CATH1(w/o L) | | 5 | | | | | | | | | 4 | 1 | | | | 2 | 3 | – | – | – |
| CATH2(w/o L) | | | | | | | | | | 4 | 2 | 5 | | | | 3 | 1 | – | – | – |
| EB vs EI | 1 | | 5 | | | | | | | | 4 | | | 3 | | | 2 | – | – | – |
| EN vs EB | 3 | 1 | | | | | 4 | | | | 5 | | | | | | 2 | – | – | – |
| EN vs EI | | 3 | 5 | | | | | | | | 2 | | 1 | 4 | | | | – | – | – |
| OF vs ON | | 1 | | | | | | 3 | | 5 | 2 | | | | | | 4 | – | – | – |
| ON vs OO | | 1 | | | | | | | 4 | 5 | 2 | | | | | | 3 | – | – | – |
| OF vs OO | 3 | | | | | 4 | 2 | 1 | | | 5 | | | | | | | – | – | – |
| AG vs AI | 5 | 2 | | | | | | | 1 | | 3 | | | | | 4 | | – | – | – |
| AG vs AB | | 4 | 1 | | | 3 | | 5 | | | 2 | | | | | | | – | – | – |
| AB vs AI | | 4 | 1 | | | 5 | | 2 | | | 3 | | | | | | | – | – | – |
| counts | 6 | 13 | 4 | 1 | 1 | 6 | 4 | 5 | 4 | 6 | 13 | 7 | 5 | 6 | 0 | 7 | 11 | 3 | 2 | 1 |

Table 4.11: Feature rankings – z normalization

| F | f_1 | f_2 | f_3 | f_4 | f_5 | f_6 | f_7 | f_8 | f_9 | f_{10} | f_{11} | f_{12} | f_{13} | f_{14} | f_{15} | f_{16} | f_{17} | f_{18} | f_{19} | f_{20} |
|---------------|-----------|-----------|-------|-------|-------|-------|-------|-------|-------|----------|-----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| MUTAG | 3 | | | | 2 | | | | | | | | 1 | | | | | 4 | | 5 |
| NCI1 | | | | | | | | 3 | | | 2 | | | | | | 1 | 4 | | 5 |
| NCI109 | | | | 3 | | | | | | 2 | 4 | | | | | | 1 | | 5 | |
| PTC(MM) | | 1 | 5 | | | 4 | | | | | 3 | 2 | | | | | | | | |
| PTC(FM) | | 5 | 2 | | | 1 | 4 | | | | | | | | | | 3 | | | |
| PTC(MR) | 2 | | | | | | | | | | | | 4 | 1 | | | | 5 | | 3 |
| PTC(FR) | | 1 | | | | | | | | 3 | 4 | | | | | 5 | 2 | | | |
| D&D | 3 | | | | 5 | | | | | | 4 | 1 | | 2 | | | | | | |
| CATH1 | | | | | | | | | | | 2 | 3 | | | | 4 | 1 | 5 | | |
| CATH2 | | | 4 | 3 | | 1 | | | | | 5 | | | | | | | 2 | | |
| CATH1(w/o L) | | | | 5 | | | | | | | 2 | 3 | | | | 4 | 1 | – | – | – |
| CATH2(w/o L) | | | 5 | 2 | | 1 | | | | 4 | 3 | | | | | | | – | – | – |
| EB vs EI | 1 | | | | | | | | | | 2 | 3 | | | | 4 | 5 | – | – | – |
| EN vs EB | 1 | 3 | | | | | | | | | | | 4 | 2 | | | 5 | – | – | – |
| EN vs EI | 1 | 5 | | | | | | | | | 2 | 3 | 4 | | | | | – | – | – |
| OF vs ON | 3 | 1 | | | | | | 5 | | | 2 | | | 4 | | | | – | – | – |
| ON vs OO | 3 | 1 | | 5 | | | | | | | 2 | | | 4 | | | | – | – | – |
| OF vs OO | 4 | | 5 | | | | 2 | 1 | | | | 3 | | | | | | – | – | – |
| AG vs AI | 5 | 1 | | | | | | 3 | | | 2 | | | 4 | | | | – | – | – |
| AG vs AB | 4 | 3 | | | | | 2 | | 5 | | 1 | | | | | | | – | – | – |
| AB vs AI | 4 | 2 | 1 | | | | | | 3 | | 5 | | | | | | | – | – | – |
| counts | 12 | 10 | 6 | 5 | 2 | 4 | 3 | 4 | 2 | 2 | 16 | 8 | 4 | 6 | 0 | 4 | 8 | 5 | 1 | 3 |

(average clustering coefficient) are also generally good features. f_8 (giant connected ratio), f_9 (percentage of isolated points), f_{10} (percentage of end points) might be effective if the graphs in the dataset contain several components, e.g., bone tissues dataset. As for the label-based features, for half of the datasets (5 out of 10) in which graphs have node labels, the feature f_{18} (label entropy) is in the top 5. Note that none of them select f_{15} (trace) as an effective discriminating feature, since none of the graphs in our experiments contain loops. A detailed dataset specific analysis of feature importance is given next.

Chemical Compounds Datasets: We first take a look at range normalization in Table 4.10, for the chemical compound datasets: MUTAG, NCI1 and NCI109, and PTC.

For MUTAG, the top 5 features are f_{13} (spectral radius), f_{12} (number of edges), f_5 (effective radius), f_1 (average degree), f_{16} (energy).

The NCI datasets all have similar top 5 feature rankings among themselves, i.e., the first 4 feature are the same, f_{17} (number of eigenvalues), f_9 (percentage of isolated points), f_{19} (neighborhood impurity), and f_{13} (spectral radius), except for the 5th feature: for NCI1 it is f_8 (giant connected ratio), and for NCI109 it is f_{14} (second largest eigenvalue). Note that both range and z-normalization choose f_{17} (number of eigenvalues) as the top feature for these tree-like datasets. There is rich literature showing that the eigenvalues of a graph capture many topological properties. For example, multiplicity of eigenvalues usually corresponds to symmetries in the graph [120] (although the correspondence is not exact). The possible explanation for f_{17} being selected by the NCI graphs is that one class has more balanced tree-like structures than the other class.

However, the PTC graphs have quite different feature rankings with each other. It is interesting to see that all of them put f_2 , average clustering coefficient, as the 1st rank. But since PTCs are chemical compounds, most graphs have zero triangles. Thus most graphs will have f_2 feature value 0. Using only average clustering coefficient as the feature for classification, the accuracies for GF(r) are: PTCMM (61.68 ± 6.83), PTCFM (59.02 ± 6.05), PTCMR (55.83 ± 7.83), PTCFR ($65.53 \pm$

9.32), which are close to the accuracies by using the full 20 features for classification (see Table 4.3). One possible explanation is that since positive or negative graphs have f_2 value 0, a SVM classifier will not be able to discriminate between them. Rather, the SVM will classify all of them as either positive or negative. Since the PTC datasets are balanced datasets to some extent (with approximately 40% positives and 60% negatives), the classifier accuracies are expected to around 60%. The SVM-wrapper feature selection method also shows that f_2 is an informative graph attribute. Other attributes are less informative or even redundant. Nevertheless, considering them together still improves the accuracies by 1% – 4%, compared to using f_2 alone (see Table 4.3).

For the z-normalization results in Table 4.11, we see that for MUTAG, the top 5 features are f_{13} (spectral radius), f_2 (average clustering coefficient), f_1 (average degree), f_{18} (label entropy), f_{20} (link impurity). For the two NCI graphs, there are two features selected by both: f_{11} (number of nodes), f_{17} (Number of eigenvalues). For the four PTC graphs, f_2 (average clustering coefficient) is selected by three datasets. Note that f_5 (effective radius), f_8 (giant connected ratio), f_9 (percentage of isolated points), f_{15} (trace), and f_{19} (neighborhood Impurity) are not selected either by range or z-normalization. Since chemical compound usually contain only one component and form a simple graph, some features such as isolated points, giant connected ratio and trace have little discriminating power.

Proteins: The protein datasets include D&D, and CATH1 and CATH2 (with and without labels). In Table 4.10, these five datasets all put feature f_{17} (number of eigenvalues) in the top 5, whereas four of them have f_{16} (energy) and f_{12} (number of edges) in the top 5, and three choose f_{11} (number of nodes). Compared to chemical compounds, protein datasets (graphs) are relatively large. Thus some simple statistics, e.g., number of nodes and number of edges might be more effective and can have more discriminating power. The dimension of the adjacency matrix for each graph is larger compared to chemical compound datasets, which increases the value of the energy feature.

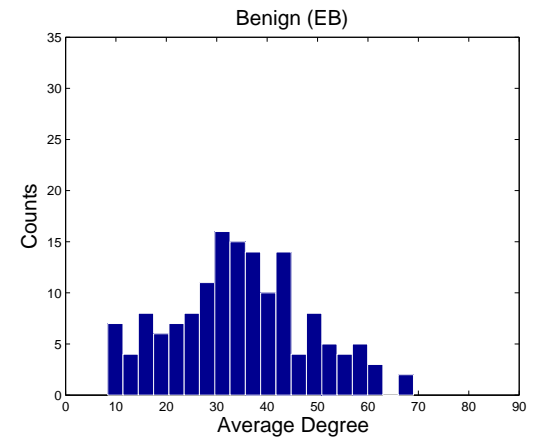
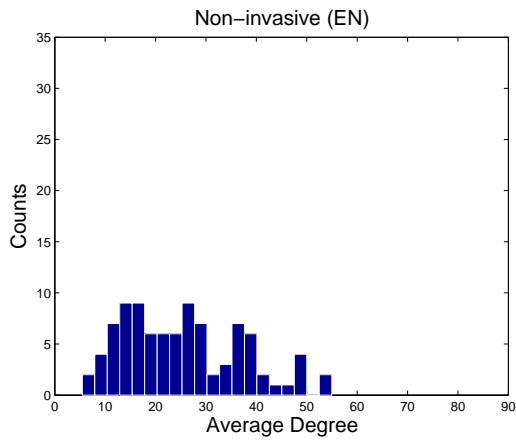
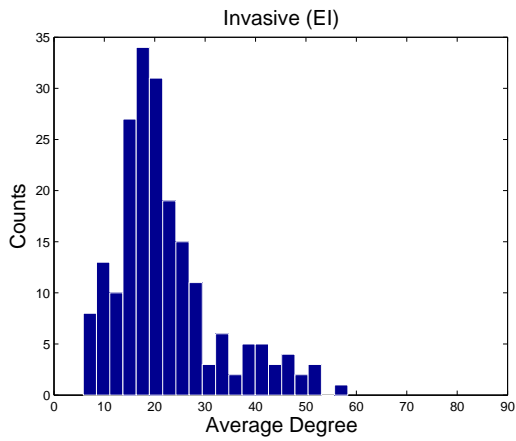
In Table 4.11, for z-normalization, besides all of the protein datasets choosing

f_{11} (number of nodes) as an important feature, three in five consider f_4 (effective diameter) and f_{12} (number of edges) important for classification. For D&D dataset, both the range and z-normalization methods select f_{14} (the second largest eigenvalue).

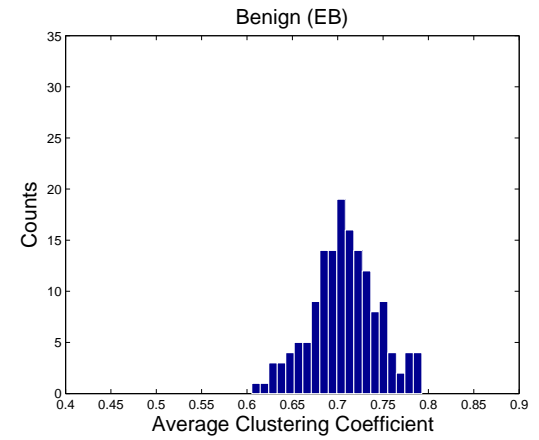
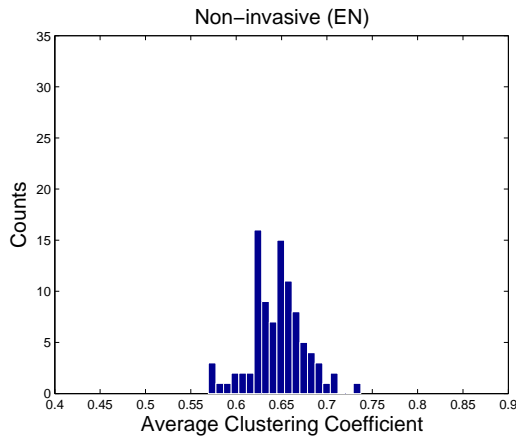
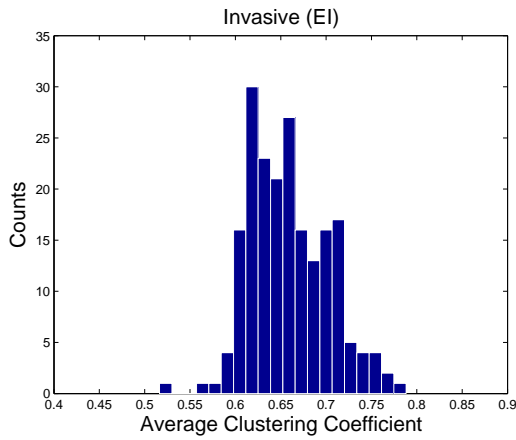
Cell-graphs: Cell-Graph are the largest graphs used in our experiments. The datasets have three different types of cancerous tissues: Breast, Bone and Brain. Each has three binary classification tasks. With range normalization (see Table 4.10) all of them consider f_{11} (number of nodes) and nearly all of them (7 in 9) put f_2 (average clustering coefficient) as important discriminating features.

For Breast-cancer datasets, besides the features mentioned above, two in three select f_1 (average degree), f_3 (average effective eccentricity), f_{14} (second largest eigenvalue), and f_{17} (number of eigenvalues) among the top 5 features. For Bone-cancer datasets, two in three select f_8 (giant connected ratio), f_{10} (percentage of end points), f_{17} (number of eigenvalues) in the top 5. Note that each graph in bone has multiple connected components and the largest number of components of one graph is 158 (including isolated points). Thus f_8 and f_{10} become important discriminating features here. For Brain-cancer datasets, besides the features mentioned above (f_{11} , f_2), two in three select f_3 (average effective eccentricity), f_6 (closeness centrality), and f_8 (giant connected ratio). It is known that different types of tissues in brain have different cellular density levels [118]. The cancerous tissues (Glioma) have higher cellular density while the healthy tissues (Benign) have lower cellular density. At the same time, cancerous tissues and damaged tissues (Inflammation) have equally high cellular density. Hence, on AG vs. AB and AB vs. AI, except for the node-based features (f_{11} and f_2), some path-based features such as f_3 (average effective eccentricity) and f_6 (closeness centrality) also show discriminating power.

For z-normalization (Table 4.11), all of the Cell-Graph datasets choose f_1 (average degree) and almost all of them (7 in 9) choose f_2 (average clustering coefficient) and f_{11} (number of nodes), as important features. Figure 4.6 plots the frequency distributions for the average degree (f_1) and average clustering coefficient (f_2) for the three breast cancer cell-graph classes (invasive: EI, non-invasive: EN,



(a) Average Degree Distribution



(b) Clustering Coefficient Distribution

Figure 4.6: Breast Cell-Graphs: Class Specific Average Degree (a) and Clustering Coefficient Distribution (b)

and benign: EB). One can observe that there are significant differences among the class-specific distributions, which help discriminate them in our GF approach.

It is worth remarking that while f_{11} (number of nodes) and f_{12} (number of edges) are good discriminating features (they are usually in the top five), the accuracy is not significantly different even without these features, as shown in Table 4.13. The table shows the performance of GF with and without these two features: GF is with the original set of 17 features (discounting the label-based features $f_{18} - f_{20}$, since Cell-graphs are unlabeled), whereas GF* is with 15 features (with f_{11} and f_{12} removed). We can see that the differences in accuracies are not significant, except for ON vs. OO, where the drop in accuracy is approximately 6%. Note that in over half the cases there is even a slight increase in accuracy for GF*, with the reduced set of features. For OF vs. OO there is a 3.5% increase for GF*. One possible explanation is that features are not mutually independent and some missing features could be compensated to some extent by other features. In any case, even without the two simplest features, namely number of nodes and edges, GF* method is superior to other graph kernels on the Cell-Graph datasets (see Table 4.8).

4.3.6 Augmented Topological Features

To further examine the effect of additional topological features, we augmented the initial 17 features (f_1 to f_{17} , mentioned in Section 4.2), with an additional 10 global graph features, for a total of 30 global features, including the three label-based features (f_{18} to f_{20}). These augmented features are described below.

- f-21. **Eigen-exponent:** It is defined as the slope of the best-fitting line for the decreasing eigen-values, i.e., λ_i versus i , in a log-log plot.
- f-22. **Hop-plot exponent:** The hop plot value reflects the size of the neighborhood between any two nodes. Let $R(h)$ denote the number of node pairs that are reachable within h hops. The hop-plot exponent is the slope of best-fitting line in a log-log plot of the number of reachable pairs $R(h)$ as a function of h .
- f-23. **Averaged current-flow closeness centrality:** A variant of closeness centrality based on effective resistance between nodes in a network [121].

Table 4.12: Frequent Subgraphs as Binary Features

| | MUTAG | NCI1 | NCI109 | PTC(MM) | PTC(FM) | PTC(MR) | PTC(FR) |
|--------|------------------|------------------|------------------|------------------|------------------|------------------|------------------------------------|
| GASTON | 81.87 \pm 5.58 | 80.92 \pm 1.45 | 81.00 \pm 1.37 | 61.35 \pm 6.91 | 59.30 \pm 9.24 | 63.14 \pm 8.69 | 68.64 \pm 8.90 |

Table 4.13: Accuracy (\pm Standard Deviation) on Cell-Graph Datasets (E: Breast, O: Bone, and A: Brain. GF*: features f_{11} , f_{12} removed. The best results for GF and GF* are shown in bold)

| | EB vs EI | EN vs EB | EN vs EI | OF vs ON | ON vs OO | OF vs OO |
|---------|------------------------------------|------------------------------------|------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|
| GF(no) | 57.78 \pm 7.25 | 61.92 \pm 5.21 | 64.48 \pm 4.96 | 65.67 \pm 20.76 | 71.19 \pm 14.06 | 58.83 \pm 16.59 |
| GF(r) | 87.70 \pm 5.30 | 86.35 \pm 7.02 | 82.76 \pm 5.35 | 98.33 \pm 5.00 | 94.29 \pm 7.00 | 53.47 \pm 22.08 |
| GF(z) | 88.05 \pm 4.27 | 84.58 \pm 6.96 | 83.84 \pm 4.41 | 97.67 \pm 6.67 | 92.86 \pm 7.14 | 63.75 \pm 14.09 |
| GF*(no) | 57.79 \pm 4.92 | 61.90 \pm 7.02 | 68.40 \pm 7.36 | 62.67 \pm 12.45 | 66.43 \pm 15.00 | 55.33 \pm 16.38 |
| GF*(r) | 86.99 \pm 6.60 | 86.87 \pm 8.14 | 84.40 \pm 5.31 | 95.00 \pm 7.64 | 87.14 \pm 11.87 | 56.81 \pm 16.30 |
| GF*(z) | 86.41 \pm 5.19 | 81.97 \pm 5.48 | 84.10 \pm 6.75 | 95.00 \pm 10.67 | 88.57 \pm 12.45 | 67.22 \pm 15.17 |

| | AG vs AI | AG vs AB | AB vs AI |
|---------|------------------------------------|------------------------------------|------------------------------------|
| GF(no) | 75.36 \pm 5.06 | 61.04 \pm 3.98 | 66.20 \pm 8.36 |
| GF(r) | 88.28 \pm 3.40 | 98.70 \pm 1.45 | 99.06 \pm 2.81 |
| GF(z) | 87.91 \pm 2.86 | 99.26 \pm 0.91 | 98.74 \pm 2.87 |
| GF*(no) | 75.02 \pm 4.55 | 81.63 \pm 4.10 | 67.56 \pm 10.67 |
| GF*(r) | 87.83 \pm 4.76 | 98.70 \pm 1.86 | 99.05 \pm 1.45 |
| GF*(z) | 86.93 \pm 4.42 | 99.44 \pm 0.85 | 99.38 \pm 1.25 |

- f-24. **Degree assortativity coefficient:** Assortativity measures the similarity of connections in the graph with respect to the node degree. It is essentially the same as the Pearson correlation coefficient of degrees between pairs of adjacent nodes [122].
- f-25. **Number of cliques:** This is the number of maximal cliques in each graph.
- f-26. **Average neighbor degree:** First, we compute the average degree of the neighborhood of each node. Then, for the whole graph we take the average of these values over all nodes.
- f-27. **Transitivity:** Defined as the fraction of all possible triangles in each graph. A possible triangle is a triple which has two edges sharing one vertex.
- f-28. **Periphery:** The periphery is the set of nodes with eccentricity equal to the effective diameter. We compute the fraction of nodes that comprise the periphery in each graph.
- f-29. **Cycle basis:** A graph may have multiple cycles. However, each cycle in the graph can be decomposed into the sum (defined as “exclusive or” of the edges) of several cycles. We call a minimal collection of such cycles as a cycle basis. The cardinality of cycle basis is selected as a global feature.
- f-30. **Square clustering coefficient:** While clustering coefficient by triangles give the likelihood that any two neighbors of u are connected, the square clustering coefficient gives the probability that two neighbors of node v share a common neighbor different from v .

Table 4.14 and Table 4.15 give the accuracy and running time, respectively, for the GF methods, with and without the augmented features. GF(20) denotes the use of the original 20 features, and GF(30) includes the augmented features. The results are shown on selected datasets from each group, i.e., MUTAG and PTC(MM) from chemical compounds, D&D and CATH2(w/o L) from proteins, and EN vs. EI from cell graphs. We observe that the augmented features improve the performance for CATH2 (w/o L) to some extent, with not much difference for the other datasets.

Table 4.14: Accuracy (\pm Standard Deviation) on selected Datasets

| | MUTAG | PTC(MM) | D&D | CATH2(w/o L) | EN vs EI |
|-----------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|
| GF(r)(20) | 87.11 \pm 8.59 | 62.78 \pm 6.83 | 76.32 \pm 2.72 | 77.89 \pm 7.74 | 87.70 \pm 5.30 |
| GF(z)(20) | 91.37\pm4.77 | 63.38\pm5.43 | 75.95 \pm 2.66 | 81.05 \pm 3.49 | 88.05\pm4.27 |
| GF(r)(30) | 87.22 \pm 9.20 | 63.37 \pm 5.11 | 76.74\pm2.60 | 80.00 \pm 5.67 | 80.38 \pm 5.76 |
| GF(z)(30) | 89.91 \pm 5.48 | 61.60 \pm 5.87 | 75.97 \pm 3.42 | 83.16\pm7.74 | 82.74 \pm 3.40 |

Table 4.15: Running Times on selected Datasets (h:hours, m:minutes, s:seconds)

| | MUTAG | PTC(MM) | D&D | CATH2 (w/o L) | EN vs EI |
|--------|-------|---------|----------|---------------|----------|
| GF(20) | 0.78s | 2.30s | 52m35s | 2m15s | 1h15m40s |
| GF(30) | 3.04s | 8.52s | 1h26m35s | 6m18s | 3h24m59s |

However, interestingly, on the cell-graph dataset EN-EI, the use of the additional features significantly reduces the accuracy. This may be because the other features already capture most of the global topological properties for the cell-graph dataset. Also, these augmented features can be expensive to compute, being typically 3 times slower than the 20 initial features. Combined with the feature importance study above, we conclude that the simpler topological features are typically sufficient to capture most of the important structural properties, and it is not that beneficial to include too many complex topological attributes likes cliques, square clustering, cycles, and so on.

4.4 Conclusions

We propose a simple yet effective and efficient graph classification approach that is based on topological and label graph attributes. The graph dataset is converted into a feature-vector dataset, which can be classified easily using any classifier. Our main idea is that graphs from the same class should have similar attribute values. Based on an extensive comparison with state-of-the-art graph kernel classifiers, we show that our approach yields competitive or better accuracies, and has typically much lower computational times. Our conclusion is that graph attributes are effective in capturing discriminating structural information from different classes. While no method is uniformly the best, our approach is particularly effective for unlabeled graphs. Combining our graph features with the best features from other

approaches, such as the WL kernel, has the potential to yield even better methods, especially for labeled graphs.

CHAPTER 5

FUTURE WORK

In this chapter, we will outline our research work in the near future. The first research direction is improving the efficiency of MCMC, which was formerly successfully applied to frequent subgraph mining and minimal DNF expressions mining (see Chapter 2 and Chapter 3). This issue may be tackled by implementing the approach on multi-core processors, as well as utilizing graphics computing units (GPUs). The second direction is to develop more effective graph kernels by combining our topological features approach with subgraph sampling methods. For very large graphs we may also combine input space sampling techniques to reduce the graph size. The third direction is to use sampling techniques for informative subspaces which contain good alternative clustering structures. Despite various studies on alternative clustering problem in general, currently there is no sampling algorithm that has been proposed. The search space for alternative/multiview clusterings grows very sharply as the data gets large because of the combinatorial explosion. Thus, it is particularly interesting if we can apply sampling techniques in this domain. We discuss some of these ideas in more detail below.

5.1 Parallel MCMC Sampling

The first research direction is improving the efficiency of MCMC, which has been successfully applied to frequent subgraph mining and minimal DNF expressions mining.

One way to tackle this issue is by implementing the approach on multi-core processors, as well as utilizing graphics computing units (GPUs). A multi-core processor is a single computing component that has at least two independent CPUs. GPUs are massively multi-threaded multi-core processors. Though utilizing multi-

This chapter is to appear in: G. Li, S. Günnemann and M. Zaki, “Stochastic Subspace Search for Top-K Multi-View Clustering,” In *Proceedings of the 4th International Workshop on Discovering, Summarizing and using Multiple clusterings (MultiClust) held in conjunction with KDD*, Chicago, IL, 2013.

core processors or GPUs can improve the efficiency, the design of flow control is very challenging. For example, a well known problem in MCMC is that its convergence speed is very slow. Thus, an important question is how to use multi-core processors or GPUs to implement a more efficient MCMC sampling strategy than the current state-of-the-art single process/thread algorithm? Of course, MCMC is inherently parallelizable, i.e., we can enforce n independent walks simultaneously. However, a new question then arises: could we further improve the efficiency based on that? Besides, in Chapter 3, we use two different strategies: random walks with jumps and restarts, to avoid being trapped in local regions of the graph which consists of non-minimal DNFs. Otherwise, our algorithm will not output minimal DNFs even after a long run. Can we utilize the inherent characteristics of multi-core processors or GPUs to improve these two strategies? For example, for random walks with jumps, can we enforce n threads so that each process/thread manages a random jump independently to an earlier minimal DNF generator in the history as its new start? Besides that, can we design parallel discriminative sampling of minDNFs so that each process/thread is responsible for collecting high contrast features from one class, which would be used to build effective classifier later? These are all interesting questions that we want to tackle in the near future.

5.2 Subspace Sampling for Multiview Clusterings

We have some promising preliminary results on sampling subspaces for multiview clusterings. Below we first give the motivation and then the definition of multiview clustering. Next we discuss our initial work in detail.

5.2.1 Introduction

Clustering is the method of organizing a set of patterns (represented by vectors in multidimensional space) into groups (called clusters). The patterns in the same group (cluster) should have more similar properties than the patterns from different groups. Note that a clustering solution is a grouping of the patterns.

Traditional clustering techniques work with only a single mixture model and each cluster is assumed to correspond to a single component's distribution. Take

clustering on movie data for example [123]. The result of applying traditional clustering method will output a single view to the user, e.g., the view ‘genre’. Some movies are grouped into ‘adventure’ category, some movies are grouped into ‘horror’ category and some movies are grouped into ‘drama’ category, etc. However, in many cases a more complicated clustering structure, where multiple views that can reveal different perspectives to the audience, may exist in the data. For example, the movie data can not only be grouped by the ‘genre’, but also based on ‘director’, ‘location’, ‘cast’ or other characteristics. That is, different views are expected in the movie data. Since the data usually is not collected for just one sole purpose, the multi-view hypothesis is reasonable for various databases. Although multiple meaningful views may exist in the data, a desirable property is that the views should have as less redundancy as possible. That is, we want to find alternative views that capture new information about the structure of the data, leading to knowledge enhancement. For example, [124] introduces orthogonality constraints into the objective function for finding new clustering views to avoid similar/redundant clustering results.

5.2.2 Fundamentals

First let us briefly introduce and define some fundamental concepts below.

5.2.2.1 Subspace and Multiview Subspaces

Definition 3 *We define a subspace \mathbf{S} as a non-empty subset of set of the dimensions.*

Note that subspace \mathbf{S} defined here usually refers to axis-parallel subspace in the literature. It is not a linear combination of the dimensions. For a d -dimensional dataset, the number of k -dimensional subspaces is $\binom{d}{k}$. Thus, the number of all possible subspaces in a d -dimensional dataset is $\sum_{k=1}^d \binom{d}{k} = 2^d - 1$. If we use brute force method for enumeration, for high dimensional datasets it is computationally infeasible.

Here let us denote $|\mathbf{S}|$ as the number of dimensions or input attributes that comprise the subspace \mathbf{S} . We have:

Definition 4 Given two subspaces \mathbf{S}_1 and \mathbf{S}_2 , we denote $\mathbf{S}_1 \subset \mathbf{S}_2$ iff \mathbf{S}_1 is a subset of \mathbf{S}_2 . If $\mathbf{S}_1 \subset \mathbf{S}_2$ and $|\mathbf{S}_1| = |\mathbf{S}_2| - 1$, we say that \mathbf{S}_1 is a parent of \mathbf{S}_2 , and \mathbf{S}_2 is a child of \mathbf{S}_1 . We define the child of the one-dimensional subspace as the empty set, denoted as \emptyset .

The state space for the Markov chain for interesting subspace mining consists of subspaces linked by immediate subset-superset or parent-child relationships. Each state can be taken to be a subspace, with transitions allowed, for example, only between parent and child subspaces. Figure 5.1 shows an example of the search space for subspaces when $d = 4$. A subspace $\mathbf{S} = \{A_2, A_3\}$ has two parent subspaces $\{A_2\}$ and $\{A_3\}$ and has two children subspaces $\{A_2, A_3, A_4\}$ and $\{A_1, A_2, A_3\}$.

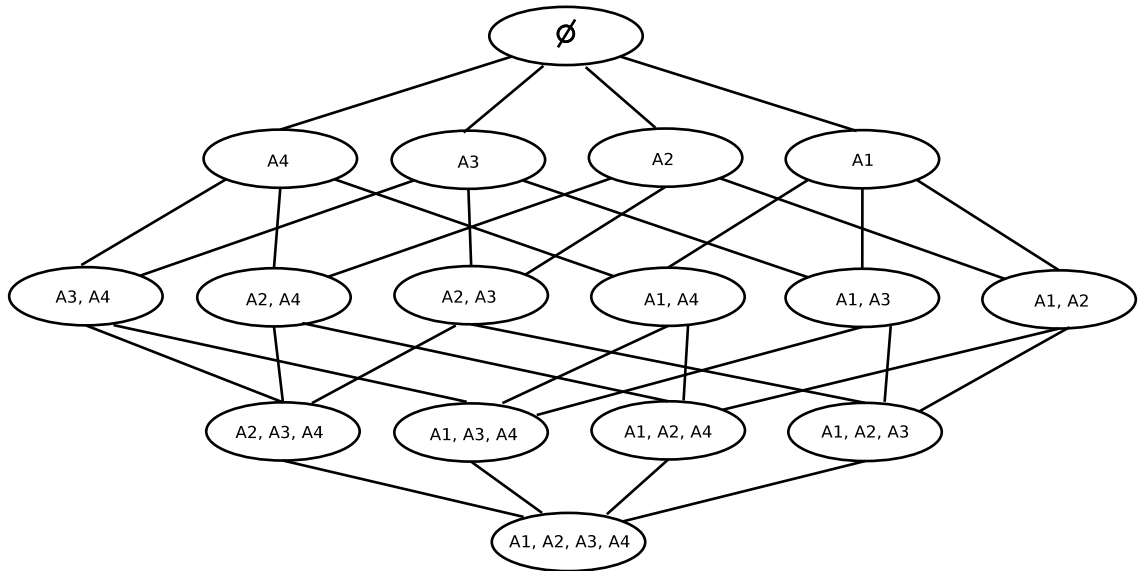


Figure 5.1: Search space for subspaces when $d = 4$

Lemma 9 Let $N(\mathbf{S})$ denote the neighbors of \mathbf{S} , i.e., the set of subspaces adjacent to \mathbf{S} in the search space. Let $|N(\mathbf{S})|$ denote the neighborhood size of \mathbf{S} . Then for any two subspaces \mathbf{S}_1 and \mathbf{S}_2 , we have $|N(\mathbf{S}_1)| = |N(\mathbf{S}_2)|$.

PROOF: It is easy to verify that for any two subspaces \mathbf{S}_1 and \mathbf{S}_2 , $|N(\mathbf{S}_1)| = |N(\mathbf{S}_2)| = d$, where d denotes the total number of dimensions in \mathcal{D} . ■

Definition 5 *The similarity of two subspaces $\mathbf{S}_1, \mathbf{S}_2$, denoted as $\text{sim}(\mathbf{S}_1, \mathbf{S}_2)$, is defined as*

$$\text{sim}(\mathbf{S}_1, \mathbf{S}_2) = \frac{|\mathbf{S}_1 \cap \mathbf{S}_2|}{|\mathbf{S}_1 \cup \mathbf{S}_2|} \quad (5.1)$$

Corollary 3 *The similarity $\text{sim}(\mathbf{S}_1, \mathbf{S}_2)$ of two subspaces $\mathbf{S}_1, \mathbf{S}_2$ satisfies:*

- a) $0 \leq \text{sim}(\mathbf{S}_1, \mathbf{S}_2) \leq 1$
- b) $\text{sim}(\mathbf{S}_1, \mathbf{S}_2) = 1$ if and only if $\mathbf{S}_1 = \mathbf{S}_2$
- c) $\text{sim}(\mathbf{S}_1, \mathbf{S}_2) = \text{sim}(\mathbf{S}_2, \mathbf{S}_1)$

Definition 6 *Given a set of already detected subspaces $\mathbf{S}^* = \{\mathbf{S}_1, \dots, \mathbf{S}_q\}$, the similarity of a subspace \mathbf{S} and the set \mathbf{S}^* is defined as:*

$$\text{sim}(\mathbf{S}, \mathbf{S}^*) = \max_{\mathbf{S}_i \in \mathbf{S}^*} (\text{sim}(\mathbf{S}, \mathbf{S}_i)) \quad (5.2)$$

Definition 7 *We say that \mathbf{S} is orthogonal to \mathbf{S}^* if $\text{sim}(\mathbf{S}, \mathbf{S}^*) \leq \varepsilon$, where ε is a user-defined orthogonal threshold.*

Definition 8 *Multiview subspace clustering is defined as the task of finding subspace clusterings with the constraint that a new subspace should be orthogonal to the already detected subspaces.*

That is, multiview clustering aims at finding subspace clusterings that have minimal subspace redundancy.

5.2.2.2 Mean Shift Clustering Algorithm

We need a basic algorithm to help evaluate the quality of the subspace. The quality of a subspace depends on how well it clusters the data. Since we want to avoid setting the number of clusters as the predefined parameter for the algorithm, we seek some nonparametric clustering technique. One good option is Mean Shift clustering algorithm [125], which does not require the prior knowledge of the number

of clusters. It is effective for detecting arbitrary shaped clusters as well. The main idea of the Mean Shift is very similar to DENCLUE 2.0 [126].

Given n data points $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ in \mathcal{R}^d , the multivariate kernel density estimator for the entire dataset is defined as:

$$f(\mathbf{x}) = \frac{1}{nh^d} \sum_{i=1}^n \kappa\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) \quad (5.3)$$

where h denotes the window radius of the kernel $\kappa(\mathbf{x})$ which we usually call the bandwidth of the kernel. Using the gradient function $\nabla f(\mathbf{x}) = 0$, we can get the local maximum values, i.e., modes of the density function.

Algorithm 1: Classical Mean Shift: **MeanShift**

Input: $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, h
Output: Clusters \mathcal{C}

- 1 $\mathcal{D}' = \mathcal{D}$ // Make a local copy;
- 2 **for** $\mathbf{x} \in \mathcal{D}'$ **do**
- 3 **while** *Conv. Cond. not satisfied* **do**
- 4 Compute the mean shift vector $\mathbf{m}_h(\mathbf{x}^t)$;
- 5 $\mathbf{x}^{t+1} = \mathbf{x}^t + \mathbf{m}_h(\mathbf{x}^t)$;
- 6 **end**
- 7 **end**
- 8 $\mathcal{C} = \mathbf{Get-Point-Membership}(\mathcal{D}')$;

Algorithm 1 shows the pseudo code for classical Mean Shift algorithm. In line 4, the mean shift vector $\mathbf{m}_h(\mathbf{x})$ is defined by [125]:

$$\mathbf{m}_h(\mathbf{x}) = \frac{\sum_{i=1}^n \mathbf{x}_i \kappa'\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right)}{\sum_{i=1}^n \kappa'\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right)} - \mathbf{x} \quad (5.4)$$

The algorithm stops when every point is shifted to its nearest mode (line 3). In line 8, the function **Get-Point-Membership** assigns each point with a cluster id according to its corresponding mode. Note that the computational complexity of Mean Shift algorithm is $O(In^2)$, where I is the average number of iterations to convergence and n is the total number of data points.

Usually we adopt Gaussian kernel $\kappa(\mathbf{x}) = \frac{1}{\sqrt{2\pi}} \exp\{-\frac{\|\mathbf{x}\|^2}{2}\}$ or Epanechnikov kernel $\kappa(\mathbf{x}) = \frac{3}{4}(1 - \|\mathbf{x}\|^2)^2$ when $\|\mathbf{x}\| < 1$.

Adaptive Bandwidth Selection: We consider adapting the kernel density estimator to the d -dimensional case:

$$\begin{aligned} f(\mathbf{x}) &= \frac{1}{nh^d} \sum_{i=1}^n \kappa\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) \\ &= \frac{1}{n} \sum_{i=1}^n \frac{1}{h^d} \kappa\left(\frac{x_1 - x_{i1}}{h}, \dots, \frac{x_d - x_{id}}{h}\right) \end{aligned} \quad (5.5)$$

κ denotes a multivariate kernel function operating on d arguments. Equation 5.5 assumes that the bandwidth h is the same for each component. In real data this may not be true. So we relax this assumption and assign a bandwidth vector $\vec{\mathbf{h}}_d = (h_{d1}, h_{d2}, \dots, h_{dd})$ in which each entry corresponds to one attribute in d -dimensional subspace. Note that $h_{di} \neq h_{d'i}$ when $d \neq d'$. In other words, the bandwidth is related to two variables, i and d . Also for simplicity, we use multiplicative kernel where $\kappa(\mathbf{x}) = K(x_1) \dots K(x_d)$ and K is a univariate kernel. It is easy to see that the Gaussian kernel qualifies for the multiplicative kernel definition. Therefore, in d -dimensional subspace, Equation 5.5 becomes:

$$\begin{aligned} f(\mathbf{x}) &= \frac{1}{n} \sum_{i=1}^n \frac{1}{h_{d1} \dots h_{dd}} \kappa\left(\frac{x_1 - x_{i1}}{h_{d1}}, \dots, \frac{x_d - x_{id}}{h_{dd}}\right) \\ &= \frac{1}{n} \sum_{i=1}^n \left\{ \prod_{j=1}^d h_{dj}^{-1} \kappa\left(\frac{x_j - x_{ij}}{h_{dj}}\right) \right\} \end{aligned} \quad (5.6)$$

We use Silverman's Rule of Thumb [127, 128, 129] to select the bandwidth in d -dimensional data.

$$h_{dj} = \left(\frac{4}{d+2} \right)^{\frac{1}{d+4}} n^{\frac{-1}{d+4}} \sigma_j \quad (5.7)$$

Here $\frac{4}{d+2} \frac{1}{d+4}$ is named *silverman factor*. σ_j is the standard deviation for the j -th attribute. The rationale behind choosing Silverman's Rule of Thumb is because when using Gaussian multiplicative kernel, the asymptotic mean integrated squared error (AMISE) is minimized [130].

5.2.3 Subspace Sampling Algorithm

Below we describe our algorithm in detail. First we describe how to evaluate subspace quality and then we describe the subspace sampling methods.

5.2.3.1 Subspace Quality

Point Density Normalization: To assure comparison of the density of points in subspaces with different number of dimensions, we normalize the density of a point \mathbf{x} as:

$$\widehat{f}^{\mathbf{S}}(\mathbf{x}) = f^{\mathbf{S}}(\mathbf{x}) \times vol(\mathbf{S}) \quad (5.8)$$

$f^{\mathbf{S}}(\mathbf{x})$ is defined as the density of \mathbf{x} in the subspace \mathbf{S} and $vol(\mathbf{S})$ is defined as the volume of a hypersphere which encloses \mathcal{D} . In our approach, \mathcal{D} is normalized within range $[0, 1]$ for each attribute, and thus the radius of the hypersphere is 0.5. Note that we evaluate $\widehat{f}^{\mathbf{S}}(\mathbf{x})$ *empirically* by multiplying $f^{\mathbf{S}}(\mathbf{x})$ by $vol(\mathbf{S})$. The reason is that this gives preference to larger subspaces and further it gave good results in our experiments.

Subspace Quality: Let C_i denote the i -th cluster, and $|C_i| = n_i$, $\sum_{i=1}^k |C_i| = n$. Define the density of a point \mathbf{x} in the subspace \mathbf{S} , *influenced* by the cluster C_i as:

$$f_{C_i}^{\mathbf{S}}(\mathbf{x}) = \frac{1}{h^d} \sum_{\mathbf{x}_j \in C_i} \kappa\left(\frac{\mathbf{x} - \mathbf{x}_j}{h}\right) \quad (5.9)$$

Thus, the quality measure $Qual(\mathbf{S})$ of the subspace \mathbf{S} is defined as the average value of accumulated normalized density of all points influenced *only* by each point's corresponding cluster:

$$\begin{aligned} Qual(\mathbf{S}) &= \frac{1}{n} \sum_{i \in [1, k]} \sum_{\mathbf{x}_j \in C_i} \widehat{f}_{C_i}^{\mathbf{S}}(\mathbf{x}_j) \\ &= \frac{1}{n} \sum_{i \in [1, k]} \sum_{\mathbf{x}_j \in C_i} f_{C_i}^{\mathbf{S}}(\mathbf{x}_j) \times vol(\mathbf{S}) \end{aligned} \quad (5.10)$$

The kernel function we used in our experiments is Gaussian Kernel.

Definition 9 We define subspace \mathbf{S} as a core subspace if its quality is a local maximum in the search space. More formally, \mathbf{S} is a core subspace iff $Qual(\mathbf{S}) > Qual(\mathbf{S}'), \forall \mathbf{S}' \in N(\mathbf{S})$.

Definition 10 Let $\overline{Qual}(\mathbf{S})$ denote the regularized quality for subspace \mathbf{S} , based on the already detected subspace set \mathbf{S}^* , i.e., $\overline{Qual}(\mathbf{S}) = (1 - sim(\mathbf{S}, \mathbf{S}^*))^\gamma Qual(\mathbf{S})$. The top- k multiview subspaces consist of k core subspaces that are detected sequentially/greedily by searching for the global maximum value $\overline{Qual}(\mathbf{S})$ in the state space.

The factor γ controls the disagreement of redundancy with the already detected subspaces and the unregularized quality $Qual(\mathbf{S})$. In our experiments we set $\gamma = 1$. The Definition 10 states that, the more similar the subspace \mathbf{S} to the already detected subspaces set \mathbf{S}^* , the less the regularized quality for \mathbf{S} .

5.2.3.2 Multiview Subspace Sampling

In the context of multiview subspace sampling, each Markov state can be taken to be a subspace, with transitions allowed, for example, only between parent and child subspaces. Starting from an initial subspace, we can then use Monte Carlo methods to perform random walks in the subspace space to sample the multiview subspaces.

Initial subspace: The initial starting subspace is crucial for good subspace exploration.

We have two strategies to select an initial subspace. The first strategy is random projection, which we denote RP for short, i.e., an attribute is accepted to be in the initial subspace with probability proportional to its weight. Initially we assign each attribute $A_i \in \mathcal{A}$ with weight $w(A_i) = \frac{1}{k}$, where k is an input parameter specified by the user denoting the desired top- k multiview subspaces, and \mathcal{A} denotes the full attributes set. The expected dimensionality of the starting subspace is equal to $\frac{|\mathcal{A}|}{k}$.

The second strategy is to always start from a one dimensional subspace. The probability of the attribute A_i to be chosen as the initial subspace is proportional to its weight $w(A_i)$. We call this strategy RT for short.

Weights update: Initially we assign each attribute $A_i \in \mathcal{A}$ with weight $w(A_i) = \frac{1}{k}$. Then we choose attributes with probability proportional to the attribute weights. Each time when we obtain a new core subspace, we restart to search for a new core subspace based on the already detected subspaces.

After obtaining a new core subspace \mathbf{S}_{new} , the corresponding weights for the chosen attribute $A_i \in \mathbf{S}_{new}$ is updated by:

$$w(A_i) = \frac{1}{k}w(A_i) \quad (5.11)$$

Greedy Local Search (GLS): Suppose we reach a subspace \mathbf{S}_i at i -th step in the markov state space, then the next state is chosen as:

$$\mathbf{S}_{i+1} = \operatorname{argmax}_{\mathbf{S}' \in N(\mathbf{S}_i)} \{ \overline{Qual}(\mathbf{S}') \mid \overline{Qual}(\mathbf{S}') > \overline{Qual}(\mathbf{S}_i) \} \quad (5.12)$$

If \mathbf{S}_{i+1} is empty, i.e., there exists no $\mathbf{S}' \in N(\mathbf{S}_i)$ such that $\overline{Qual}(\mathbf{S}') > \overline{Qual}(\mathbf{S}_i)$, we have reached a core subspace.

Algorithm 2 outlines the main idea of GLS-based exploration for multiview subspace detection. In line 4, we use silverman factor to calculate the optimal bandwidth in the current subspace \mathbf{S}_j . Then in line 5 we use Mean Shift algorithm to get the clusters. In line 6 we use Definition 10 to calculate the regularized quality $\overline{Qual}(\mathbf{S}_j)$ for the current subspace \mathbf{S}_j . If we reach a local core subspace (line 8), we insert it into the multiview subspace set \mathcal{M} (line 9) and update the corresponding weights of the selected attributes in \mathbf{S}_j (line 10).

Simulated Annealing (SA): We also implemented a Simulated Annealing [131] approach to sample subspaces. SA is an optimization method whose goal is to search a good approximation of the global optimum of a target function. SA was inspired from thermodynamic simulation, which involves heating and controlled cooling of a material in order to reduce the defect rate. Here, we sample a subspace \mathbf{S} from the space of all possible subspaces with probability proportional to $\exp \left\{ \frac{\overline{Qual}(\mathbf{S})}{T} \right\}$, i.e.:

$$p(\mathbf{S}) \propto \exp \left\{ \frac{\overline{Qual}(\mathbf{S})}{T} \right\} \quad (5.13)$$

Algorithm 2: Greedy Local Search (GLS)

Input: $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}, k$
Output: Multiview Subspaces \mathcal{M}

- 1 Initialize attribute weights \vec{w} by $w(A_i) = \frac{1}{k}$;
- 2 $\mathbf{S}_1 = \mathbf{Init-Subspace}(\mathcal{A}, \vec{w})$;
- 3 **while** $|\mathcal{M}| \neq k$ **do**
- 4 Calculate $h_{\mathbf{S}_j}$ for $\mathcal{D}(\mathbf{S}_j)$ //Eq. 5.7;
- 5 $\mathcal{C}_{\mathbf{S}_j} = \mathbf{MeanShift}(\mathcal{D}(\mathbf{S}_j), h_{\mathbf{S}_j})$ //Alg. 1;
- 6 $\overline{Qual}(\mathbf{S}_j) = \mathbf{Compute-Qual}(\mathcal{D}(\mathbf{S}_j), \mathcal{C}_{\mathbf{S}_j})$ //Def. 10;
- 7 $\mathbf{S}_{j+1} = \mathbf{Select-Next}(\mathbf{S}_j, \overline{Qual}(\mathbf{S}_j))$ //Eq. 5.12;
- 8 **if** $\mathbf{S}_{j+1} == \emptyset$ **then**
- 9 Insert \mathbf{S}_j in \mathcal{M} ;
- 10 $\mathbf{Update-Attribute-Weights}(\vec{w}, \mathbf{S}_j)$; //Eq. 5.11
- 11 $\mathbf{S}_{j+1} = \mathbf{Init-Subspace}(\mathcal{A}, \vec{w})$;
- 12 **end**
- 13 **end**

where T is the temperature. Let us denote $\Delta\overline{Qual}(\mathbf{S}, \mathbf{S}') = \overline{Qual}(\mathbf{S}) - \overline{Qual}(\mathbf{S}')$. Give a subspace \mathbf{S}_i of the current state, and the already detected subspaces set \mathbf{S}^* , for a random \mathbf{S}' where $\mathbf{S}' \in N(\mathbf{S}_i)$, the acceptance probability is:

$$\mathcal{A}_{SA}(\mathbf{S}_i, \mathbf{S}') = \begin{cases} 1 & \text{if } \overline{Qual}(\mathbf{S}') > \overline{Qual}(\mathbf{S}_i) \\ \exp\left\{\frac{\Delta\overline{Qual}(\mathbf{S}', \mathbf{S}_i)}{T}\right\} & \text{otherwise} \end{cases} \quad (5.14)$$

In Metropolis form:

$$\begin{aligned}
 \mathcal{A}_{SA}(\mathbf{S}_i, \mathbf{S}') &= \min\left\{1, \frac{\exp\left\{\frac{\overline{Qual}(\mathbf{S}')}{T}\right\} |N(\mathbf{S}_i)|}{\exp\left\{\frac{\overline{Qual}(\mathbf{S}_i)}{T}\right\} |N(\mathbf{S}')|}\right\} \\
 &= \min\left\{1, \exp\left\{\frac{\Delta\overline{Qual}(\mathbf{S}', \mathbf{S}_i)}{T}\right\}\right\} \quad (5.15)
 \end{aligned}$$

Cooling Scheme: We adopt a simple and commonly used cooling scheme, i.e., the geometric rule for temperature variation [132]:

$$T_{j+1} = \alpha T_j \quad (5.16)$$

where α is a positive constant smaller but close to 1. Typical value lies between 0.80 to 0.99. The initial temperature T_0 is often chosen to be large enough to enable the algorithm to move off a local minimum but small enough not to move off a global minimum.

Algorithm 3: Simulated Annealing (SA) exploration

Input: $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, k , T_0 , α

Output: Multiview Subspaces \mathcal{M}

```

1  $T = T_0$ ;
2 Initialize attribute weights  $\vec{w}$  by  $w(A_i) = \frac{1}{k}$ ;
3  $\mathbf{S}_1 = \text{Init-Subspace}(\mathcal{A}, \vec{w})$ ;
4 Calculate  $h_{\mathbf{S}_1}$  for  $\mathcal{D}(\mathbf{S}_1)$  //Eq. 5.7;
5  $\mathcal{C}_{\mathbf{S}_1} = \text{MeanShift}(\mathcal{D}(\mathbf{S}_1), h_{\mathbf{S}_1})$  //Alg. 1;
6  $\overline{Qual}(\mathbf{S}_1) = \text{Compute-Qual}(\mathcal{D}(\mathbf{S}_1), \mathcal{C}_{\mathbf{S}_1})$  //Def. 10;
7 while  $|\mathcal{M}| \neq k$  do
8   Choose  $\mathbf{S}_{j+1}$  uniformly from  $N(\mathbf{S}_j)$ ;
9   Accept  $\mathbf{S}_{j+1}$  with probability propto Eq. 5.15;
10  if  $T == 0$  then
11    if  $N(\mathbf{S}_j) == \emptyset$  then
12      Insert  $\mathbf{S}_j$  in  $\mathcal{M}$ ;
13      Update-Attribute-Weights( $\vec{w}, \mathbf{S}_j$ ); //Eq. 5.11
14       $\mathbf{S}_{j+1} = \text{Init-Subspace}(\mathcal{A}, \vec{w})$ ;
15    end
16    if  $\mathbf{S}_{j+1} == \mathbf{S}_j$  //Reject the chosen neighbor then
17      Remove  $\mathbf{S}_{j+1}$  from  $N(\mathbf{S}_j)$ ;
18    end
19  end
20  UpdateT( $T, \alpha$ ) //Eq. 5.16;
21 end
```

Algorithm 3 outlines the main idea of SA-based exploration for multiview subspace detection. Like GLS algorithm, initially we assign each attribute $A_i \in \mathcal{A}$ with weights $w(A_i) = \frac{1}{k}$ (line 1). In line 8, we uniformly choose a neighbor \mathbf{S}_{j+1} from the current subspace \mathbf{S}_j 's neighborhood, and accept it with probability according to Equation 5.15 (line 9). If the chosen neighbor is rejected, then we stay at the current subspace, i.e., $\mathbf{S}_{j+1} = \mathbf{S}_j$. When the current temperature $T = 0$ (line 10), we only accept the neighbor \mathbf{S}_{j+1} whose regularized quality $\overline{Qual}(\mathbf{S}_{j+1})$ is larger than $\overline{Qual}(\mathbf{S}_j)$. If $\overline{Qual}(\mathbf{S}_{j+1}) < \overline{Qual}(\mathbf{S}_j)$, we remove \mathbf{S}_{j+1} from \mathbf{S}_j 's neighborhood (line

17). If all \mathbf{S}_j 's neighbors are removed (line 11), we have reached a core subspace \mathbf{S}_j . In line 20 we update the temperature T according to the geometric rule in Equation 5.16.

5.2.4 Optimizations

Kmeans-based Mean Shift Speedup: The Mean Shift algorithm has one major drawback: it is computationally intensive. The complexity is $O(In^2)$, where I is the average number of iterations to convergence and n is the total number of data points. Fortunately, we can use Kmeans algorithm to overcome this problem. The key observation is that we need only a few points to get the peaks of the density distribution for Mean Shift.

For example, Fig. 5.2 plots two views in our synthetic dataset. There are 7 dimensions in total in the dataset. The first three dimensions a_1, a_2, a_3 form a view (top left plot in Fig. 5.2) and the second three dimensions a_4, a_5, a_6 form another view (top right plot in Fig. 5.2). a_7 is a noise dimension. The bottom two plots in Fig. 5.2 show the corresponding density peaks in two views. Ideally, in the bottom left plot, we only need three points to obtain three peaks, and in bottom right we only need four points to obtain four peaks.

To utilize this observation, we can first apply Kmeans algorithm with k' sufficiently larger than the number of ground truth clusters k as input. Since for KMeans, the time complexity is only $O(kN)$ operations, it can be applied to relatively large datasets. Then for each cluster found by Kmeans, we calculate the center of all points in the cluster. After that, we shift only k' centers by Mean Shift. Note that some of the k' centers might converge to the same peak.

Subspace Quality Cache: To further improve the execution time, we build a hash table to cache the already computed subspace quality. When we reach a previously explored subspace in the search space again, we can quickly look up the candidate quality in the hash table. This step avoids reapplying the clustering algorithm to compute the quality and thus improves the efficiency.

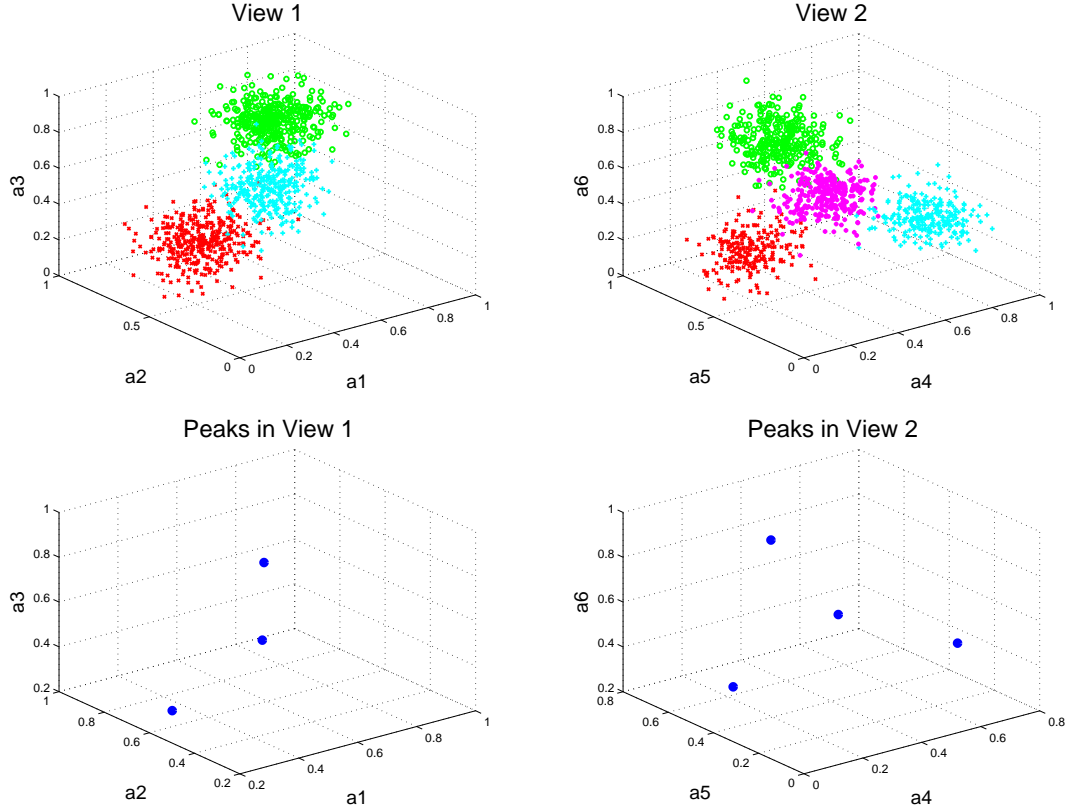


Figure 5.2: A synthetic dataset

5.2.5 Preliminary Experiments

Evaluation Metric: We used E4SC [133] as the evaluation metric. E4SC measure is a recently proposed subspace clustering evaluation measure which successfully meets the requirements of four awareness criteria for measuring the subspace quality: object awareness, subspace awareness, redundancy awareness and identification awareness. E4SC is computed based on transforming subspace clusters into micro-object clusters [134]. The basic idea is that, for a subspace cluster $C = (O, \mathbf{S})$, where O denotes the set of objects and \mathbf{S} is the set of dimensions as before. a micro-object $o_{i,d}$ is constructed for each object $o_i \in O$ and each dimension $d \in \mathbf{S}$. If we use $t(C)$ to represent the micro-objects set of the cluster C , we have $t(C) = \{o_{i,d} \mid o_i \in O(C) \wedge d \in \mathbf{S}(C)\}$.

More formally, let C_r and C_g denote a found cluster and a ground truth cluster, respectively. The recall and precision on the cluster level are defined as:

$$prec(C_r, C_g) = \frac{|t(C_r) \cap t(C_g)|}{|t(C_r)|} \quad (5.17)$$

and

$$recall(C_r, C_g) = \frac{|t(C_r) \cap t(C_g)|}{|t(C_g)|} \quad (5.18)$$

Then, the harmonic mean of precision $prec(C_r, C_g)$ and recall $recall(C_r, C_g)$, i.e., F1 measure on the cluster level is:

$$F1(C_r, C_g) = \frac{2 * prec(C_r, C_g) * recall(C_r, C_g)}{prec(C_r, C_g) + recall(C_r, C_g)} \quad (5.19)$$

Thus, the F1 measure on the clustering level is defined as:

$$F1_{Clus}(G_1, G_2) = \frac{1}{|G_1|} \sum_{C_i \in G_1} \max_{C_j \in G_2} F1(C_i, C_j) \quad (5.20)$$

where G_1, G_2 denote two clusterings.

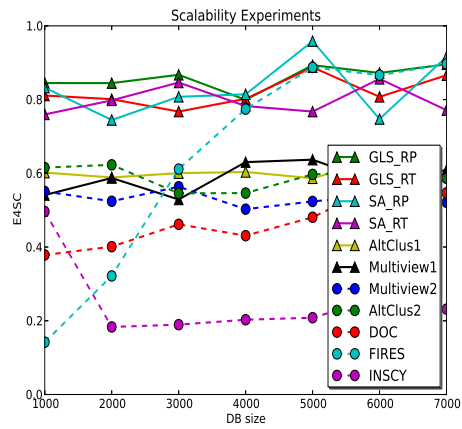
Finally, E4SC measure on two clusterings G_1, G_2 is defined as [133]:

$$E4SC(G_1, G_2) = \frac{2 * F1_{Clus}(G_1, G_2) * F1_{Clus}(G_2, G_1)}{F1_{Clus}(G_1, G_2) + F1_{Clus}(G_2, G_1)} \quad (5.21)$$

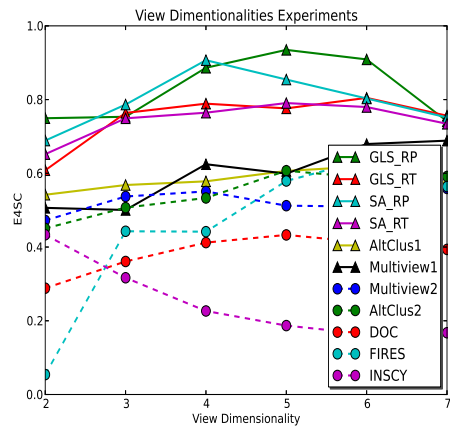
Note that the range of E4SC is in $[0, 1]$ and the higher the E4SC measure, the better the subspace clustering result.

Synthetic Data: Our synthetic data contains multiple views generated by our data generator. The default dataset contains 4 views and each view has 5 dimensions. The number of clusters in each view is 2, 4, 6 and 8, respectively. Each cluster follows a multivariate normal distribution. The overlapping number of dimensions between two views is 0 and noise dimensionality in each view is 2. The dataset has 3000 points by default.

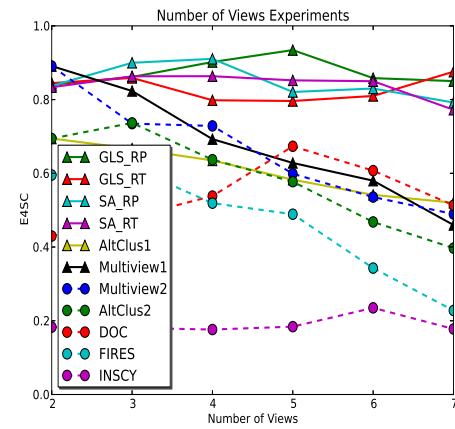
The subspace methods we compared with are DOC [135], FIRES [136] and INSCY [137]. DOC is a monte carlo method, and FIRES and INSCY are all density-based methods. Note that SUBCLU [138] is also a density method. However, SUBCLU is very slow and in most of the experiments it did not finish in one day so we omit its results here. For competing multiview methods we selected Multiview1



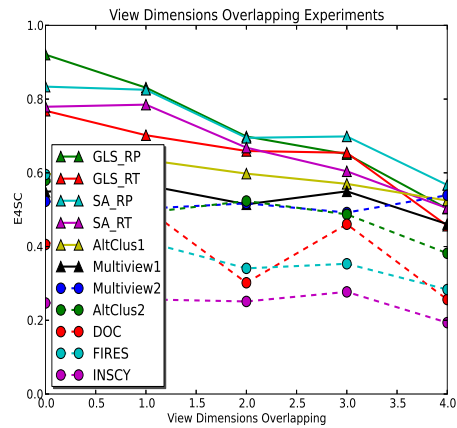
(a) Scalability



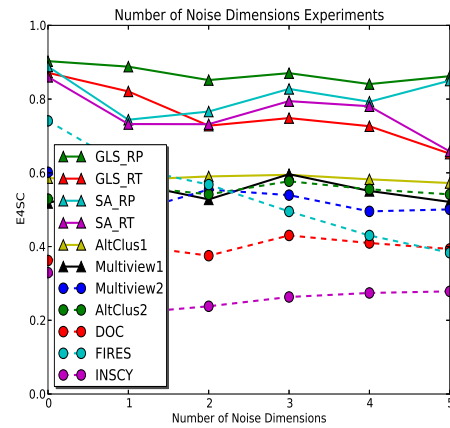
(b) View Dimensionality



(c) Number of Views



(d) Overlapped Views



(e) Noise Dimensionality

Figure 5.3: Experiments on Synthetic Datasets

and Multiview2 proposed in [124] and two variants AltClus1 and AltClus2 of the Alternative Clustering methods proposed in [139].

Since DOC is a monte carlo method, we ran DOC as many times as the number of ground truth views. Then we combined the detected subspace clusters by DOC from all runs as the final result. Another important thing to note is that four competing multiview methods, Multiview1, Multiview2, AltClus1 and AltClus2 do not detect subspaces! The output of the algorithms does not have information about the detected clusters' subspaces. So to ensure fair comparison, we ignored the subspaces during the evaluation. In other words, we evaluated only the point groupings. That is, we assume all clusters are located in the "correct" subspaces. This way the competing multiview methods have a huge advantage, however, as we will see in the experiments, the results show that the competing multiview methods still have low quality.

Since we have two search strategies Greedy Local Search (GLS) and Simulated Annealing (SA), and we have two initial subspace starting strategies Random Projection (RP) and starting from a one dimensional subspace (RT), we denote our methods GLS_RP, GLS_RT, SA_RP and SA_RT, respectively. Since our methods are randomized, we repeat each method 10 times, and report the average. For SA_RP, SA_RT, we set initial temperature $T_0 = 50$ and cooling rate $\alpha = 0.8$. We set the initial number of clusters in Kmeans as 15, for MeanShift algorithm.

Figure 5.3 shows the E4SC values for each algorithm. We conducted 5 groups of experiments, i.e., scalability (the number of total instances), view dimensionality (the number of attributes in each view), number of views (the number of total views in the dataset), overlapped views (the number of overlapped attributes for every two views) and noise dimensions (the number of irrelevant dimensions for the views). For each group of experiments, we generated the dataset by varying one parameter which is shown in x-axis in Figure 5.3 while keeping other parameters fixed.

From Figure 5.3, we observe that our four methods: GLS_RP, GLS_RT, SA_RP and SA_RT are more effective in detecting multiview subspaces than other methods on the synthetic data. Thus, these preliminary results show that our methods can detect multiple views with different subspace dimensionality.

Nevertheless, many challenges still remain. Firstly, for Equation 5.8, we need to find a more reasonable form of normalized density that has more theoretical evidence. Currently, we just evaluated $\hat{f}^{\mathbf{S}}(\mathbf{x})$ *empirically* by multiplying $f^{\mathbf{S}}(\mathbf{x})$ by $vol(\mathbf{S})$, where $vol(\mathbf{S})$ is defined as the volume of a hypersphere which encloses \mathcal{D} . It is hard to guarantee that Equation 5.8 is a good estimation that is suitable for other cases. Secondly, although we applied Kmeans-based Mean Shift and subspace quality cache techniques for optimization, efficiency still remains an issue. Thirdly, the mean shift clustering algorithm has two main drawbacks, i.e., it is computationally intensive and it is also sensitive to noise or outliers. Thus, can we find an alternative method that can serve as a good substitute for Mean Shift to estimate the quality of the subspace? How to successfully tackle these issues is our future research direction.

LITERATURE CITED

- [1] M. Al Hasan, “Mining interesting subgraphs by output space sampling,” Ph.D. dissertation, Rensselaer Polytechnic Institute, Troy, NY, 2009.
- [2] M. J. Zaki and C.-J. Hsiao, “CHARM: An efficient algorithm for closed itemset mining,” in *Proceedings of the 2nd SIAM International Conference on Data Mining*, Arlington, VA, 2002, pp. 457–73.
- [3] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Verkamo, “Fast discovery of association rules,” *Advances in Knowledge Discovery and Data Mining*, vol. 12, no. 1, pp. 307–328, February 1996.
- [4] R. Agrawal and R. Srikant, “Fast algorithms for mining association rules,” in *Proceedings of the 20th International Conference on Very Large Data Bases*, San Francisco, CA, 1994, pp. 487–499.
- [5] L. Ward, “Partially ordered topological spaces,” *Proceedings of the American Mathematical Society*, vol. 5, no. 1, pp. 144–161, February 1954.
- [6] J. Han, H. Cheng, D. Xin, and X. Yan, “Frequent pattern mining: Current status and future directions,” *Data Mining and Knowledge Discovery*, vol. 15, no. 1, pp. 55–86, August 2007.
- [7] S. Bandyopadhyay, U. Maulik, L. B. Holder, and D. J. Cook, *Advanced Methods for Knowledge Discovery from Complex Data*. Secaucus, NJ: Springer-Verlag New York Inc., 2005.
- [8] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, Eds., *Advances in Knowledge Discovery and Data Mining*. Menlo Park, CA: American Association for Artificial Intelligence, 1996.
- [9] S. Brin, R. Motwani, J. Ullman, and S. Tsur, “Dynamic itemset counting and implication rules for market basket data,” in *Proceedings of the 16th SIGMOD International Conference on Management of Data*, New York, NY, 1997, pp. 255–264.
- [10] J. Han, J. Pei, and Y. Yin, “Mining frequent patterns without candidate generation,” in *Proceedings of the 19th SIGMOD International Conference on Management of Data*, New York, NY, 2000, pp. 1–12.
- [11] A. Savasere, E. Omiecinski, and S. B. Navathe, “An efficient algorithm for mining association rules in large databases,” in *Proceedings of the 21st International Conference on Very Large Data Bases*, San Francisco, CA, 1995, pp. 432–444.

- [12] M. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, “New algorithms for fast discovery of association rules,” in *Proceedings of the 3rd SIGKDD International Conference on Knowledge Discovery and Data Mining*, Newport Beach, CA, 1997, pp. 326–335.
- [13] M. Zaki and K. Gouda, “Fast vertical mining using diffsets,” in *Proceedings of the 9th SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington, D.C., 2003, pp. 326–335.
- [14] R. Agrawal and R. Srikant, “Mining sequential patterns,” in *Proceedings of the 11th International Conference on Data Engineering*, Washington, D.C., 1995, pp. 3–14.
- [15] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu, “Sequential pattern mining using a bitmap representation,” in *Proceedings of the 8th SIGKDD International Conference on Knowledge Discovery and Data Mining*, Alberta, Canada, 2002, pp. 429–435.
- [16] H. Mannila and H. Toivonen, “Discovering generalized episodes using minimal occurrences,” in *Proceedings of the 2nd International Conference on Knowledge Discovery in Databases and Data Mining*, Portland, OR, 1996, pp. 146–151.
- [17] H. Mannila, H. Toivonen, and A. Verkamo, “Discovering frequent episodes in sequences extended abstract,” in *Proceedings of the 1st SIGKDD International Conference on Knowledge Discovery and Data Mining*, Montreal, Canada, 1995, pp. 210–215.
- [18] J. Han, J. Pei, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu, “PrefixSpan: Mining sequential patterns efficiently by prefix projected pattern growth,” in *Proceedings of the 17th International Conference on Data Engineering*, Heidelberg, Germany, 2001, pp. 215–224.
- [19] R. Srikant and R. Agrawal, *Mining Sequential Patterns: Generalizations and Performance Improvements*. London, UK: Springer-Verlag, 1996.
- [20] M. Zaki, “SPADE: An efficient algorithm for mining frequent sequences,” *Machine Learning*, vol. 42, no. 1, pp. 31–60, January 2001.
- [21] M. Garofalakis, R. Rastogi, and K. Shim, “SPIRIT: Sequential pattern mining with regular expression constraints,” in *Proceedings of the 25th International Conference on Very Large Data Bases*, Edinburgh, Scotland, 1999, pp. 223–234.
- [22] M. Zaki, “Sequence mining in categorical domains: Incorporating constraints,” in *Proceedings of the 9th International Conference on Information and Knowledge Management*, McLean, VA, 2000, pp. 422–429.

- [23] D. Cook and L. Holder, “Substructure discovery using minimum description length and background knowledge,” *Journal of Artificial Intelligence Research*, vol. 1, no. 1, pp. 231–255, August 1994.
- [24] L. Dehaspe, H. Toivonen, and R. King, “Finding frequent substructures in chemical compounds,” in *Proceedings of the 4th SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, 1998, pp. 30–36.
- [25] J. Huan, W. Wang, and J. Prins, “Efficient mining of frequent subgraphs in the presence of isomorphism,” in *Proceedings of the 3rd International Conference on Data Mining*, Melbourne, FL, 2003, pp. 549–552.
- [26] A. Inokuchi, T. Washio, and H. Motoda, “An apriori-based algorithm for mining frequent substructures from graph data,” in *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, London, UK, 2000, pp. 13–23.
- [27] S. Kramer, L. De Raedt, and C. Helma, “Molecular feature mining in HIV data,” in *Proceedings of the 7th SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, CA, 2001, pp. 136–143.
- [28] S. Nijssen and J. Kok, “A quickstart in frequent structure mining can make a difference,” in *Proceedings of the 10th SIGKDD International Conference on Knowledge Discovery and Data Mining*, Seattle, WA, 2004, pp. 647–652.
- [29] X. Yan and J. Han, “GSPAN: Graph-based substructure pattern mining,” in *Proceedings of the 2nd International Conference on Data Mining*, Washington, D.C., 2002, pp. 721–724.
- [30] D. Burdick, M. Calimlim, J. Flannick, J. Gehrke, and T. Yiu, “MAFIA: A maximal frequent itemset algorithm,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 11, pp. 1490–1504, November 2005.
- [31] R. Bayardo Jr, “Efficiently mining long patterns from databases,” in *Proceedings of the 17th SIGMOD International Conference on Management of Data*, Seattle, WA, 1998, pp. 85–93.
- [32] B. Ganter, G. Stumme, and R. Wille, Eds., *Formal Concept Analysis: Foundations and Applications*. Berlin, Heidelberg: Springer-Verlag, 2005.
- [33] Y. Bastide, R. Taouil, N. Pasquier, G. Stumme, and L. Lakhal, “Mining frequent patterns with counting inference,” *ACM SIGKDD Explorations Newsletter*, vol. 2, no. 2, pp. 66–75, December 2000.

- [34] B. Goethals and M. Zaki, “Advances in frequent itemset mining implementations: Report on FIMI’03,” *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 109–117, June 2004.
- [35] M. Zaki and N. Ramakrishnan, “Reasoning about sets using redescription mining,” in *Proceedings of the 11th SIGKDD International Conference on Knowledge Discovery and Data Mining*, Chicago, IL, 2005, pp. 364–373.
- [36] G. Dong, C. Jiang, J. Pei, J. Li, and L. Wong, “Mining succinct systems of minimal generators of formal concepts,” in *Proceedings of the 10th International Conference on Database Systems for Advanced Applications*, Beijing, China, 2005, pp. 175–187.
- [37] Y. Shima, S. Mitsuishi, K. Hirata, and M. Harao, “Extracting minimal and closed monotone DNF formulas,” in *Proceedings of the 7th International Conference on Discovery Science*, Padova, Italy, 2004, pp. 298–305.
- [38] L. Zhao, M. Zaki, and N. Ramakrishnan, “BLOSOM: A framework for mining arbitrary boolean expressions,” in *Proceedings of the 12th SIGKDD International Conference on Knowledge Discovery and Data Mining*, Philadelphia, PA, 2006, pp. 827–832.
- [39] J. Wang and J. Han, “BIDE: Efficient mining of frequent closed sequences,” in *Proceedings of the 20th International Conference on Data Engineering*, Boston, MA, 2004, pp. 79–90.
- [40] J. Balcázar and G. Garriga, “Horn axiomatizations for sequential data,” *Theoretical Computer Science*, vol. 371, no. 3, pp. 247–264, March 2007.
- [41] X. Yan and J. Han, “CloseGraph: Mining closed frequent graph patterns,” in *Proceedings of the 9th SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington, D.C., 2003, pp. 286–295.
- [42] J. Wang, Z. Zeng, and L. Zhou, “CLAN: An algorithm for mining closed cliques from large dense graph databases,” in *Proceedings of the 22nd International Conference on Data Engineering*, Atlanta, GA, 2006, pp. 73–73.
- [43] Y. Chi, Y. Xia, Y. Yang, and R. Muntz, “Mining closed and maximal frequent subtrees from databases of labeled rooted trees,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 2, pp. 190–202, February 2005.
- [44] T. T. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1990.

- [45] B. Scholkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA: MIT Press, 2001.
- [46] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York, NY: Springer-Verlag New York Inc, 1995.
- [47] T. Gärtner, P. Flach, and S. Wrobel, “On graph kernels: Hardness results and efficient alternatives,” in *Proceedings of the 16th Annual Conference on Computational Learning Theory*, Washington, D.C., 2003, pp. 129–143.
- [48] H. Kashima, K. Tsuda, and A. Inokuchi, “Marginalized kernels between labeled graphs,” in *Proceedings of the 20th International Conference on Machine Learning*, Washington, D.C., 2003, pp. 321–328.
- [49] K. M. Borgwardt and H.-P. Kriegel, “Shortest-path kernels on graphs,” in *Proceedings of the 5th IEEE International Conference on Data Mining*, Houston, TX, 2005, pp. 74–81.
- [50] T. Horváth, T. Gärtner, and S. Wrobel, “Cyclic pattern kernels for predictive graph mining,” in *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Seattle, WA, 2004, pp. 158–167.
- [51] J. Ramon and T. Gärtner, “Expressivity versus efficiency of graphs kernels,” in *Proceedings of the 1st International Workshop on Mining Graphs, Trees and Sequences*, Cavtat-Dubrovnik, Croatia, 2003, pp. 65–74.
- [52] L. Ralaivola, S. J. Swamidass, H. Saigo, and P. Baldi, “Graph kernels for chemical informatics,” *Neural Networks*, vol. 18, no. 8, pp. 1093–1110, June 2005.
- [53] N. Shervashidze and K. M. Borgwardt, “Fast subtree kernels on graphs,” in *Proceedings of the 26th Advances in Neural Information Processing Systems*, Vancouver, Canada, 2009, pp. 1660–1668.
- [54] N. Shervashidze, S. V. Vishwanathan, T. H. Petri, K. Mehlhorn, and K. M. Borgwardt, “Efficient graphlet kernels for large graph comparison,” in *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*, Clearwater Beach, FL, 2009, pp. 488–495.
- [55] I. R. Kondor, N. Shervashidze, and K. M. Borgwardt, “The graphlet spectrum,” in *Proceedings of the 26th International Conference on Machine Learning*, Montreal, Canada, 2009, pp. 529–536.
- [56] M. Thoma, H. Cheng, A. Gretton, J. Han, H.-P. Kriegel, A. Smola, L. Song, P. S. Yu, X. Yan, and K. M. Borgwardt, “Discriminative frequent subgraph

- mining with optimality guarantees,” *Statistical Analysis and Data Mining*, vol. 3, no. 5, pp. 302–318, October 2010.
- [57] G. Li and M. J. Zaki, “Sampling minimal frequent boolean (DNF) patterns,” in *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Beijing, China, 2012, pp. 87–95.
- [58] A. Frank and A. Asuncion, “UCI machine learning repository,” <http://archive.ics.uci.edu/ml>, (Date Last Accessed June 25, 2013).
- [59] G. Li, M. Semerci, B. Yener, and M. J. Zaki, “Effective graph classification based on topological and label attributes,” *Statistical Analysis and Data Mining*, vol. 5, no. 4, pp. 265–283, August 2012.
- [60] A. A. Nanavati, K. P. Chitrapura, S. Joshi, and R. Krishnapuram, “Mining generalised disjunctive association rules,” in *Proceedings of the 10th International Conference on Information and Knowledge Management*, Atlanta, GA, 2001, pp. 482–489.
- [61] E. Loekito and J. Bailey, “Fast mining of high dimensional expressive contrast patterns using zero-suppressed binary decision diagrams,” in *Proceedings of the 12th SIGKDD International Conference on Knowledge Discovery and Data Mining*, Philadelphia, PA, 2006, pp. 307–316.
- [62] H. Toivonen, “Sampling large databases for association rules,” in *Proceedings of the 22th International Conference on Very Large Data Bases*, San Francisco, CA, 1996, pp. 134–145.
- [63] B. Chen, P. Haas, and P. Scheuermann, “A new two-phase sampling based algorithm for discovering association rules,” in *Proceedings of the 8th SIGKDD International Conference on Knowledge Discovery and Data Mining*, Alberta, Canada, 2002, pp. 462–468.
- [64] M. Zaki, S. Parthasarathy, W. Li, and M. Ogihara, “Evaluation of sampling for data mining of association rules,” in *Proceedings of the 7th International Workshop Research Issues in Data Engineering*, Washington, D.C., 1997, pp. 42–50.
- [65] V. Chakaravarthy, V. Pandit, and Y. Sabharwal, “Analysis of sampling techniques for association rule mining,” in *Proceedings of the 12th International Conference on Database Theory*, St. Petersburg, Russia, 2009, pp. 276–283.
- [66] M. Boley and H. Grosskreutz, “Approximating the number of frequent sets in dense data,” *Knowledge and Information Systems*, vol. 21, no. 1, pp. 65–89, October 2009.

- [67] M. Boley, C. Lucchese, D. Paurat, and T. Gärtner, “Direct local pattern sampling by efficient two-step random procedures,” in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Diego, CA, 2011, pp. 582–590.
- [68] D. Gunopulos, H. Mannila, and S. Saluja, “Discovering all most specific sentences by randomized algorithms,” in *Proceedings of the 6th International Conference on Database Theory*, Delphi, Greece, 1997, pp. 215–229.
- [69] M. Boley, T. Gärtner, H. Grosskreutz, and I. Fraunhofer, “Formal concept sampling for counting and threshold-free local pattern mining,” in *Proceedings of the 10th SIAM International Conference on Data Mining*, Columbus, OH, 2010, pp. 177–188.
- [70] J. Leskovec and C. Faloutsos, “Sampling from large graphs,” in *Proceedings of the 12th SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Jose, CA, 2006, pp. 631–636.
- [71] B. Ribeiro and D. Towsley, “Estimating and sampling graphs with multidimensional random walks,” in *Proceedings of the 10th Annual Conference on Internet Measurement*, Melbourne, Australia, 2010, pp. 390–403.
- [72] C. Hubler, H. Kriegel, K. Borgwardt, and Z. Ghahramani, “Metropolis algorithms for representative subgraph sampling,” in *Proceedings of the 8th International Conference on Data Mining*, Pisa, Italy, 2008, pp. 283–292.
- [73] A. Mislove, M. Marcon, K. Gummadi, P. Druschel, and B. Bhattacharjee, “Measurement and analysis of online social networks,” in *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, San Diego, CA, 2007, pp. 29–42.
- [74] Y. Ahn, S. Han, H. Kwak, S. Moon, and H. Jeong, “Analysis of topological characteristics of huge online social networking services,” in *Proceedings of the 16th International Conference on World Wide Web*, Alberta, Canada, 2007, pp. 835–844.
- [75] C. Wilson, B. Boe, A. Sala, K. Puttaswamy, and B. Zhao, “User interactions in social networks and their implications,” in *Proceedings of the 4th ACM European Conference on Computer Systems*, Nuremberg, Germany, 2009, pp. 205–218.
- [76] H. Kwak, C. Lee, H. Park, and S. Moon, “What is Twitter, a social network or a news media?” in *Proceedings of the 19th International Conference on World Wide Web*, Raleigh, NC, 2010, pp. 591–600.

- [77] M. Al Hasan and M. Zaki, "Output space sampling for graph patterns," *VLDB Journal*, vol. 2, no. 1, pp. 730–741, August 2009.
- [78] M. Al Hasan, V. Chaoji, S. Salem, J. Besson, and M. Zaki, "ORIGAMI: Mining representative orthogonal graph patterns," in *Proceedings of the 10th International Conference on Data Mining*, Omaha, NE, June 2007, pp. 153–162.
- [79] V. Chaoji, M. Al Hasan, S. Salem, J. Besson, and M. J Zaki, "ORIGAMI: A novel and effective approach for mining representative orthogonal graph patterns," *Statistical Analysis and Data Mining*, vol. 1, no. 2, pp. 67–84, June 2008.
- [80] M. Al Hasan and M. Zaki, "MUSK: Uniform sampling of k maximal patterns," in *Proceedings of the 9th SIAM International Conference on Data Mining*, Sparks, NV, 2009, pp. 650–661.
- [81] A. Rasti, M. Torkjazi, R. Rejaie, N. Duffield, W. Willinger, and D. Stutzbach, "Respondent-driven sampling for characterizing unstructured overlays," in *Proceedings of the 28th Conference on Computer Communications*, Rio de Janeiro, Brazil, 2009, pp. 2701–2705.
- [82] M. Hansen and W. Hurwitz, "On the theory of sampling from finite populations," *The Annals of Mathematical Statistics*, vol. 14, no. 4, pp. 333–362, December 1943.
- [83] W. Gilks, S. Richardson, and D. Spiegelhalter, *Markov Chain Monte Carlo in Practice*. London, UK: Chapman & Hall/CRC, 1996.
- [84] M. Gjoka, M. Kurant, C. Butts, and A. Markopoulou, "Walking in Facebook: A case study of unbiased sampling of OSNs," in *Proceedings of the 29th Conference on Computer Communications*, San Diego, CA, 2010, pp. 1–9.
- [85] J. Wang and G. Karypis, "HARMONY: Efficiently mining the best rules for classification," in *Proceedings of the 5th SIAM International Conference on Data Mining*, Newport Beach, CA, 2005, pp. 205–216.
- [86] M. van Leeuwen, J. Vreeken, and A. Siebes, "Compression picks item sets that matter," in *Proceedings of the 10th European Conference on Principle and Practice of Knowledge Discovery in Databases*, Berlin, Germany, 2006, pp. 585–592.
- [87] X. Zhang, G. Dong, and R. Kotagiri, "Information-based classification by aggregating emerging patterns," in *Proceedings of the 2nd International Conference on Intelligent Data Engineering and Automated Learning, Data Mining, Financial Engineering, and Intelligent Agents*, London, UK, 2000, pp. 48–53.

- [88] G. Dong and J. Li, “Efficient mining of emerging patterns: Discovering trends and differences,” in *Proceedings of the 5th SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Diego, CA, 1999, pp. 43–52.
- [89] E. Loekito and J. Bailey, “Using highly expressive contrast patterns for classification: Is it worthwhile?” in *Proceedings of the 13th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, Bangkok, Thailand, 2009, pp. 483–490.
- [90] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. V. N. Vishwanathan, A. J. Smola, and H.-P. Kriegel, “Protein function prediction via graph kernels,” *Bioinformatics*, vol. 21, no. 1, pp. 47–56, January 2005.
- [91] S. V. N. Vishwanathan, K. M. Borgwardt, and N. N. Schraudolph, “Fast computation of graph kernels,” in *Proceedings of the 23rd Advances in Neural Information Processing Systems*, Vancouver, Canada, 2006, pp. 1–12.
- [92] P. Mahè and J.-P. Vert, “Graph kernels based on tree patterns for molecules,” *Machine Learning*, vol. 75, no. 1, pp. 3–35, April 2009.
- [93] I. R. Kondor and J. Lafferty, “Diffusion kernels on graphs and other discrete structures,” in *Proceedings of the 19th International Conference on Machine Learning*, San Francisco, CA, 2002, pp. 315–322.
- [94] P. Mahè, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert, “Extensions of marginalized graph kernels,” in *Proceedings of the 21st International Conference on Machine Learning*, Banff, Canada, 2004, pp. 552–559.
- [95] S. V. N. Vishwanathan, N. N. Schraudolph, I. R. Kondor, and K. M. Borgwardt, “Graph kernels,” *Journal of Machine Learning Research*, vol. 11, no. 1, pp. 1201–1242, April 2010.
- [96] U. Fayyad and K. Irani, “Multi-interval discretization of continuous-valued attributes for classification learning,” in *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Chambery, France, 1993, pp. 1022–1029.
- [97] T. Curk, J. Demsar, Q. Xu, G. Leban, U. Petrovic, I. Bratko, G. Shaulsky, and B. Zupan, “Microarray data mining with visual programming,” *Bioinformatics*, vol. 21, no. 3, pp. 396–398, February 2005.
- [98] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and other Kernel-based Learning Methods*. New York, NY: Cambridge University Press, 2000.
- [99] I. Guyon, B. Boser, and V. Vapnik, “Automatic capacity tuning of very large VC-dimension classifiers,” *Advances in Neural Information Processing Systems*, vol. 1, no. 1, pp. 147–147, November 1993.

- [100] C. Chang and C. Lin, “LIBSVM: A library for support vector machines,” *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 3, pp. 1–39, April 2011.
- [101] M. Cowles and B. Carlin, “Markov chain monte carlo convergence diagnostics: A comparative review,” *Journal of the American Statistical Association*, vol. 91, no. 1, pp. 883–904, June 1996.
- [102] T. Joachims, T. Hofmann, Y. Yue, and C.-N. Yu, “Predicting structured objects with support vector machines,” *Communications of the ACM*, vol. 52, no. 11, pp. 97–104, November 2009.
- [103] C. Bilgin, C. Demir, C. Nagi, and B. Yener, “Cell graph mining for breast tissue modeling and classification,” in *Proceedings of the 29th International Conference in Medicine and Biology Society*, Hingham, MA, 2007, pp. 5311–5314.
- [104] P. Mahé and J.-P. Vert, “Graph kernels based on tree patterns for molecules,” *Machine Learning*, vol. 75, no. 1, pp. 3–35, April 2009.
- [105] C. D. Meyer, Ed., *Matrix Analysis and Applied Linear Algebra*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2000.
- [106] J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2000.
- [107] S. Nijssen and J. Kok, “A quickstart in frequent structure mining can make a difference,” in *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Seattle, WA, 2004, pp. 647–652.
- [108] E. Jones, T. Oliphant, and P. Peterson, “SciPy: Open source scientific tools for Python,” <http://www.scipy.org/>, (Date Last Accessed June 25, 2013).
- [109] A. A. Hagberg, D. A. Schult, and P. J. Swart, “Exploring network structure, dynamics, and function using NetworkX,” in *Proceedings of the 7th Python in Science Conference*, Pasadena, CA, 2008, pp. 11–15.
- [110] D. Linke, “Python performance,” <http://wiki.scipy.org/PerformancePython>, (Date Last Accessed June 25, 2013).
- [111] C.-C. Chang and C.-J. Lin, “LIBSVM: A library for support vector machines,” *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 3, pp. 1–27, April 2011.

- [112] A. Debnath, R. L. de Compadre, G. Debnath, A. Shusterman, and C. Hansch, “The structure activity relationship of mutagenic aromatic nitro compounds,” *Journal of Medicinal Chemistry*, vol. 34, no. 1, pp. 786–797, February 1991.
- [113] N. Wale and G. Karypis, “Comparison of descriptor spaces for chemical compound retrieval and classification,” in *Proceedings of the 6th IEEE International Conference on Data Mining*, Hong Kong, China, 2006, pp. 347–375.
- [114] C. Helma, R. King, S. Kramer, and A. Srinivasan, “The predictive toxicology challenge,” <http://www.predictive-toxicology.org/ptc>, (Date Last Accessed June 25, 2013).
- [115] P. Dobson and A. J. Doig, “Distinguishing enzyme structures from non-enzymes without alignments,” *Journal of Molecular Biology*, vol. 330, no. 4, pp. 771–783, July 2003.
- [116] I. Sillitoe, A. L. Cuff, B. H. Dessailly, N. L. Dawson, N. Furnham, D. Lee, J. G. Lees, T. E. Lewis, R. A. Studer, and R. Rentzsch, “New functional families (FunFams) in CATH to improve the mapping of conserved functional sites to 3D structures,” *Nucleic Acids Research*, vol. 41, no. 1, pp. 490–498, November 2013.
- [117] C. C. Bilgin, P. Bullough, G. E. Plopper, and B. Yener, “ECM-aware cell graph mining for bone tissue modeling and classification,” *Data Mining and Knowledge Discovery*, vol. 20, no. 3, pp. 416–438, May 2010.
- [118] C. Demir, S. H. Gultekin, and B. Yener, “Learning the topological properties of brain tumors,” *IEEE Transactions on Computational Biology and Bioinformatics*, vol. 2, no. 3, pp. 262–270, July 2005.
- [119] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, “Gene selection for cancer classification using support vector machines,” *Machine Learning*, vol. 46, no. 1, pp. 389–422, January 2002.
- [120] L. Lovász and K. Vesztegombi, *Geometric Representations of Graphs*. New York, NY: Springer-Verlag, 1999.
- [121] U. Brandes and D. Fleischer, *Centrality Measures based on Current Flow*. Berlin, Heidelberg: Springer-Verlag, 2005.
- [122] M. Newman, *Networks: An Introduction*. New York, NY: Oxford University Press, Inc., 2010.
- [123] S. Günnemann, I. Färber, and T. Seidl, “Multi-view clustering using mixture models in subspace projections,” in *Proceedings of the 18th ACM SIGKDD*

International Conference on Knowledge Discovery and Data Mining, Beijing, China, pp. 132–140.

- [124] Y. Cui, X. Fern, and J. Dy, “Non-redundant multi-view clustering via orthogonalization,” in *Proceedings of the 7th International Conference on Data Mining*, Omaha, NE, 2007, pp. 133–142.
- [125] D. Comaniciu and P. Meer, “Mean shift: A robust approach toward feature space analysis,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, May 2002.
- [126] A. Hinneburg and H. Gabriel, “DENCLUE 2.0: Fast clustering based on kernel density estimation,” in *Proceedings of the 7th International Conference on Intelligent Data Analysis*, Ljubljana, Slovenia, 2007, pp. 70–80.
- [127] B. Silverman, *Density Estimation for Statistics and Data Analysis*. London, UK: Chapman & Hall/CRC, 1986.
- [128] W. Härdle, *Nonparametric and Semiparametric Models*. Berlin, Heidelberg: Springer-Verlag, 2004.
- [129] D. Bashtannyk and R. Hyndman, “Bandwidth selection for kernel conditional density estimation,” *Computational Statistics and Data Analysis*, vol. 36, no. 3, pp. 279–298, May 2001.
- [130] D. Scott, *Multivariate Density Estimation: Theory, Practice, and Visualization*. New York, NY: John Wiley & Sons, 1992.
- [131] S. Kirkpatrick, D. G. Jr., and M. P. Vecchi, “Optimization by simulated annealing,” *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.
- [132] P. J. M. Laarhoven and E. H. L. Aarts, Eds., *Simulated Annealing: Theory and Applications*. Norwell, MA: Kluwer Academic Publishers, 1987.
- [133] S. Günnemann, I. Färber, E. Müller, I. Assent, and T. Seidl, “External evaluation measures for subspace clustering,” in *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, Glasgow, UK, 2011, pp. 1363–1372.
- [134] A. Patrikainen and M. Meila, “Comparing subspace clusterings,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 7, pp. 902–916, July 2006.
- [135] C. Procopiuc, M. Jones, P. Agarwal, and T. Murali, “A monte carlo algorithm for fast projective clustering,” in *Proceedings of the 21st SIGMOD International Conference on Management of Data*, Madison, WI, 2002, pp. 418–427.

- [136] H. Kriegel, P. Kroger, M. Renz, and S. Wurst, “A generic framework for efficient subspace clustering of high-dimensional data,” in *Proceedings of the 5th IEEE International Conference on Data Mining*, Washington, D.C., 2005, pp. 250–257.
- [137] I. Assent, R. Krieger, E. Muller, and T. Seidl, “INSCY: Indexing subspace clusters with in-process-removal of redundancy,” in *Proceedings of the 8th IEEE International Conference on Data Mining*, Pisa, Italy, 2008, pp. 719–724.
- [138] K. Kailing, H. Kriegel, and P. Kröger, “Density-connected subspace clustering for high-dimensional data,” in *Proceedings of the 4th SIAM International Conference on Data Mining*, Orlando, FL, 2004, pp. 246–257.
- [139] Z. Qi and I. Davidson, “A principled and flexible framework for finding alternative clusterings,” in *Proceedings of the 15th SIGKDD International Conference on Knowledge Discovery and Data Mining*, Paris, France, 2009, pp. 717–726.